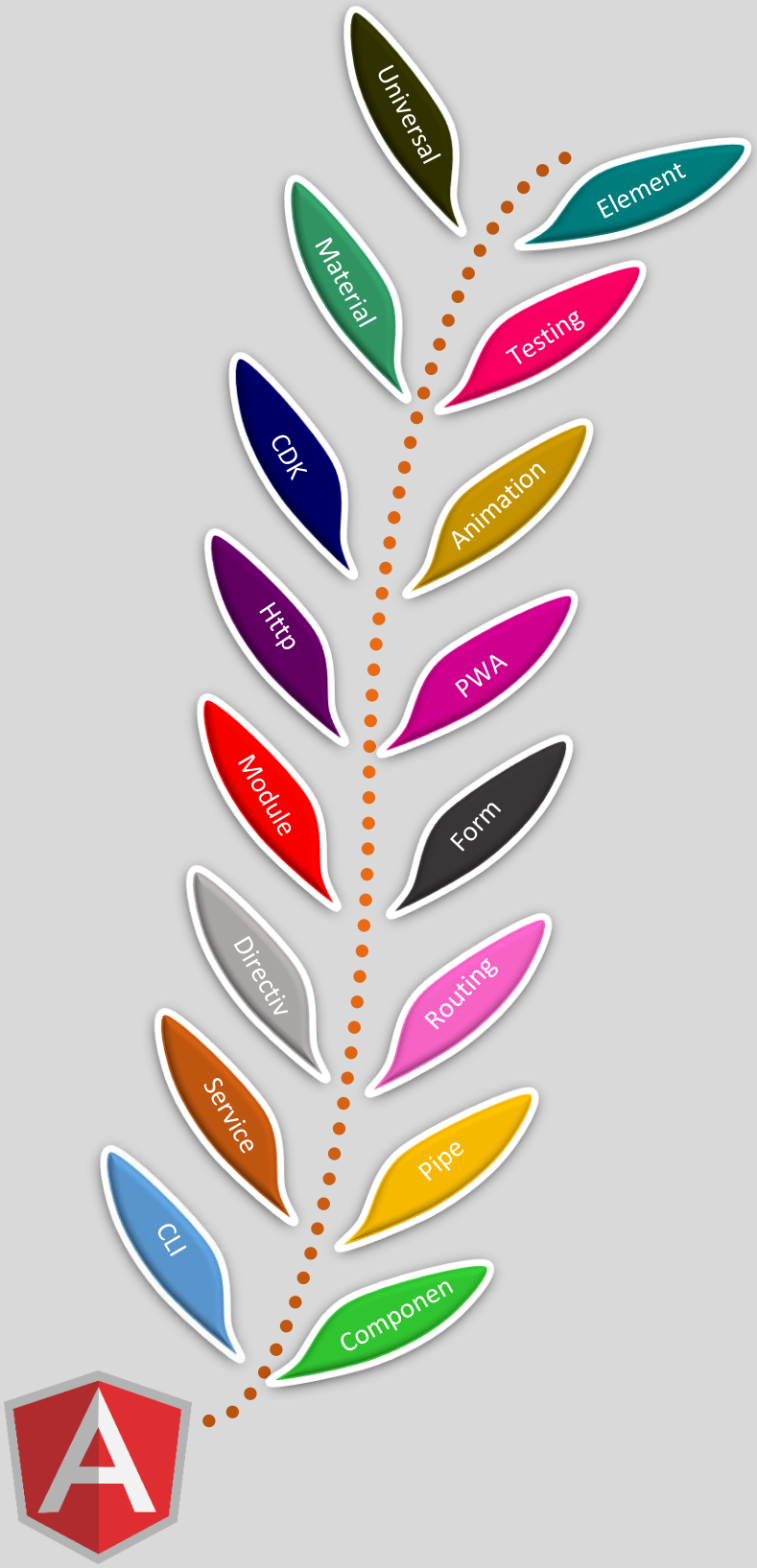
سلسلة تعلم Angular بالعربي



5

Angular Forms

فيصل الفهد

Angular Forms

الكتاب الخامس من

سلسلة تعلم Angular بالعربي

تم انتاج هذا العمل في عام

٢٠٢٠

المؤلف

فيصل الفهد

<u>إهداء</u>

إلى أطهر قلبين في حياتي... والديَّ العزيزين.

إلى من شاركني السرَّاء والضراء، ولم أرها عابسة يومًا..... زوجتي المخلصة.

إلى من أتشوَّق لأن أرى مستقبلهما المشرق بإذن الله..... ابنائي وبناتي.

إلى جميع متلهف للعلم ويتوق للمعرفة

أهديكم هذا الكتاب المتواضع، وأدعو الله أن يحوز إعجابكم.

المؤلف

جدول المحتويات

مقدمة الكتاب

يقدم لنا angular forms مع النماذج طريقتين للتعامل معها الأولى اسمها angular template driven forms واختصاراً angular TDF وهي مناسبة للنماذج الثابتة الغير ديناميكية كما انه التحقق من الصحة validation فيها هو عبارة عن مجموعة من الدايركتيف والخصائص الجاهزة التي تسهل وتساعد المطور في عمله، اما الطريقة الثانية هي angular reactive forms وهي مناسبة للنماذج الديناميكية بمعنى انه يتم رسم أدوات النموذج بشكل ديناميكي داخل النموذج اثناء تشغيل هذا النموذج، كأن يكون لدينا نموذج اختبار الكتروني وكل سؤال هو عبارة عن أداة أو مجموعة أدوات على النموذج في هذه الحالة يتم رسم الأدوات بشكل ديناميكي على النموذج تبعاً لعدد الأسئلة القادمة من قاعدة البيانات، او ان يكون لدينا حقل لإدخال رقم الهاتف ونتيح للمستخدم إضافة حقول أخرى لإدخال رقم الهاتف وذلك عن طريق إضافة زر وكل ما تم الضغط على هذا الزر يتم إضافة حقل جديد لرقم الهاتف وزر آخر لحذف الحقل وهكذا، والأمثلة متعددة على النماذج الديناميكية ولكن المقصد أنه reactive forms مناسبة لهذا النوع من النماذج، كما انها مناسبة Custom validation وتعطي مرونة أكثر من angular TDF وتدعم ايضاً التحقق من الصحة الديناميكي dynamic Custom validation، والتحقق من الصحة المشروط conditional validation، اما من ناحية Unit Testing فتعتبر هي اخيار الأفضل، وأخيراً سوف نلاحظ أن اغلب الأكواد تكتب في ملف TS للcomponent الموجود فيه النموذج بعكس الطريقة الأولى التي يُكتب اغلب اكوادها في ملف HTML.

الفصل الأول

مدخل إلى

Angular Templete Driven Forms (TDF)

1.1. مقدمة:

في هذا الفصل سوف نتكلم عن التقنية الأولى التي يقدمها لنا إطار عمل angular لتعامل مع النماذج، وهي Angular Template Driven Forms وتكتب اختصاراً angular TDF، وسوف يكون الشرح ليس مجرد شرح مفهوم فقط، وإنما سوف أقوم بإعطاء مثال متكامل على نموذج وكلما تعلمنا مفهوم جديد نضيفه إلى هذا النموذج إلى أن أن تكتمل لدينا جميع المفاهيم من خلال مثال واحد فقط.

2.1. المفاهيم الأساسية لبناء النماذج في angular TDF:

سوف نقوم بشرح هذه المفاهيم عن طريق الشرح العملي لذلك نقوم بإنشاء مشروع angular جديد ونضيف له bootstrap (لمعرفة طريقة انشاء مشروع Angular جديد وبنية الخاصة به بالإضافة إلى طريقة إضافة مكتبات خارجية الرجاء مراجعة الكتاب الأول من هذه السلسلة Angular Environment Setup)، وفي ملف template (ملف app.component.html) نقوم بإضافة الكود التالي:

ملف app.component.html

```
1.    <div class="container-fluid">
2.        <div class="card mx-auto col-sm-12 col-md-6 col-lg-6 col-xl-6" >
3.
4.            <div class="card-header">
5.                <h4 class="text-center">Template Driven Forms</h4>
6.            </div>
7.
8.            <div class="card-body">
9.                <form>
10.
11.                   <div class="form-group">
12.                       <label for="">Name</label>
13.                       <input type="text" class="form-control">
14.                   </div>
15.                   <div class="form-group">
16.                       <label for="">E-Mail</label>
17.                       <input type="email" class="form-control">
18.                   </div>
19.                   <div class="form-group">
20.                       <label for="">Password</label>
21.                       <input type="password" class="form-control">
22.                   </div>
23.                   <div class="form-group">
24.                       <label for=""> Confirm Password </label>
25.                       <input type="password" class="form-control">
26.                   </div>
27.                   <div class="form-group">
28.                       <label for="">Phone</label>
29.                       <input type="tel" class="form-control">
30.                   </div>
31.                   <div class="form-group">
32.                       <select class="custom-select">
33.                           <option value=""> I am Interested in ..</option>
```

```
34.                         <option *ngFor="let topic of topics">{{topic}}</option>
35.                     </select>
36.                 </div>
37.                 <div class="mb-3">
38.                     <label>Time Preference</label>
39.                     <div class="form-check">
40.                         <input class="form-check-
input" type="radio" name="TimePreference" value="Morning">
41.                         <label class="form-check-
label">Morning 9AM - 12PM</label>
42.                     </div>
43.                     <div class="form-check">
44.                         <input class="form-check-
input" type="radio" name="TimePreference" value="Evining">
45.                         <label class="form-check-label">Evining 5PM - 8PM</label>
46.                     </div>
47.                 </div>
48.                 <div class="form-check mb-3">
49.                   <input class="form-check-input" type="checkbox">
50.                   <label class="form-check-
label"> Send Me Promotional Offers by phone</label> <br>
51.                   <input class="form-check-input" type="checkbox">
52.                   <label class="form-check-
label"> Send Me Promotional Offers by Email</label>
53.                 </div>
54.                 <div class="card-footer text-muted">
55.                   <button class="btn btn-
primary " type="submit">Submit Form</button>
56.                 </div>
57.
58.             </form>
59.         </div>
60.     </div>
61. </div>
```

وفي ملف الكلاس (app.component.ts) نقوم بإضافة المصفوفة التالية:

ملف app.component.ts

```
1.import { Component } from '@angular/core';
2.@Component({
3.    selector: 'app-root',
4.    templateUrl: './app.component.html',
5.    styleUrls: ['./app.component.css']
6.})
7.export class AppComponent {
8.   topics = ['Angular', 'React', 'Vue'];
9.}
```

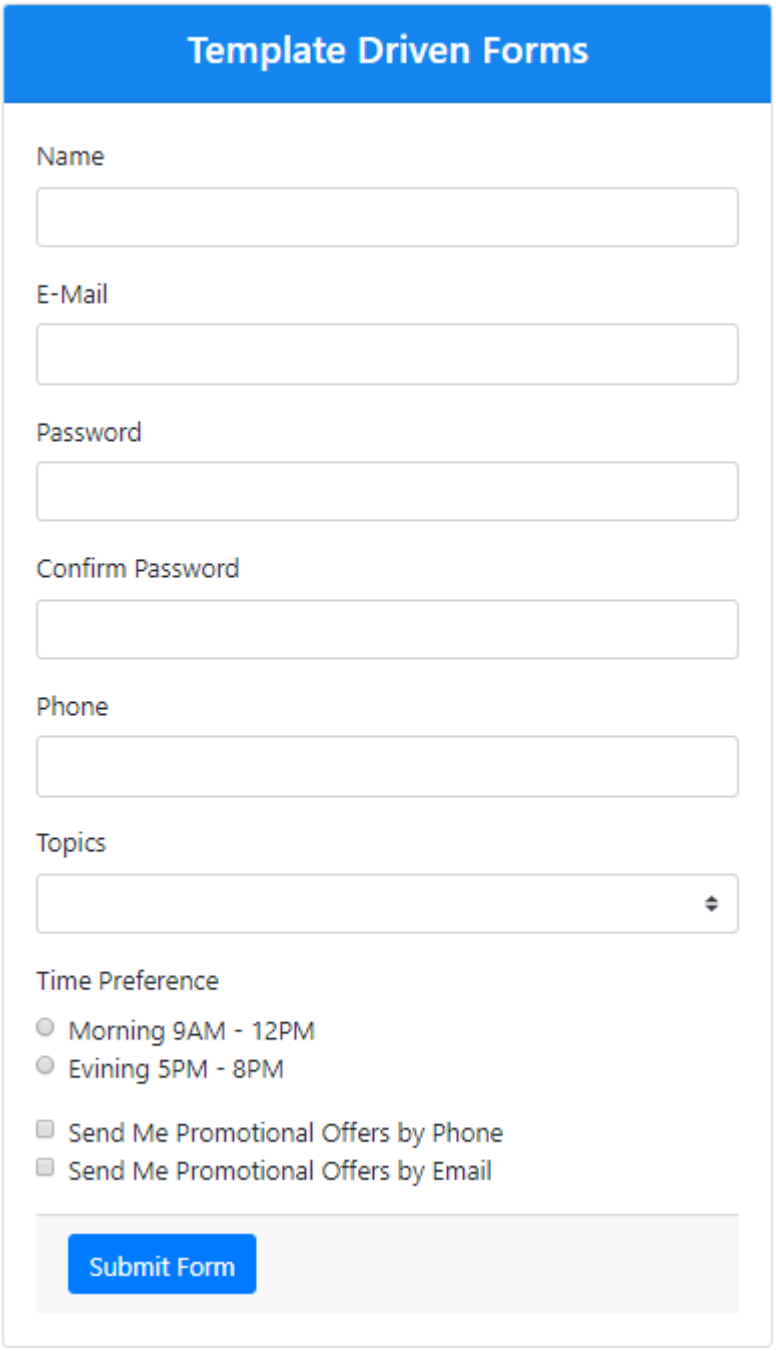
المصفوفة في السطر 8 باسم topics

وفي ملف style - (ملف app.component.css) نضيف الكود التالي:

ملف app.component.css

```
1. .card {
2.   margin-top: 50px;
3.   padding-top: 20px;
4.   padding-right: 0px;
5.   padding-left: 0px;
6. }
7.
8. .card-header {
9.   background-color: rgb(20, 133, 238);
10.  color: white;
11.}
12.
13.input {
14.  font-family: FontAwesome, "Open Sans", Verdana, sans-serif;
15.}
16.
17..alert-danger {
18.  border: .1em solid darksalmon;
19.}
20.
21..btn-primary.disabled,
22..btn-primary:disabled {
23.  background-color: #6c757d;
24.  border-color: #6c757d
25.}
26.
```

نحفظ المشروع ومن ثم نقوم بتشغيله عن طريق المتصفح (سوف استخدم متصفح chrome) وذلك بكتابة الأمر ng serve -o في terminal (يجب التأكد أنك على نفس مسار المشروع)، ولمعرفة طريقة التعامل مع Angular CLI الرجاء مراجعة الكتاب الأول من هذه السلسلة، اما الآن لنرى شكل النموذج Form بعدما كتبنا الأمر السابق في المتصفح، كالتالي:



ما سبق هو عبارة عن كود html و CSS مع بعض كلاسات bootstrap 4 والقليل جداً من كود الـ angular ولا يوجد أي كود يتعلق بـ angular forms، لذلك لكي نتعامل مع angular forms لابد من إضافة FormsModule في ملف app.module.ts كالتالي:

ملف app.module.ts

```
1. import { BrowserModule } from '@angular/platform-browser';
2. import { NgModule } from '@angular/core';
3. import { FormsModule } from '@angular/forms';
4. import { AppComponent } from './app.component';
5. @NgModule({
6.     declarations: [
7.         AppComponent
8.     ],
9.     imports: [
```

```
10.         BrowserModule,
11.         FormsModule,
12.         NgModule
13.     ],
14.     providers: [],
15.     bootstrap: [AppComponent]
16.})
17.export class AppModule { }
18.
```

في السطر 3 استدعينا FormModule وفي السطر 11 قمنا بإضافته إلى مصفوفة الأستدعاءات

يحتوي FormsModule على مجموعة من الDirective منها ngForm، حيث يقدم لنا هذا الدايكرتيف معلومات ذات قيمة عن الForm المحدد، مثلاً قيم الأدوات التي يحتويها هذا الForm وما إذا هذه القيم صحيحة أو لا، وللاستفادة من هذا الDirective نقوم اولاً بإنشاء متغير (template reference variable) – ولمعرفة أكثر عن template reference variable الرجاء راجعة الكتاب الثاني من هذه السلسلة – لهذا الForm ومن ثم نربطه بالدايكرتيف وسوف أسمي هذا المتغير #userForm، وثانياً نضيف Directive آخر أسمه ngModel لجميع الأدوات التي نريد من ngForm الوصول لها، ثالثاً نضيف خاصية الname لكل أداة تحتوي على الدايكرتيف ngModel، كالتالي:

ملف app.component.html

```
1. <div class="container-fluid">
2.   <div class="card mx-auto col-sm-12 col-md-6 col-lg-6 col-xl-6">
3.     <div class="card-header">
4.       <h4 class="text-center">Template Driven Forms</h4>
5.     </div>
6.     <div class="card-body">
7.       <form #userForm="ngForm">
8.         <div class="form-group">
9.           <label for="">Name</label>
10.          <input type="text" class="form-control" name="userName" ngModel />
11.        </div>
12.        <div class="form-group">
13.          <label for="">E-Mail</label>
14.          <input type="email" class="form-control" name="email" ngModel />
15.        </div>
16.        <div class="form-group">
17.          <label for="">Password</label>
18.          <input type="password" class="form-control" name="password" ngModel />
19.        </div>
20.        <div class="form-group">
21.          <label for=""> Confirm Password </label>
22.          <input
23.            type="password"
24.            class="form-control"
25.            name="confirmPassword"
26.            ngModel
27.          />
28.        </div>
29.        <div class="form-group">
```

```
30.           <label for="">Phone</label>
31.           <input type="tel" class="form-control" name="phone" ngModel />
32.         </div>
33.         <div class="form-group">
34.           <select class="custom-select" name="topics" ngModel>
35.             <option value="">I am Interested in ..</option>
36.             <option *ngFor="let topic of topics">{{topic}}</option>
37.           </select>
38.         </div>
39.         <div class="mb-3">
40.           <label>Time Preference</label>
41.           <div class="form-check">
42.             <input
43.               class="form-check-input"
44.               type="radio"
45.               name="TimePreference"
46.               ngModel
47.               value="Morning"
48.             />
49.             <label class="form-check-label">Morning 9AM - 12PM</label>
50.           </div>
51.           <div class="form-check">
52.             <input
53.               class="form-check-input"
54.               type="radio"
55.               name="TimePreference"
56.               ngModel
57.               value="Evining"
58.             />
59.             <label class="form-check-label">Evining 5PM - 8PM</label>
60.           </div>
61.         </div>
62.         <div class="form-check mb-3">
63.           <input
64.             class="form-check-input"
65.             type="checkbox"
66.             name="subscribePhone"
67.             ngModel
68.           />
69.           <label class="form-check-label">
70.             Send Me Promotional Offers by phone</label
71.           >
72.           <br />
73.           <input
74.             class="form-check-input"
75.             type="checkbox"
76.             name="subscribeEmail"
77.             ngModel
78.           />
79.           <label class="form-check-label">
80.             Send Me Promotional Offers by Email</label
81.           >
82.         </div>
```

```
83.         <div class="card-footer text-muted">
84.           <button class="btn btn-primary" type="submit">Submit Form</button>
85.         </div>
86.       </form>
87.     </div>
88.   </div>
89. </div>
```

راجع الأسطر (7-10-14-18-25-26-31-34-45-46-55-56-66-67-76-77)

ملاحظة: لك حرية أختيار الاسماء للخاصية name ولكن يفضل أن تكون معبرة عن الأداة الموجودة بها هذه الخاصية، كما يجب الانتباه ان الدايكريتف ngModel لا يعمل إذا لم تكن هنالك الخاصية name

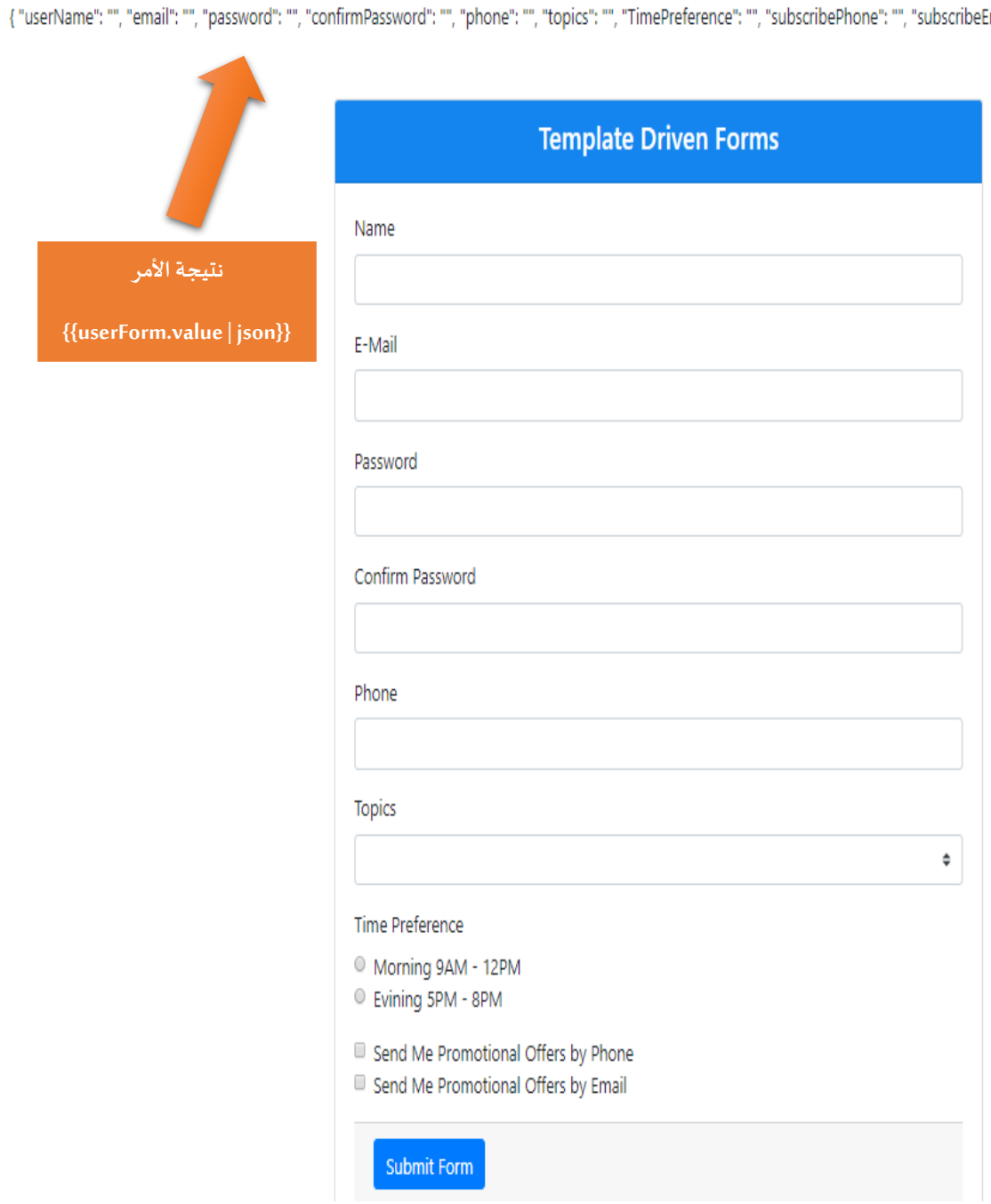
الآن المتغير userForm أصبح يمتلك جميع مميزات الدايركتيف ngForm ومن أهمها خاصية الvalue التي تجلب لنا جميع قيم الأدوات داخل الForm، ولتأكد من صحة ما قمنا به سوف اطبع قيم الأدوات عن طريق الأمر userForm.value في أي مكان داخل ملف template عن طريق ميزة Binding ذات الاسم interpolation ولمعرفة أكثر عن هذه الميزة تستطيع مراجعة الكتاب الثاني من هذه السلسلة، بعد تحويلها إلى تنسيق Json عن طريق الpipe المسمى json – يقوم بتحويلها على شكل كائن object، لمعرفة أكثر عن Pipes الرجاء مراجعة الكتاب الثالث من هذه السلسلة Angular Pipes and Directives، بحيث يصبح الأمر في شكله النهائي كالتالي:

```
{{ userForm.value |  json}}
```

مع ملاحظة انني سوف أقوم بكتابة هذه الأمر تحت اول في الملف، كالتالي:

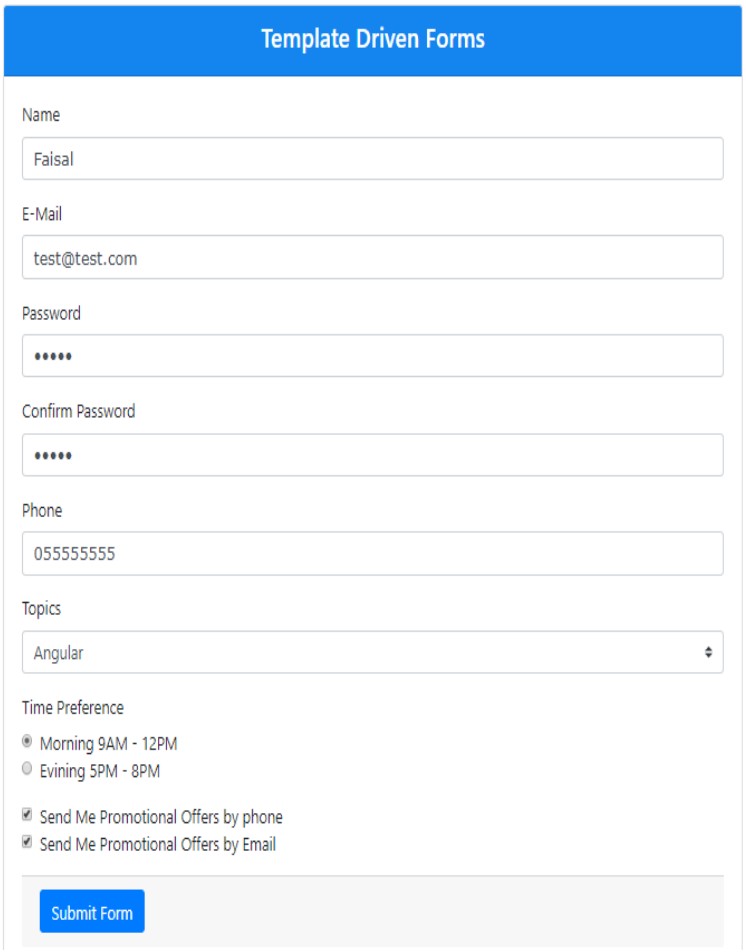
```
1.     <div class="container-fluid">
2.     {{ userForm.value |  json}}
```

الآن لنحفظ المشروع ونرى التغيرات على النموذج form داخل المتصفح:



نلاحظ أنه فوق النموذج ظهر لنا كائن من النوع json وهو نتيجة الأمر {{userForm.value | json}} وهذا الأمر يجلب لنا اسماء الأدوات التي سجلناها سابقاً للأدوات عن طريق الخاصية name لكل أداة مع value قيمة الأداة، وكما نلاحظ أن القيم داخل هذا الكائن فارغة والسبب كما هو واضح أن النموذج فارغ، الآن لنقم بإدخال بعض القيم لهذه النموذج ونرى التغيرات على الكائن:

{ "userName": "Faisal", "email": "test@test.com", "password": "12345", "confirmPassword": "12345", "phone": "055555555", "topics": "Angular", "TimePreference": "Morning", "subscribePhone": true, "subscribeEmail": true }



نلاحظ أن القيم تتحدث تلقائياً داخل الكائن بمجرد كتابتنا لقيم داخل الأدوات الخاصة بـForm وأصبحنا نستطيع الوصول لقيم جميع هذه الأدوات عن طريق المتغير userForm بعد أن ربطناه بالدايركتيف ngForm، وهذه أول ميزة يقدمها لنا الـ angular forms، فالذي قمنا به أننا اضفنا بعض الـ Directives الجاهزة وهو قام بباقي العمل واعطانا وصول لهذه القيم بكل سهولة.

كما أن الـangular forms يقدم لنا Directive آخر اسمه ngModelGroup وهذا الدايركتيف يُستفاد منه في حالة أن لدينا مجموعة من الأدوات داخل الـForm لها نفس المجال، كأن نريد من المستخدم أن يقوم بتسجيل عنوان ونضع له مجموعة أدوات، أداة لإدخال اسم المدينة وأخرى اسم الشارع وأخرى أسم الحي، أو ان يقوم بتسجيل الاسم الأول في أداة والاسم الثاني في أداة أخرى والثالث في أداة ثالثة، بمعنى آخر أن يكون لدينا نموذج فرعي داخل النموذج الرئيسي، مع العلم أنه يمكننا إضافة العدد الذي نريده من النماذج الفرعية بحسب احتياجنا، في هذه الحالة نستطيع أن نضيف متغير ونربطه بالدايركتيف ngModelGroup لكي يشير إلى قيم هذه الأدوات ذات المجال الواحد، ولتوضيح سوف اضيف مجموعة أكواد في ملف app.component.html بحيث يستطيع المستخدم من إضافة العنوان الخاص به، والعنوان يتم إدخاله من خلال ثلاث أدوات (أداة لإدخال اسم المدينة – أداة لإدخال أسم الحي – أداة أخرى لإدخال اسم الشارع)، وبما انها ذات مجال واحد سوف تجمع في tag واحد وهو div، ونضيف لهذا الdiv الدايريكتف ngModelGroup ونمرر له متغير بحيث أن هذا

المتغير يشير إلى الأدوات الثلاث السابقة ونستطيع إن نصل إلى قيمها عن طريق هذا المتغير، مع ملاحظة أنه لابد ايضاً من إضافة الدايركتيف ngModel لكل أداة وخاصية الـname كما وضحناها سابقاً، كالتالي:

```
1. <div class="container-fluid">
2.   {{userForm.value | json}}
3.   <div
4.     class="card mx-auto col-sm-12 col-md-6 col-lg-6 col-xl-6"
5.     style="padding-top: 0"
6.   >
7.     <div class="card-header">
8.       <h4 class="text-center">Template Driven Forms</h4>
9.     </div>
10.
11.    <div class="card-body">
12.      <form #userForm="ngForm">
13.        <div class="form-group">
14.          <label for="">Name</label>
15.          <input type="text" class="form-control" name="userName" ngModel />
16.        </div>
17.
18.        <div class="form-group">
19.          <label>E-Mail</label>
20.          <input class="form-control" type="email" name="email" ngModel />
21.        </div>
22.
23.        <div class="form-group">
24.          <label>Password</label>
25.          <input class="form-control" type="password" name="password" ngModel />
26.        </div>
27.
28.        <div class="form-group">
29.          <label>Confirm Password</label>
30.          <input
31.            class="form-control"
32.            type="password"
33.            name="confirmPassword"
34.            ngModel
35.          />
36.        </div>
37.
38.        <div class="form-group">
39.          <label>Phone</label>
40.          <input type="tel" class="form-control" name="phone" ngModel />
41.        </div>
42.
43.        <div ngModelGroup="address">
44.          <div class="form-group">
45.            <label>City</label>
46.            <input type="text" class="form-control" name="city" ngModel />
47.          </div>
48.          <div class="form-group">
49.            <label>District</label>
```

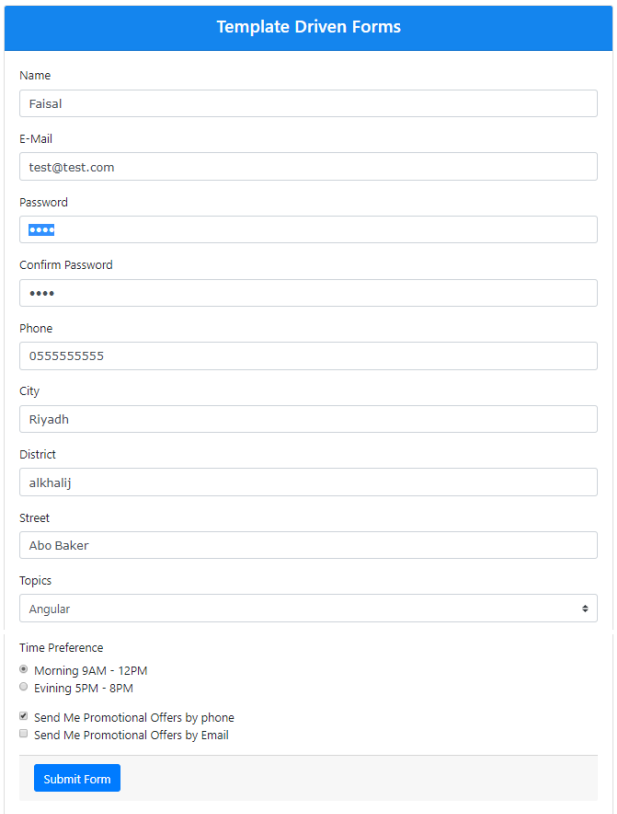
```
50.             <input type="text" class="form-control" name="district" ngModel />
51.           </div>
52.           <div class="form-group">
53.             <label>Street</label>
54.             <input type="text" class="form-control" name="street" ngModel />
55.           </div>
56.         </div>
57.
58.         <div class="form-group">
59.           <label>Topics</label>
60.           <select class="custom-select" name="topics" ngModel>
61.             <option value="">I am Interested in ..</option>
62.             <option *ngFor="let topic of topics">{{topic}}</option>
63.           </select>
64.         </div>
65.
66.         <div class="mb-3">
67.           <label>Time Preference</label>
68.           <div class="form-check">
69.             <input
70.               class="form-check-input"
71.               type="radio"
72.               name="TimePreference"
73.               ngModel
74.               value="Morning"
75.             />
76.             <label class="form-check-label">Morning 9AM - 12PM</label>
77.           </div>
78.           <div class="form-check">
79.             <input
80.               class="form-check-input"
81.               type="radio"
82.               name="TimePreference"
83.               ngModel
84.               value="Evining"
85.             />
86.             <label class="form-check-label">Evining 5PM - 8PM</label>
87.           </div>
88.         </div>
89.
90.         <div class="form-check mb-3">
91.           <input
92.             class="form-check-input"
93.             type="checkbox"
94.             name="subscribePhone"
95.             ngModel
96.           />
97.           <label class="form-check-label">
98.             Send Me Promotional Offers by phone</label
99.           >
100.             <br />
101.             <input
102.               class="form-check-input"
```

```
103.               type="checkbox"
104.               name="subscribeEmail"
105.               ngModel
106.             />
107.             <label class="form-check-label">
108.               Send Me Promotional Offers by Email</label
109.             >
110.           </div>
111.
112.           <div class="card-footer text-muted">
113.             <button class="btn btn-primary" type="submit">Submit Form</button>
114.           </div>
115.         </form>
116.       </div>
117.     </div>
118.   </div>
```

الاسطر المُضافة من الاسطر 43 إلى 56.

الآن لنرى التغيرات على النموذج form ونتيجة الأمر {{userForm.value | json}}:


```
{ "userName": "Faisal", "email": "test@test.com", "password": "1234", "confirmPassword": "1234", "phone":
"0555555555", "address": { "city": "Riyadh", "district": "alkhalij", "street": "Abo Baker" }, "topics":
"Angular", "TimePreference": "Morning", "subscribePhone": true, "subscribeEmail": "" }
```

نلاحظ ان الأدوات الثلاث أصبحت object باسم address من ضمن الobject الرئيسي، واصبح هذا المتغير address يشير لها ونستطيع الوصول إلى قيم هذه الأدوات من خلال هذا المتغير.

الآن نحذف اكواد الـHTML الخاصة بـngModelGroup فقد ذكرتها لتوضيح فقط، ونكمل شرح الـ angular forms على نموذجنا بشكله الأول.

وبذلك نكون أنهينا المفاهيم الأساسية لبناء النموذج عن طريق angular TDF، من ngForm و ngModel..الخ من المفاهيم الأساسية، واصبحنا مستعدين للأنتقال إلى النقطة الثانية وهي كيفية إنشاء كلاس ليكون نقطة وصل بين بيانات وقيم النموذج وقاعدة البيانات.

كنوع من التمرين نحتاج أن نقوم بإنشاء وصف لبيانات النموذج بحيث تكون كنوع يتم تعريف Object الذي نربطه بالngForm، لأنه في التطبيقات الواقعية اغلب هذه النماذج التي تستقبل هذه البيانات يتم ارسالها في العادة إلى قاعدة بيانات لحفظها، او العكس جلبها من قاعدة البيانات وعرضها في هذه النماذج لتعديل عليها، لذلك نحتاج إلى انشاء نوع او ما يسمى Custom Type لوصف هذه البيانات لكيلا يحدث أي خطأ بين البيانات في النموذج وما يقابلها من حقول في جداول قواعد البيانات، وبطبيعة الحال هنالك طرق عديدة لإضافة وصف لهذه البيانات منها عن طريق interface ومنها عن طريق class، وايضاً class نفسه نستطيع إضافة وصف له بأكثر من طريقة، ويجب ان أشير ان هذا الأمر ليس له علاقة بالangular forms وانما يرتبط مباشرة بمفاهيم البرمجة بإساليب OOP لذلك لن ادخل بالتفاصيل والمفاهيم الخاصة به لكيلا نشتت انتباه المتعلم عن الهدف الأساسي من هذا الكتاب.

أولاً: نُنشئ class بأسم user وذلك بكتابة الأمر التالي في terminal ونتأكد أننا بنفس مسار المشروع:

```
ng g class user
```

ثانياً: نفتح ملف user.ts الذي انشأناه في الأمر السابق، وفي داخله class باسم User نقوم بانشاء الخصائص وانواع هذه الخصائص: (لا يشترط أن تكون اسماء الخصائص نفس اسماء الخاصية name الموجودة في كل أداة):

user.ts ملف

```
1. export class User {
2.     constructor(
3.         public name: string,
4.         public email: string,
5.         public password: number,
6.         public phone: number,
7.         public topic: string,
8.         public timePreference: string,
9.         public subscribePhone: boolean,
10.        public subscribeEmail: boolean
11.    ) { }
12.}
```

ثالثاً: في ملف الكلاس (app.component.ts) نستدعي الملف user.ts ومن ثم نعمل instance (نُنشئ كائن object) من class الموجود داخله المُسمى User، وليكن اسم هذا الكائن userData:

ملف app.component.ts

```
1. import { Component, OnInit } from '@angular/core';
2. import { User } from './user';
3.
4. @Component({
5.     selector: 'app-root',
6.     templateUrl: './app.component.html',
7.     styleUrls: ['./app.component.css']
8. })
9.
10.export class AppComponent implements OnInit {
11.
12.    topics = ['Angular', 'React', 'Vue'];
13.
14.    userData = new User('', '', null, null, '', '', false, false)
15.
16.    constructor() { }
17.
18.    ngOnInit() { }
19.}
20.
```

السطر 2 استدعينا الملف user.ts والسطر 14 انشأنا كائن من class واسميناه userData

رابعاً: في ملف template نقوم بربط خصائص الكائن userData بكل أداة مكافئة له في النموذج، عن طريق خاصية Tow Way DataBinding او ما يسمى [(ngModel)] (ولمعرفة أكثر الرجاء مراجعة الكتاب الثاني من هذه السلسلة) التي تقدمها لنا angular، كالتالي:

ملف app.component.html

```
1. <div class="container-fluid">
2.   <div class="card mx-auto col-sm-12 col-md-6 col-lg-6 col-xl-6">
3.     <div class="card-header">
4.       <h4 class="text-center">Template Driven Forms</h4>
5.     </div>
6.
7.     <div class="card-body">
8.       <form #userForm="ngForm">
9.         <div class="form-group">
10.          <label for="">Name</label>
11.          <input
12.            type="text"
13.            class="form-control"
14.            name="userName"
15.            [(ngModel)]="userData.name"
16.          />
17.        </div>
18.
19.        <div class="form-group">
20.          <label>E-Mail</label>
21.          <input
22.            class="form-control"
```

```
            type="email"
            name="email"
            [(ngModel)]="userData.email"
          />
        </div>

        <div class="form-group">
          <label>Password</label>
          <input
            class="form-control"
            type="password"
            name="password"
            [(ngModel)]="userData.password"
          />
        </div>

        <div class="form-group">
          <label>Confirm Password</label>
          <input
            class="form-control"
            type="password"
            name="confirmPassword"
            ngModel
          />
        </div>

        <div class="form-group">
          <label>Phone</label>
          <input
            type="tel"
            class="form-control"
            name="phone"
            [(ngModel)]="userData.phone"
          />
        </div>

        <div class="form-group">
          <label>Topics</label>
          <select
            class="custom-select"
            name="topics"
            [(ngModel)]="userData.topic"
          >
            <option value="">I am Interested in ..</option>
            <option *ngFor="let topic of topics">{{topic}}</option>
          </select>
        </div>

        <div class="mb-3">
          <label>Time Preference</label>
          <div class="form-check">
            <input
              class="form-check-input"
```

```
76.              type="radio"
77.              name="TimePreference"
78.              [(ngModel)]="userData.timePreference"
79.              value="Morning"
80.            />
81.            <label class="form-check-label">Morning 9AM - 12PM</label>
82.          </div>
83.          <div class="form-check">
84.            <input
85.              class="form-check-input"
86.              type="radio"
87.              name="TimePreference"
88.              [(ngModel)]="userData.timePreference"
89.              value="Evining"
90.            />
91.            <label class="form-check-label">Evining 5PM - 8PM</label>
92.          </div>
93.        </div>
94.
95.        <div class="form-check mb-3">
96.          <input
97.            class="form-check-input"
98.            type="checkbox"
99.            name="subscribePhone"
100.              [(ngModel)]="userData.subscribePhone"
101.            />
102.            <label class="form-check-label">
103.              Send Me Promotional Offers by phone</label
104.            >
105.            <br />
106.            <input
107.              class="form-check-input"
108.              type="checkbox"
109.              name="subscribeEmail"
110.              [(ngModel)]="userData.subscribeEmail"
111.            />
112.            <label class="form-check-label">
113.              Send Me Promotional Offers by Email</label
114.            >
115.          </div>
116.
117.          <div class="card-footer text-muted">
118.            <button class="btn btn-primary" type="submit">Submit Form</button>
119.          </div>
120.        </form>
121.      </div>
122.    </div>
123.  </div>
124.
```

راجع الاسطر (15 – 25 – 35 – 55 – 64 – 78 – 88 – 100 – 110)

ملاحظة: تم حذف الأمر {{userForm.value | json}} من الكود لإنتفاء الحاجة له فقد وضعته لتجربة والتأكد من صحة ما قمنا به.

الآن أصبح النموذج جاهز لأرسال بياناته لحفظها في قاعدة البيانات، ولكن قبل ارسال النموذج لابد أن نعمل للبيانات validation والتأكد من صحتها وهل هي صالحة أم لا وفق شروط معينة، وهذا الذي سوف نتعلمه إن شاء الله في الفصل الثاني.

الفصل الثاني

Validations in Angular TDF

1.2. مقدمة:

في هذا الفصل سوف نتعلم بإذن الله المفاهيم والأساسيات لـ validation في angular forms TDF وسو أحاول بإذن الله تأسيس بنية قوية للقارئ يستطيع الانطلاق منها وتطوير نفسه، أما من ناحية التأكد من صحة البيانات وتقديم تغذية راجعة للمستخدم عن طريق رسائل تحدد نوع الخطأ فهي تعتبر من الأمور المهمة في أي نظام، فمثلاً لو ان المستخدم لم يقم بإدخال البريد بشكل صحيح او كلمة السر لم تكن وفق الشروط التي حددها مصمم النظام او ترك خانة فارغة ويجب عليه تعبأتها لكي يكتمل ارسال النموذج إلى السيرفر لمعالجته...الخ، هذه الأخطاء وغيرها الكثير تختلف على حسب نوع البيانات المراد إدخالها في الـ Form لابد من معالجتها والتأكد من صحة البيانات.

وهنالك نوعين من validation التي تقدمها لنا angular TDF:

الأولى built in validation :وهي عبارة عن مجموعة من الخصائص والكلاسات الجاهزة التي تقدمها لنا angular forms التي تسهل لنا عملية التحقق من الصحة من خلال الاستفادة من خصائص validation الموجودة في HTML5 مثل required و pattern...وmaxlength الخ.

الثانية custom validation :بعض الأحيان هنالك بعض الأنواع من التحقق من الصحة غير متوفرة في angular forms مثل منع المستخدم من إدخال بعض الأسماء أو التأكد من أن كلمة السر مشابهة لإعادة إدخال كلمة السر وغيرها من هذه الأنواع، في هذه الحال يجب على المطور بناء التحقق من الصحة الخاص به مع الأستفادة منbuilt in validation ، فهي مزيج من الاثنين، وبإذن الله سوف نتكلم عن كلا النوعين.

2.2. Built in Validation:

تقدم لنا angular في (TDF) مجموعة من الكلاسات وDirectives الجاهزة التي تساعد في تحديد نوع الخطأ والتأكد من صحته، فمثلاً تقدم لنا الـangular عن طريق angular forms ست كلاسات css تحدد ما إذا المستخدم لمس الأداة أم لا او قام بتغيير قيمة الأداة او قام بتركها فارغة بدون أي قيمة، كما في الجدول التالي:

نوع الحالة	التحقق والقيمة	أسم الكلاس (class)
إذا المستخدم لمس الأداة او لا	في حال لمس الأداة (True)	ng-touched
	في حال عدم لمس الأداة (False)	ng-untouched
إذا المستخدم عدل من قيمة الأداة	في حال التعديل (True)	ng-dirty
	في حال عدم التعديل (False)	ng-pristine
في حال المستخدم ترك الأداة بدون قيمة أو كانت القيمة غير صالحة وفق ما يطلبه النظام	في حال القيمة ليست فارغة أو صحيحة	ng-valid
	في حال قيمة فارغة أو غير صحيحة	ng-invalid

وهذه الكلاسات تأتي بشكل افتراضي بمجرد أضافتنا angular forms للمشروع الخاص بنا وكما أشرت قبل قليل أن هذه الكلاسات في الحقيقة هي كلاسات css، ولتأكد من هذا الأمر سوف نضيف متغير لأحد الأدوات الموجودة في الـForm، ولتكن

الأداة الخاصة بإدخال الاسم، ويكون اسم المتغير #name مع كتابة الأمر التالي {{name.className | json}} في ملف template، لطباعة جميع الكلاسات في هذه الأداة، كالتالي:

ملف app.component.html

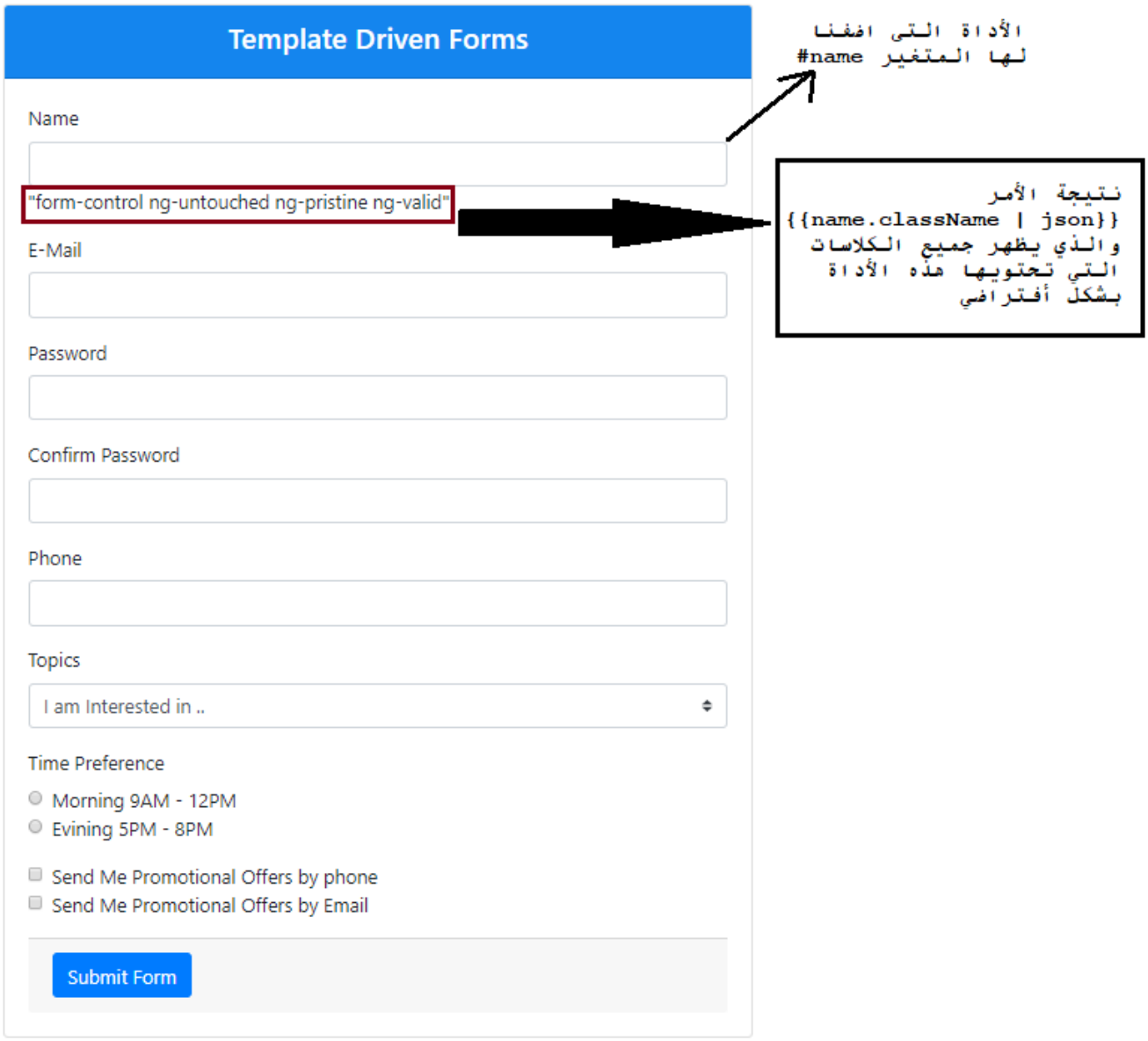
```
1. <div class="container-fluid">
2.   <div
3.     class="card mx-auto col-sm-12 col-md-6 col-lg-6 col-xl-6"
4.     style="padding-top: 0"
5.   >
6.     <div class="card-header">
7.       <h4 class="text-center">Template Driven Forms</h4>
8.     </div>
9.
10.    <div class="card-body">
11.      <form #userForm="ngForm">
12.        <div class="form-group">
13.          <label for="">Name</label>
14.          <input
15.            type="text"
16.            #name
17.            class="form-control"
18.            name="userName"
19.            [(ngModel)]="userData.name"
20.          />
21.          {{name.className | json }}
22.        </div>
23.
24.        <div class="form-group">
25.          <label>E-Mail</label>
26.          <input
27.            class="form-control"
28.            type="email"
29.            name="email"
30.            [(ngModel)]="userData.email"
31.          />
32.        </div>
33.
34.        <div class="form-group">
35.          <label>Password</label>
36.          <input
37.            class="form-control"
38.            type="password"
39.            name="password"
40.            [(ngModel)]="userData.password"
41.          />
42.        </div>
43.
44.        <div class="form-group">
45.          <label>Confirm Password</label>
46.          <input
47.            class="form-control"
48.            type="password"
```

```
            name="confirmPassword"
            ngModel
          />
        </div>

        <div class="form-group">
          <label>Phone</label>
          <input
            type="tel"
            class="form-control"
            name="phone"
            [(ngModel)]="userData.phone"
          />
        </div>

        <div class="form-group">
          <label>Topics</label>
          <select
            class="custom-select"
            name="topics"
            [(ngModel)]="userData.topic"
          >
            <option value="">I am Interested in ..</option>
            <option *ngFor="let topic of topics">{{topic}}</option>
          </select>
        </div>

        <div class="mb-3">
          <label>Time Preference</label>
          <div class="form-check">
            <input
              class="form-check-input"
              type="radio"
              name="TimePreference"
              [(ngModel)]="userData.timePreference"
              value="Morning"
            />
            <label class="form-check-label">Morning 9AM - 12PM</label>
          </div>
          <div class="form-check">
            <input
              class="form-check-input"
              type="radio"
              name="TimePreference"
              [(ngModel)]="userData.timePreference"
              value="Evining"
            />
            <label class="form-check-label">Evining 5PM - 8PM</label>
          </div>
        </div>

          <div class="form-check mb-3">
            <input
```

```
102.               class="form-check-input"
103.               type="checkbox"
104.               name="subscribePhone"
105.               [(ngModel)]="userData.subscribePhone"
106.             />
107.             <label class="form-check-label">
108.               Send Me Promotional Offers by phone</label
109.             >
110.             <br />
111.             <input
112.               class="form-check-input"
113.               type="checkbox"
114.               name="subscribeEmail"
115.               [(ngModel)]="userData.subscribeEmail"
116.             />
117.             <label class="form-check-label">
118.               Send Me Promotional Offers by Email</label
119.             >
120.           </div>
121.
122.           <div class="card-footer text-muted">
123.             <button class="btn btn-primary" type="submit">Submit Form</button>
124.           </div>
125.         </form>
126.       </div>
127.     </div>
128.   </div>
129.
```

راجع السطر 16 والسطر 21

والآن لنرى ما قمنا به من تعديلات على الForm الخاص بنا على المتصفح:



نلاحظ أن نتيجة name.className اظهر لنا أن هذه الأداة تحتوي على أربع كلاسات هي:

form-control: وهو عبارة عن كلاس من الbootstrap.

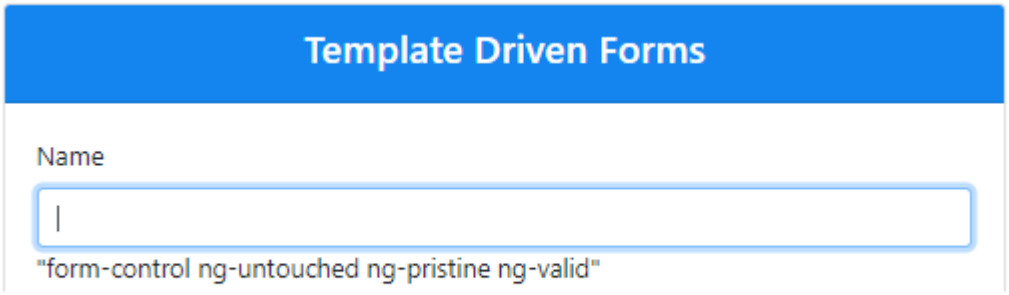
ng-untouched: كلاس تمت اضافته في الأداة عن طريق الangular forms في حال أن المستخدم لم يلمس الأداة.

ng-pristine: كلاس تمت اضافته في الأداة عن طريق الangular forms في حال أن المستخدم لم يُغير قيمة الأداة.

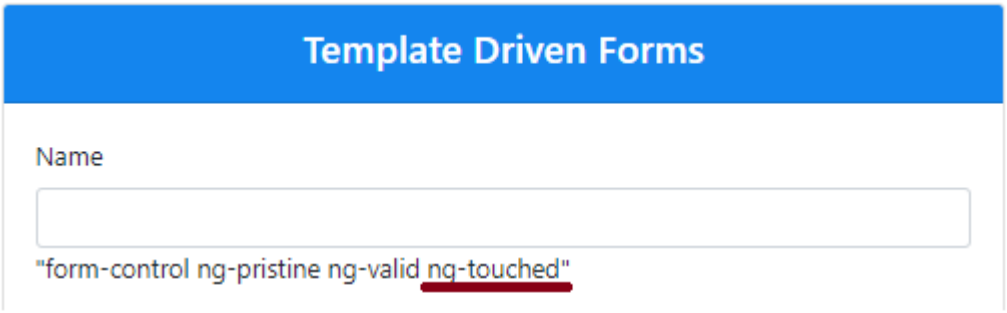
ng-valid: كلاس تمت اضافته في الأداة عن طريق الangular forms في حال أن المستخدم وضع قيمة للأداة او لم يتركها فارغة (لم نضع شروط التحقق من الصحة إلى الآن لذلك أعتبرها valid).

ومن المعلوم أن كلاسات bootstrap هي كلاسات css وهذا يدل ايضاً أن هذه الكلاسات أنها كلاسات css لأننا جمعناها جميعاً في أمر واحد والأمر الذي أظهر لنا كلاس bootstrap هو نفسه الذي اظهر لنا هذه الكلاسات.

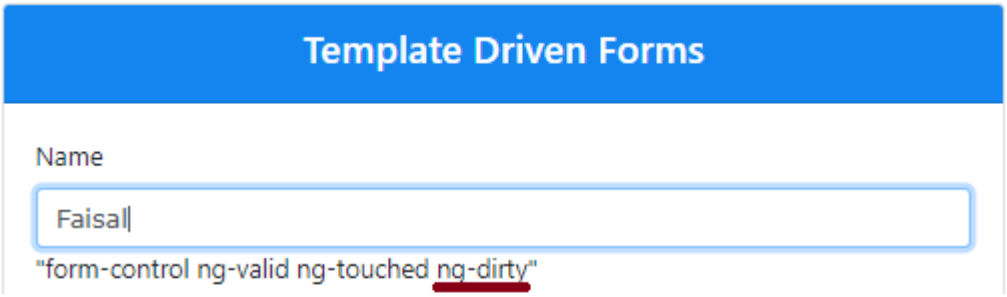
الآن سوف نقوم بالمس الأداة عن طريق وضع المؤشر عليها ومن ثم نقوم بالضغط في أي منطقة خارج الأداة، ونرى ماذا سوف يحصل:



نلاحظ أنه في حال لمس الأداة عن طريق الضغط عليها ووضع مؤشر الفأرة لم يحدث شيء



لكن في حال الضغط في أي مكان خارج الأداة نلاحظ أن الangular
حذف الكلاس ng-untouched وأضاف الكلاس ng-touched.
كأنه يُخبرنا أنه تم لمس هذه الأداة ولكن لم يتم التعديل على قيمتها



عند تغييرنا للقيمة الموجودة في الأداة نلاحظ انه تم حذف الكلاس ng-pristine وتمت إضافة الكلاس ng-dirty، كأنه يخبرنا أن قيمة الأداة قد تغيرت في حال أردت اظهار رسالة للمستخدم أو القيام بأي أمر آخر تريده.

اما الكلاس ng-valid و ng-invalid لكي نلاحظ التغيرات بينهما لابد من إضافة خاصية required التحقق من الصحة المبنية ضمنياً من ضمن HTML5 (كما قلنا سابقاً أن angular forms TDF يستفيد من خصائص التحقق من الصحة الموجودة في HTML5 ويسهل لنا الوصول لهذه الخصائص للأستفادة منها كشروط لأظهار رسائل الخطأ للمستخدمين، كما سوف اشرحه لاحقاً بإذن الله)، وخاصية required تُخبر المتصفح أن هذه الأداة لابد أن تحتوي على قيمة ولايمكن أن تُترك فارغة.

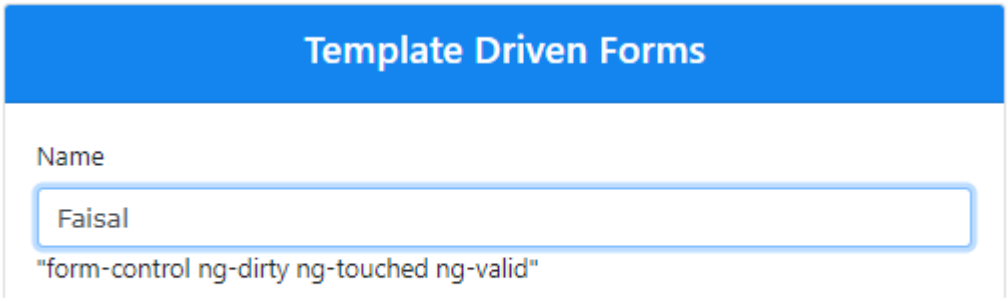
لذلك لنذهب إلى ملف template ونضيف هذه الخاصية للأداة الخاصة بإدخال الأسم والتي إضفنا لها متغير باسم #name، كالتالي:

app.component.html جزء من ملف

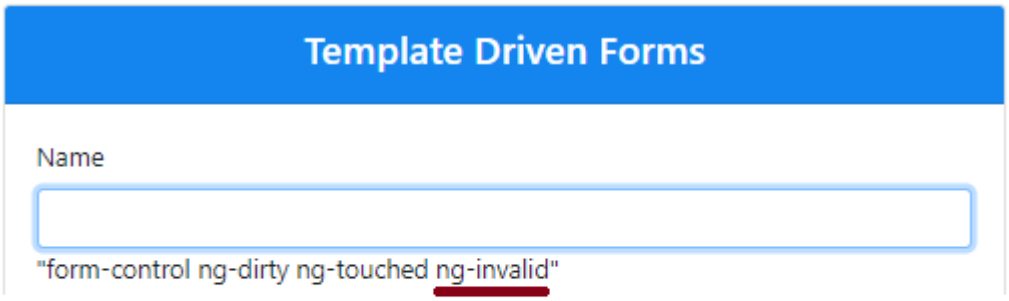
```
1. <div class="form-group">
2.   <label for="">Name</label>
3.   <input
4.     type="text"
5.     #name
6.     class="form-control"
7.     required
8.     name="userName"
9.     [(ngModel)]="userData.name"
10.  />
11.  {{name.className | json }}
12.</div>
13.
```

راجع السطر 7

الآن لنرى التغيرات على النموذج:



نلاحظ أن الكلاس ng-valid موجود في حال أن هنالك قيمة في الأداة



وعند مسح القيمة من الأداة يتغير الكلاس من **ng-valid** إلى **ng-invalid**
وهذا الكلاس يعني أن قيمة هذه الأداة أصبحت غير صالحة لأنها لابد أن تحتوي على قيمة
بسبب وجود الخاصية **required** التي ذكرناها سابقاً

وبذلك نستطيع التعامل مع هذه الكلاسات لتحقق من صحة البيانات المدخلة، وفي حال أن لم تستسغ هذه الطريقة فلا تقلق angular forms قدمت لنا طريقة ابسط وأسهل فبدلاً من التعامل مع الكلاس نستطيع التعامل مع خصائص وهذه الخصائص مسمياتها مشابهه لمسميات هذه الكلاس وتقوم بنفس المهمة ولكنها تُرجع قيمة منطقية True أو False، كما في الجدول التالي:

الكلاس	الخاصية	القيمة
ng-touched	touched	True في حال اللمس False في حال عدم اللمس
ng-untouched	untouched	True في حال عدم اللمس False في حال اللمس
ng-pristine	pristine	True في حال عدم التعديل False في حال التعديل
ng-dirty	dirty	True في حال التعديل False في حال عدم التعديل
ng-valid	valid	True في حال وجود قيمة False في حال عدم وجود قيمة
ng-invalid	invalid	True في حال عدم وجود قيمة False في حال وجود قيمة

ونستطيع الأستفادة من هذه الخصائص كشروط كأن نقول إذا كانت قيمة invalid تساوي true اظهر رسالة الخطأ أو إذا كانت قيمة untouched تساوي false أخفي رسالة الخطأ وهكذا، وهذه الخصائص موجودة في الدايركتيف ngModel وللوصول إلى هذه الخصائص عن طريق هذا الدايركتيف نقوم بإسناده إلى المتغير الخاص بالإداة، فمثلاً الأداة الخاصة بإدخال الأسم، المتغير الخاص بها اسميناه #name نقوم بإسناد الدايركتيف ngModel إلى هذا المتغير بحيث يصبح الأمر كالتالي:

```
#name="ngModel"
```

ملاحظة: سوف نعتمد الخصائص لتحقق من صحة البيانات بدلاً من الكلاسات وذلك لمرونتها وسهولة الأستخدام، وقد ذكرتها لكي يكون المتعلم ملم بجميع جوانب angular forms TDF.

الآن لنقوم بمسح الأمر:

```
{{name.className | json}}
```

وأستبداله بالأوامر التالية:

```
name.touched= {{name.touched | json}} - name.valid= {{name.valid | json}} - name.dirty=
{{name.dirty | json}}
```

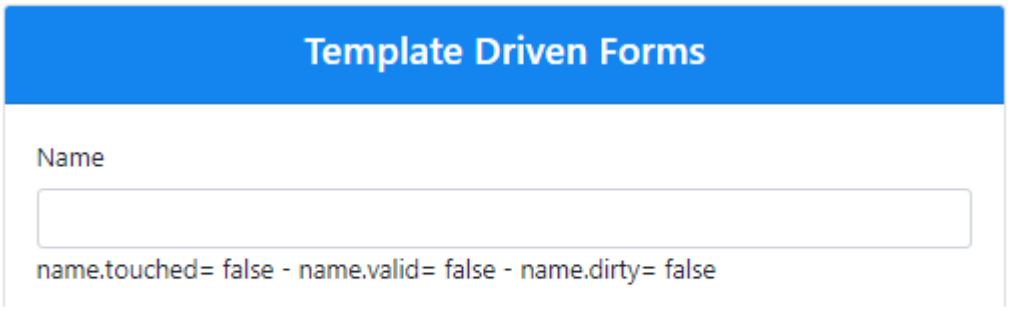
وهذه الأوامر مؤقته فقط لنرى نتيجة ما نقوم به، وايضا لا ننسى نضيف الدايكرتيف إلى المتغير name، وجميع هذه الأوامر تُكتب في ملف template لتصبح بالشكل التالي:

ملف app.component.html

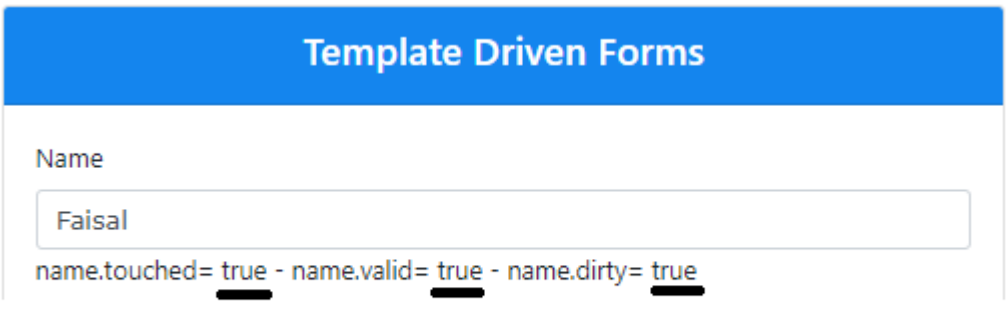
```
1. <div class="form-group">
2.   <label for="">Name</label>
3.   <input
4.     type="text"
5.     #name="ngModel"
6.     class="form-control"
7.     required
8.     name="userName"
9.     [(ngModel)]="userData.name"
10.  />
11.  name.touched= {{name.touched | json}} - name.valid= {{name.valid | json}} -
12.  name.dirty= {{name.dirty | json}}
13.</div>
14.
```

راجع الاسطر (5 – 11 – 12)

الآن لنرى ما قمنا به من تعديلات على النموذج:



نلاحظ أن جميع الخواص قيمتها false بمعنى أنه لم يتم لمس الأداة وقيمتها فارغة وبنفس الوقت لم يتم التعديل عليها



أما عند تعديلنا لقيمة الأداة تغيرة قيمة هذه الخصائص واصبحت قيمتها true

الآن لنطبق ما تعلمناه على هذه الأداة على بقية الأدوات، حيث سوف نعطي لكل أداة أسم متغير مسبوقاً بالعلامة # ونسند له الدايركتيف ngModel (مع مراعاة أن يكون اختيار اسم المتغير ذو معنى ويدل على الأداة التي يُشير إليها) وايضاً نضيف الخاصية required، كالتالي:

ملف app.component.html

```
<div class="container-fluid">
  <div
    class="card mx-auto col-sm-12 col-md-6 col-lg-6 col-xl-6"
    style="padding-top: 0"
  >
    <div class="card-header">
      <h4 class="text-center">Template Driven Forms</h4>
    </div>

    <div class="card-body">
      <form #userForm="ngForm">
        <div class="form-group">
          <label for="">Name</label>
          <input
            class="form-control"
            type="text"
            #name="ngModel"
            required
            name="userName"
            [(ngModel)]="userData.name"
          />
          name.touched= {{name.touched | json}} - name.valid= {{name.valid |
          json}} - name.dirty= {{name.dirty | json}}
        </div>

        <div class="form-group">
          <label>E-Mail</label>
          <input
            class="form-control"
            type="email"
            #email="ngModel"
            required
            name="email"
            [(ngModel)]="userData.email"
          />
        </div>

        <div class="form-group">
          <label>Password</label>
          <input
            class="form-control"
            type="password"
            #password="ngModel"
            required
            name="password"
            [(ngModel)]="userData.password"
```

```
47.          />
48.        </div>
49.
50.        <div class="form-group">
51.          <label>Confirm Password</label>
52.          <input
53.            class="form-control"
54.            type="password"
55.            #confirmPassword="ngModel"
56.            required
57.            name="confirmPassword"
58.            ngModel
59.          />
60.        </div>
61.
62.        <div class="form-group">
63.          <label>Phone</label>
64.          <input
65.            class="form-control"
66.            type="tel"
67.            #phone="ngModel"
68.            required
69.            name="phone"
70.            [(ngModel)]="userData.phone"
71.          />
72.        </div>
73.
74.        <div class="form-group">
75.          <label>Topics</label>
76.          <select
77.            class="custom-select"
78.            #topic="ngModel"
79.            name="topics"
80.            [(ngModel)]="userData.topic"
81.          >
82.            <option value="">I am Interested in ..</option>
83.            <option *ngFor="let topic of topics">{{topic}}</option>
84.          </select>
85.        </div>
86.
87.        <div class="mb-3">
88.          <label>Time Preference</label>
89.          <div class="form-check">
90.            <input
91.              class="form-check-input"
92.              type="radio"
93.              #timePreference="ngModel"
94.              required
95.              name="TimePreference"
96.              [(ngModel)]="userData.timePreference"
97.              value="Morning"
98.            />
99.            <label class="form-check-label">Morning 9AM - 12PM</label>
```

```
100.            </div>
101.            <div class="form-check">
102.              <input
103.                class="form-check-input"
104.                type="radio"
105.                #timePreference="ngModel"
106.                required
107.                name="TimePreference"
108.                [(ngModel)]="userData.timePreference"
109.                value="Evining"
110.              />
111.              <label class="form-check-label">Evining 5PM - 8PM</label>
112.            </div>
113.          </div>
114.
115.          <div class="form-check mb-3">
116.            <input
117.              class="form-check-input"
118.              type="checkbox"
119.              #subscribePhone="ngModel"
120.              name="subscribePhone"
121.              [(ngModel)]="userData.subscribePhone"
122.            />
123.            <label class="form-check-label">
124.              Send Me Promotional Offers by phone</label
125.            >
126.            <br />
127.            <input
128.              class="form-check-input"
129.              type="checkbox"
130.              #subscribeEmail="ngModel"
131.              name="subscribeEmail"
132.              [(ngModel)]="userData.subscribeEmail"
133.            />
134.            <label class="form-check-label">
135.              Send Me Promotional Offers by Email</label
136.            >
137.          </div>
138.
139.          <div class="card-footer text-muted">
140.            <button class="btn btn-primary" type="submit">Submit Form</button>
141.          </div>
142.        </form>
143.      </div>
144.    </div>
145.  </div>
146.
```

راجع الاسطر (17 – 18 – 31 – 32 – 43 – 44 – 55 – 56 – 67 – 68 – 78 – 93 – 94 – 105 – 106 – 119 – 120 – 130 – 131)

الآن اصبحت الأدوات لا تقبل قيمة فارغة، ولكن هنالك أنواع أخرى من التحقق من الصحة مثلا نريد من أداة رقم الهاتف أن تقبل كحد أقصى 10 خانات وقيم رقمية فقط وأداة الأسم ان تقبل كحد أدنى 3 خانات وأداة الرقم السري أن تقبل كحد أدنى 8 خانات مع مزيج من الحروف الكبيرة والصغيرة والأرقام، في هذه الحالة سوف نعتمد على الخصائص التحقق من الصحة validation الموجودة في HTML5، كالتالي:

خاصية minlength: تحدد أقل قيمة خانات مقبولة للأداة المستهدفة وصيغتها minlength="n" حيث n تمثل عدد الخانات.

خاصية maxlength: تحدد أعلى قيمة خانات مقبولة للأداة وصيغتها maxlength="n" حيث n تمثل عدد الخانات.

خاصية pattern: حيث تقوم هذه الخاصية بالتحقق من توافق قيمة أداة معينة مع تعبير قياسي regular expression يحدده المبرمج وتكون الصيغة العامة لهذه الخاصية كالتالي: pattern = "regular expression" حيث regular expression عبارة عن التعبير القياسي، ويتم وضعها داخل التاغ – Tag – الخاص بالأداة المستهدفة.

وسوف أورد بعض الأمثلة لبعض التعبيرات القياسية والمعنى الخاص بها: (المصدر - w3schools.com http://html5pattern.com)، كالتالي:

Input/Type	Regular Expression	Explanation
Input/password	.{6,}	يجعل الأداة تقبل على الأقل ٦ خانات
Input/password	(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,}	يجعل أداة الدخال تقبل على الأقل ٨ خانات ما بين حروف كبيرة وصغيرة وأرقام فقط ولا تقبل رموز خاصة
Input/password	(?=^.{8,}$)((?=.*\d)\|(?=.*\W+))(?![.\n])(?=.*[A-Z])(?=.*[a-z]).*$	يجعل الأداة تقبل على الأقل ٨ خانات ما بين حروف كبيرة وصغيرة وأرقام ورموز خاصة
Input/text	([1-9])+(?:-?\d){4,}	يجعل أداة الادخال تقبل ارقام فقط بحد أدنى ٤ ارقام
Input/text	^[a-zA-Z][a-zA-Z0-9-_\.]{1,20}$	يجعل الأداة تقبل فقط حروف وأرقام بحد أدنى خانتين وحد أعلى ٢٠ خانة
Input/email	[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,}$	يجعل الأداة تقبل فقط صيغة الأيميل
Input/search	[^'\x22]+	يجعل أداة الدخال لا تقبل علامة التنصيص (") او علامة التنصيص المفردة (')

والتعابير القياسية كثيرة وليس هنا المقام لحصرها.

كما أن خصائص التحقق من الصحة validation الخاصة بـ HTML5 ايضاً متعددة وكثيرة وقد أوردت ما نحتاج إلية فقط، ويمكن الرجوع إلى مصادر موثوقة كثيرة على الأنترنت في حال الرغبة في الأستزادة أكثر.

الآن لنضيف خصائص validation لكل أداة كما هو موضح في الجدول التالي:

الأداة المستهدفة	خاصية HTML5 Validation
أداة الدخال الخاصة بالاسم	minlength="3"
أداة الإدخال الخاصة بالبريد الإلكتروني	pattern="[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,}$"
أداة الدخال الخاصة بالرقم السري	pattern="(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{6,}"
أداة الإدخال الخاصة برقم الهاتف	pattern="^\d{10}$" maxlength="10"

ملف template: (ملف HTML):

ملف app.component.html

```
1. <div class="container-fluid">
2.   <div
3.     class="card mx-auto col-sm-12 col-md-6 col-lg-6 col-xl-6"
4.     style="padding-top: 0"
5.   >
6.     <div class="card-header">
7.       <h4 class="text-center">Template Driven Forms</h4>
8.     </div>
9.
10.    <div class="card-body">
11.      <form #userForm="ngForm">
12.        <div class="form-group">
13.          <label for="">Name</label>
14.          <input
15.            class="form-control"
16.            type="text"
17.            #name="ngModel"
18.            required
19.            minlength="3"
20.            name="userName"
21.            [(ngModel)]="userData.name"
22.          />
23.        </div>
24.
25.        <div class="form-group">
26.          <label>E-Mail</label>
27.          <input
28.            class="form-control"
29.            type="email"
30.            #email="ngModel"
31.            required
32.            pattern="[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,}$"
33.            name="email"
34.            [(ngModel)]="userData.email"
35.          />
36.        </div>
```

```
37.
38.         <div class="form-group">
39.           <label>Password</label>
40.           <input
41.             class="form-control"
42.             type="password"
43.             #password="ngModel"
44.             required
45.             pattern="(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{6,}"
46.             name="password"
47.             [(ngModel)]="userData.password"
48.           />
49.         </div>
50.
51.         <div class="form-group">
52.           <label>Confirm Password</label>
53.           <input
54.             class="form-control"
55.             type="password"
56.             #confirmPassword="ngModel"
57.             required
58.             name="confirmPassword"
59.             ngModel
60.           />
61.         </div>
62.
63.         <div class="form-group">
64.           <label>Phone</label>
65.           <input
66.             class="form-control"
67.             type="tel"
68.             #phone="ngModel"
69.             required
70.             pattern="^\d{10}$"
71.             maxlength="10"
72.             name="phone"
73.             [(ngModel)]="userData.phone"
74.           />
75.         </div>
76.
77.         <div class="form-group">
78.           <label>Topics</label>
79.           <select
80.             class="custom-select"
81.             #topic="ngModel"
82.             name="topics"
83.             [(ngModel)]="userData.topic"
84.           >
85.             <option value="">I am Interested in ..</option>
86.             <option *ngFor="let topic of topics">{{topic}}</option>
87.           </select>
88.         </div>
89.
```

```
90.         <div class="mb-3">
91.           <label>Time Preference</label>
92.           <div class="form-check">
93.             <input
94.               class="form-check-input"
95.               type="radio"
96.               #timePreference="ngModel"
97.               required
98.               name="TimePreference"
99.               [(ngModel)]="userData.timePreference"
100.                 value="Morning"
101.               />
102.               <label class="form-check-label">Morning 9AM - 12PM</label>
103.             </div>
104.             <div class="form-check">
105.               <input
106.                 class="form-check-input"
107.                 type="radio"
108.                 #timePreference="ngModel"
109.                 required
110.                 name="TimePreference"
111.                 [(ngModel)]="userData.timePreference"
112.                 value="Evining"
113.               />
114.               <label class="form-check-label">Evining 5PM - 8PM</label>
115.             </div>
116.           </div>
117.
118.           <div class="form-check mb-3">
119.             <input
120.               class="form-check-input"
121.               type="checkbox"
122.               #subscribePhone="ngModel"
123.               name="subscribePhone"
124.               [(ngModel)]="userData.subscribePhone"
125.             />
126.             <label class="form-check-label">
127.               Send Me Promotional Offers by phone</label
128.             >
129.             <br />
130.             <input
131.               class="form-check-input"
132.               type="checkbox"
133.               #subscribeEmail="ngModel"
134.               name="subscribeEmail"
135.               [(ngModel)]="userData.subscribeEmail"
136.             />
137.             <label class="form-check-label">
138.               Send Me Promotional Offers by Email</label
139.             >
140.           </div>
141.
142.           <div class="card-footer text-muted">
```

```
143.             <button class="btn btn-primary" type="submit">Submit Form</button>
144.           </div>
145.         </form>
146.       </div>
147.     </div>
148.   </div>
149.
```

راجع الاسطر (19 – 32 – 45 – 70 – 71)

الآن نستطيع أن نقول أننا قمنا بتغطية أهم المفاهيم والأساسيات في validation سواء الخصائص الموجودة في ngModel مثل dirty او valid..الخ أو الموجودة ضمن HTML5 مثل required او pattern...الخ وسوف نستخدم هذه الخصائص في التحكم بإظهار أو اخفاء رسائل الخطأ للمستخدم، وهذا ما سوف نتعلمه إن شاء الله في الجزء التالي.

3.2. إظهار رسائل الخطأ والتغذية الراجعة:

في هذا الجزء بإذن الله سوف نطبق المفاهيم والأساسيات التي شرحناها في الدرس السابق، بالتحكم في إظهار رسائل الخطأ للمستخدم بحسب نوع الخطأ وذلك بالأعتماد على الخصائص سواء الموجودة في ngModel او HTML5، فمثلاً إذا كانت خاصية required قيمتها true نظهر رسالة (قيمة الحقل لا يمكن أن تكون فارغة) أو إذا كان minlength قيمته true نظهر رسالة (حقل الأسم لايقبل أقل من ثلاث خانات) وهكذا.

وللوصول إلى هذه الخصائص هنالك طريقتين:

الأولى: إذا كانت الخاصية من الخصائص الموجودة في ngModel مثل (touched-valid-invalid-...الخ) تكون الصيغة العامة لها variable.property حيث أن variable هو اسم المتغير الخاص بالأداة (او بصيغة أخرى هو Template Reference Variable للعنصر، والتي قمت بشرحها بالتفصيل في الكتاب الثاني من هذه السلسلة) التي انشأناه في الدرس السابق اما property هي الخاصية الموجودة في ngModel، فلو اردنا الوصول للخاصية invalid الموجودة في الأداة الخاصة بإدخال الأسم ذات المتغير #name، نقوم بكتابة الأمر التالي name.invalid وهذا الأمر يُرجع لنا قيمة منطقية true او false، ونستطيع الأستفادة منه بإظهار أو اخفاء الرسائل.

الثانية: إذا كانت الخاصية من الخصائص التحقق من الصحة الموجودة في HTML5 مثل (required-maxlength-pattern-...الخ) تكون الصيغة العامة variable.errors?.property حيث أن variable هو اسم المتغير الخاص بالأداة التي انشأناه في الدرس السابق اما property هي الخاصية الموجودة في HTML5، فلو اردنا الوصول للخاصية required الموجودة في الأداة الخاصة بإدخال الأسم ذات المتغير #name، نقوم بكتابة الأمر التالي name.errors?.required وهذا الأمر يُرجع لنا قيمة منطقية true او false، ونستطيع الأستفادة منه بإظهار أو اخفاء الرسائل، أما errors فهي ثابتة وعلامة الاستفهام اختيارية وتستخدم لتفادي الأخطاء الغير متوقع مثل أن تكون الخاصية قيمتها undefined او null عند بداية تشغيل النموذج.

ملاحظة: يمكن أستخدام الصيغة التالية variable.hasError('property') بدلاً من الصيغة variable.errors?.property.

الآن نريد تقديم أول تغذية راجعة للمستخدم وهي عبارة عن مربع أحمر يظهر على حدود إطار الأداة المستهدفة، وللقيام بهذا الأمر نستخدم كلاس جاهز من bootstrap يقدم لنا هذا الأمر واسم هذا الكلاس is-invalid وعن طريقة ميزة class binding التي تقدمها لنا angular (لفهم أعمق عن class binding الرجاء مراجعة الكتاب الثاني من هذه السلسلة) نضيف هذا الكلاس او نخفية وفق مجموعة شروط وهذه الشروط هي ان تكون الخاصية touched قيمتها true (و) الخاصية invalid قيمتها true (أو) minlength قيمتها true، كالتالي:

```
[class.is-invalid]="name.touched && (name.invalid || name.errors?.minlength)"
```

ولو نلاحظ أن الخاصيتين valid و invalid هم بالأساس مرتبطتين بخواص التحقق من الصحة الموجودة في HTML5 بحيث لو كانت الخاصية minlength قيمتها true ايضاً تصبح قيمة invalid تساوي true، من هذا المنطلق نستطيع اختصار الكود السابق كالتالي:

```
[class.is-invalid]="name.touched && name.invalid"
```

بمعنى أنه إذا كانت الخاصية touched تساوي true والخاصية invalid تساوي true أضف الكلاس is-invalid للأداة وفي حال كانت قيمهما جميعاً false أزل هذا الكلاس من الأداة، الآن لنضيف هذا السطر البرمجي للأداة المستهدفة، كالتالي:

جزء من ملف app.component.html

```
1. <div class="form-group">
2.   <label for="">Name</label>
3.   <input
4.     class="form-control"
5.     type="text"
6.     #name="ngModel"
7.     required
8.     minlength="3"
9.     name="userName"
10.    [class.is-invalid]="name.touched && name.invalid"
11.    [(ngModel)]="userData.name"
12.  />
13.</div>
14.
```

راجع السطر 10

لو نلاحظ الكود السابق وخصوصاً السطر 4 أن هنالك كلاس آخر من bootstrap اسمه form-control (هذا الكلاس لاعلاقة له في هذه الدورة) ولكن بما اننا استخدمنا class binding قبل قليل لأضافة أو اخفاء الكلاس is-invalid نستطيع دمج السطر 3 والسطر 10 بسطر واحد وذلك عن طريق استخدام نوع آخر من class binding اسمه [ngClass] (هذه النوع يسمح لنا بالتعامل مع أكثر من كلاس وليس كلاس واحد فقط من خلال أمر واحد، ولفهم أكثر الرجاء مراجعة الكتاب الثاني من هذه السلسلة Angular Pipes and Directives)، بحيث يصبح الأمر، كالتالي:

```
[ngClass]= "{'form-control': true, 'is-invalid': name.touched && name.invalid}"
```

ومعنى السطر السابق أجعل الكلاس form-control مضاف دائماً إلى الأداة وذلك بجعل قيمته true أما الكلاس is-invalid فقم بإضافته وحذفه وفق الشروط التي ذكرناها سابقاً.

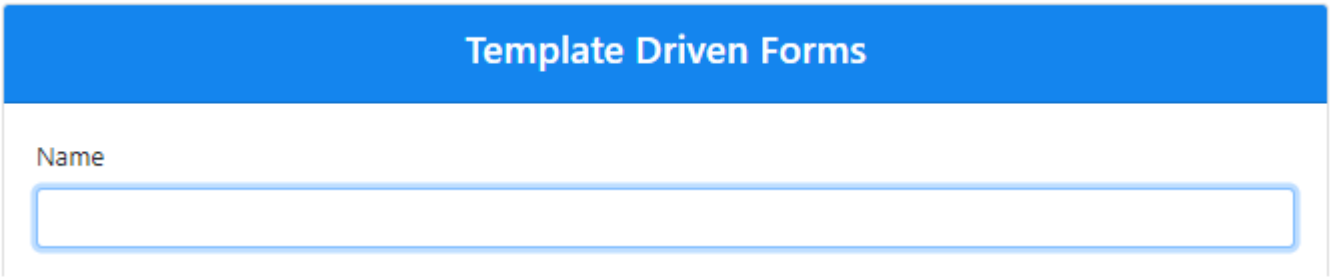
الآن لنقم بإضافة السطر السابق إلى كود الأداة المستهدفة، كالتالي:

جزء من ملف app.component.html

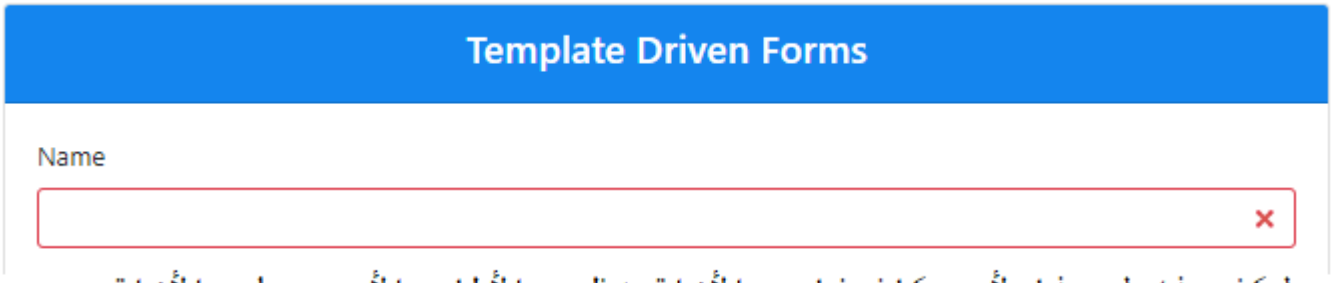
```
1. <div class="form-group">
2.   <label for="">Name</label>
3.   <input
4.     type="text"
5.     #name="ngModel"
6.     required
7.     minlength="3"
8.     [ngClass]="{'form-control': true, 'is-invalid': name.touched && name.invalid}"
9.     name="userName"
10.    [(ngModel)]="userData.name"
11.  />
12.</div>
13.
```

راجع السطر 8

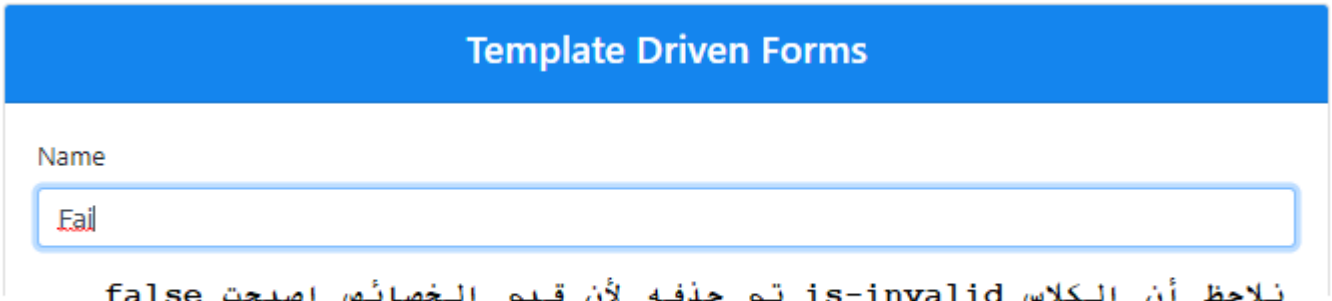
لنرى ما قمنا به من تعديلات على Form:



بداية عند تشغيل النموذج ووضع المؤشر على الأداة المستهدفة، نلاحظ لم يحدث



لكن عند لمسنا لأي مكان خارج الأداة يظهر الأطار الأحمر على الأداة بمعنى اضاف الكلاس is-invalid لأن قيمة touched تساوي true لأننا لمسنا الأداة وقيمة required ايضاً تساوي true لأننا تركنا الأداة فارغة



نلاحظ أن الكلاس is-invalid تم حذفه لأن قيم الخصائص اصبحت false حيث أن الأداة تحتوي على قيمة وبنفس الوقت عدد الخانات من ثلاث خانات فأكثر

الآن لنطبق ما تعلمناه على هذه الأداة لبقية الأدوات:

ملف app.component.html

```
1. <div class="container-fluid">
2.   <div
3.     class="card mx-auto col-sm-12 col-md-6 col-lg-6 col-xl-6"
4.     style="padding-top: 0"
5.   >
6.     <div class="card-header">
7.       <h4 class="text-center">Template Driven Forms</h4>
8.     </div>
9.
10.    <div class="card-body">
11.      <form #userForm="ngForm">
12.        <div class="form-group">
13.          <label for="">Name</label>
14.          <input
15.            type="text"
16.            #name="ngModel"
17.            required
18.            minlength="3"
19.            [ngClass]="{'form-control': true, 'is-
   invalid': name.touched && name.invalid}"
20.            name="userName"
21.            [(ngModel)]="userData.name"
22.          />
23.        </div>
24.
25.        <div class="form-group">
26.          <label>E-Mail</label>
27.          <input
28.            type="email"
29.            #email="ngModel"
30.            required
31.            pattern="[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,}$"
32.            [ngClass]="{'form-control': true, 'is-
   invalid': email.touched && email.invalid}"
```

```
33.            name="email"
34.            [(ngModel)]="userData.email"
35.          />
36.        </div>
37.
38.        <div class="form-group">
39.          <label>Password</label>
40.          <input
41.            type="password"
42.            #password="ngModel"
43.            autocomplete
44.            required
45.            pattern="(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{6,}"
46.            [ngClass]="{'form-control': true, 'is-
   invalid': password.touched && password.invalid}"
47.            name="password"
48.            [(ngModel)]="userData.password"
49.          />
50.        </div>
51.
52.        <div class="form-group">
53.          <label>Confirm Password</label>
54.          <input
55.            type="password"
56.            #confirmPassword="ngModel"
57.            autocomplete
58.            required
59.            [ngClass]="{'form-control': true, 'is-
   invalid': confirmPassword.touched && confirmPassword.invalid}"
60.            name="confirmPassword"
61.            ngModel
62.          />
63.        </div>
64.
65.        <div class="form-group">
66.          <label>Phone</label>
67.          <input
68.            type="tel"
69.            #phone="ngModel"
70.            required
71.            pattern="^\d{10}$"
72.            maxlength="10"
73.            [ngClass]="{'form-control': true, 'is-
   invalid': phone.touched && phone.invalid}"
74.            name="phone"
75.            [(ngModel)]="userData.phone"
76.          />
77.        </div>
78.
79.        <div class="form-group">
80.          <label>Topics</label>
81.          <select
82.            class="custom-select"
```

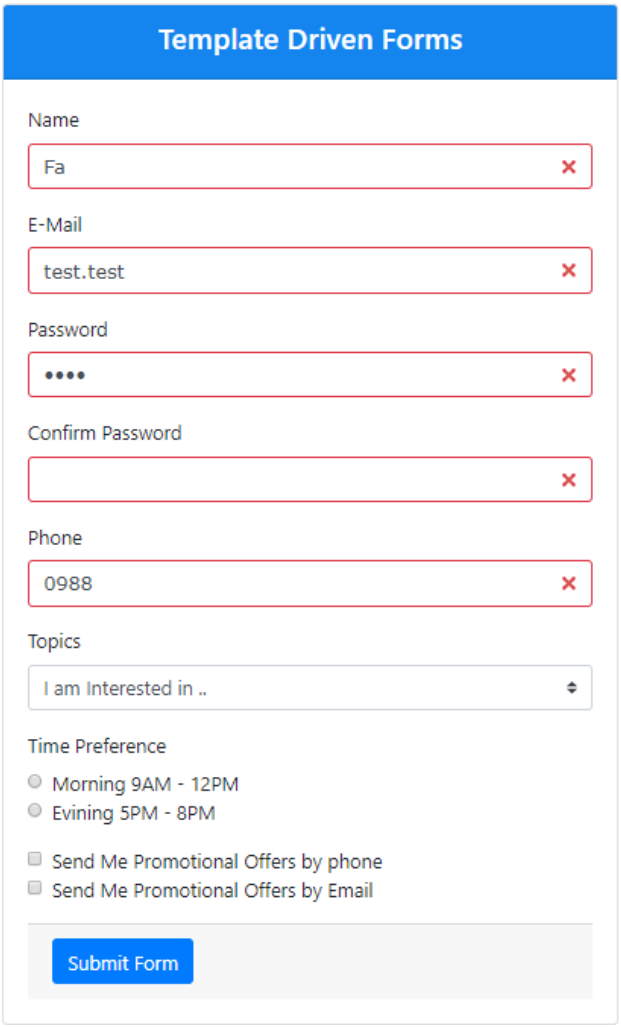
```
            #topic="ngModel"
            name="topics"
            [(ngModel)]="userData.topic"
          >
            <option value="">I am Interested in ..</option>
            <option *ngFor="let topic of topics">{{topic}}</option>
          </select>
        </div>

        <div class="mb-3">
          <label>Time Preference</label>
          <div class="form-check">
            <input
              class="form-check-input"
              type="radio"
              #timePreference="ngModel"
              required
                 name="TimePreference"
                 [(ngModel)]="userData.timePreference"
                 value="Morning"
               />
               <label class="form-check-label">Morning 9AM - 12PM</label>
             </div>
             <div class="form-check">
               <input
                 class="form-check-input"
                 type="radio"
                 #timePreference="ngModel"
                 required
                 name="TimePreference"
                 [(ngModel)]="userData.timePreference"
                 value="Evining"
               />
               <label class="form-check-label">Evining 5PM - 8PM</label>
             </div>
           </div>

           <div class="form-check mb-3">
             <input
               class="form-check-input"
               type="checkbox"
               #subscribePhone="ngModel"
               name="subscribePhone"
               [(ngModel)]="userData.subscribePhone"
             />
             <label class="form-check-label">
               Send Me Promotional Offers by phone</label
             >
             <br />
             <input
               class="form-check-input"
               type="checkbox"
               #subscribeEmail="ngModel"
```

```
136.               name="subscribeEmail"
137.               [(ngModel)]="userData.subscribeEmail"
138.             />
139.             <label class="form-check-label">
140.               Send Me Promotional Offers by Email</label
141.             >
142.           </div>
143.
144.           <div class="card-footer text-muted">
145.             <button class="btn btn-primary" type="submit">Submit Form</button>
146.           </div>
147.         </form>
148.       </div>
149.     </div>
150.   </div>
151.
```

راجع الاسطر (19 – 32 – 46 – 59 – 73)

ملاحظة: بعض الأدوات لم اضيف الأمر السابق لها كأداة radio او أداة checkbox لحدم الحاجة لهذا النوع من التغذية الراجعة وسوف أكتفي بإظهار رسائل الخطأ.

الآن لنرى ما قمنا به من تعديلات على النموذج:



الآن لنقوم بإظهار رسائل توضح للمستخدم نوع الخطأ، مثل القيمة فارغة أو صيغة البريد الإلكتروني غير صحيحة أو رقم الهاتف لابد أن يحتوي على ١٠ خانات..الخ، وسوف نقوم بهذا الأمر بالأعتماد ايضاً على الخصائص التي تعلمناها في الدرس السابق ولكن هنا سوف نستخدم الدايركتيف nglf بدلاً من class binding، ولفهم أعق عن nglf الرجاء مراجعة الكتاب الثالث من هذه السلسلة.

ولنبدأ في أول أداة وهي أداة أدخال الأسم، هنا لدينا نوعين من الأخطاء الأول أن تكون فارغة ويمكن معرفة ذلك عن طريق الخاصية required والثاني ان تكون الخانات أقل من ٣ ويمكن معرفة ذلك عن طريق الخاصية minlength، مع إضافة بعض كلاسات bootstrap لتجميل رسائل الخطأ، بحيث تكون الاسطر البرمجية بالشكل التالي:

```
<small class="alert alert-danger text-center w-100" *ngIf="name.touched && name.errors?.required">حقل الأسم مطلوب</small>
<small class="alert alert-danger text-center w-100" *ngIf="name.touched &&
name.errors?.minlength">القيمة أقل من ثلاث خانات</small>
```

ومعنى الكود السابق إنه في حالة أن المستخدم لمس الأداة وبنفس الوقت قيمة الخاصية required تساوي true أضف العنصر <small> أما إذا المستخدم لمس الأداة وكانت قيمة الخاصية minlength تساوي true أضف عنصر <small> الثاني.

نلاحظ أن كلاسات bootstrap متكررة وايضاً الامر name.touched لكلا العنصرين لذلك نستطيع إعادة كتابة الكود بشكل أكثر أحترافية وتقليل للتكرار وذلك بالأستفادة من ميزة [ngClass] وأحد كلاسات bootstrap الذي اسمه d-none ومهمة هذا الكلاس أخفاء أو اظهار عنصر معين، بحيث يكون الكود بعد التعديل:

```
<div [ngClass]="{'alert': true, 'alert-danger': true, 'text-center': true, 'w-100': true, 'd-none': name.untouched || name.valid}">

    <small *ngIf="name.errors?.required">حقل الأسم مطلوب</small>

    <small *ngIf="name.errors?.minlength">القيمة أقل من ثلاث خانات</small>

</div> >
```

الآن لنضيف الكود السابق للأداة المستهدفة:

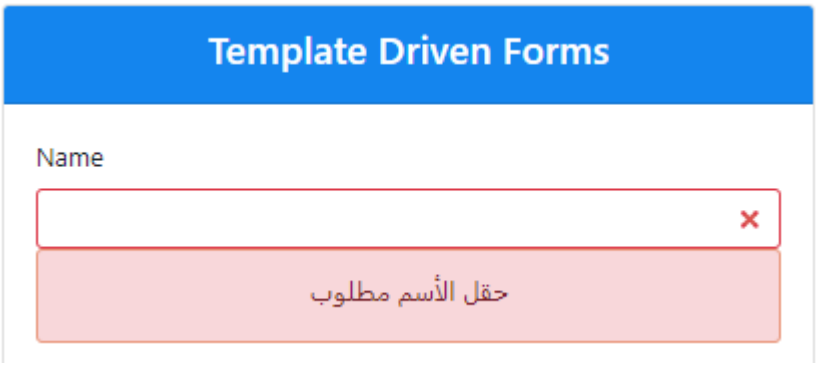
app.component.html جزء من ملف

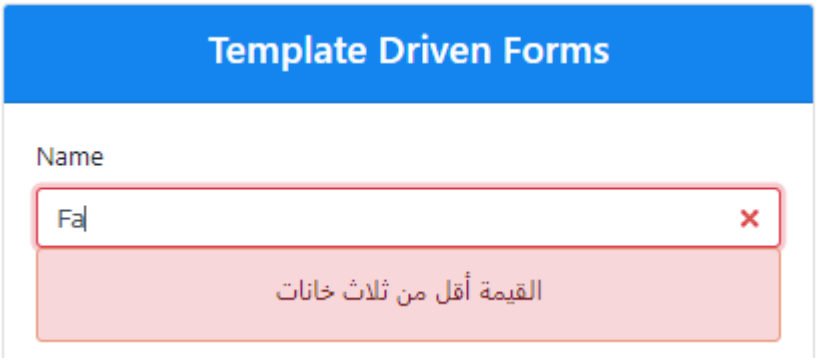
```
1. <div class="form-group">
2.   <label for="">Name</label>
3.   <input
4.     type="text"
5.     #name="ngModel"
6.     required
7.     minlength="3"
8.     [ngClass]="{'form-control': true, 'is-invalid': name.touched && name.invalid}"
9.     name="userName"
10.    [(ngModel)]="userData.name"
11.  />
12.  <div
13.    [ngClass]="{'alert': true, 'alert-danger': true, 'text-center': true, 'w-
   100': true, 'd-none': name.untouched || name.valid}"
14.  >
```

```
15.     <small *ngIf="name.errors?.required">حقل الأسم مطلوب</small>
16.     <small *ngIf="name.errors?.minlength">القيمة أقل من ثلاث خانات</small>
17.   </div>
18. </div>
19.
```

راجع الأسطر من 12 إلى 17

الآن لنرى ما قمنا به من تعديلات على النموذج:





الآن لنطبق ما تعلمناه هنا على بقية الأدوات مع مراعاة الأختلاف في انواع رسائل الخطأ والتعامل مع الخصائص - pattern maxlength - minlength - required:

```
1. <div class="container-fluid">
2.   <div
3.     class="card mx-auto col-sm-12 col-md-6 col-lg-6 col-xl-6"
4.     style="padding-top: 0"
5.   >
6.     <div class="card-header">
7.       <h4 class="text-center">Template Driven Forms</h4>
8.     </div>
9.
10.    <div class="card-body">
11.      <form #userForm="ngForm">
12.        <div class="form-group">
13.          <label for="">Name</label>
14.          <input
15.            type="text"
16.            #name="ngModel"
```

```
17.            required
18.            minlength="3"
19.            [ngClass]="{'form-control': true, 'is-
   invalid': name.touched && name.invalid}"
20.            name="userName"
21.            [(ngModel)]="userData.name"
22.          />
23.          <div
24.            [ngClass]="{'alert': true, 'alert-danger': true, 'text-center': true, 'w-
   100': true, 'd-none': name.untouched || name.valid}"
25.          >
26.            <small *ngIf="name.errors?.required">حقل الأسم مطلوب</small>
27.            <small *ngIf="name.errors?.minlength"
28.              >القيمة أقل من ثلاث خانات</small
29.            >
30.          </div>
31.        </div>
32.
33.        <div class="form-group">
34.          <label>E-Mail</label>
35.          <input
36.            type="email"
37.            #email="ngModel"
38.            required
39.            pattern="[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,}$"
40.            [ngClass]="{'form-control': true, 'is-
   invalid': email.touched && email.invalid}"
41.            name="email"
42.            [(ngModel)]="userData.email"
43.          />
44.          <div
45.            [ngClass]="{'alert': true, 'alert-danger': true, 'text-center': true, 'w-
   100': true, 'd-none': email.untouched || email.valid}"
46.          >
47.            <small *ngIf="email.errors?.required"
48.              >حقل البريد الإلكتروني مطلوب</small
49.            >
50.            <small *ngIf="email.errors?.pattern"
51.              >صيغة البريد الإلكتروني غير صحيحة</small
52.            >
53.          </div>
54.        </div>
55.
56.        <div class="form-group">
57.          <label>Password</label>
58.          <input
59.            type="password"
60.            #password="ngModel"
61.            autocomplete
62.            required
63.            pattern="(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{6,}"
64.            [ngClass]="{'form-control': true, 'is-
   invalid': password.touched && password.invalid}"
```

```
65.            name="password"
66.            [(ngModel)]="userData.password"
67.          />
68.          <div
69.            [ngClass]="{'alert': true, 'alert-danger': true, 'text-center': true, 'w-
   100': true, 'd-none': password.untouched || password.valid}"
70.          >
71.            <small *ngIf="password.errors?.required"
72.              >حقل الرقم السري مطلوب</small
73.            >
74.            <small *ngIf="password.errors?.pattern"
75.              > كلمة السر لابد أن تكون 6 خانات ما بين حروف كبيرة وصغيرة
76.              وأرقام</small
77.            >
78.          </div>
79.        </div>
80.
81.        <div class="form-group">
82.          <label>Confirm Password</label>
83.          <input
84.            type="password"
85.            #confirmPassword="ngModel"
86.            autocomplete
87.            required
88.            [ngClass]="{'form-control': true, 'is-
   invalid': confirmPassword.touched && confirmPassword.invalid}"
89.            name="confirmPassword"
90.            ngModel
91.          />
92.          <div
93.            [ngClass]="{'alert': true, 'alert-danger': true, 'text-center': true, 'w-
   100': true, 'd-none': confirmPassword.untouched || confirmPassword.valid}"
94.          >
95.            <small *ngIf="confirmPassword.errors?.required"
96.              >حقل إعادة كلمة السر مطلوب</small
97.            >
98.          </div>
99.        </div>
100.
101.         <div class="form-group">
102.           <label>Phone</label>
103.           <input
104.             type="tel"
105.             #phone="ngModel"
106.             required
107.             pattern="^\d{10}$"
108.             maxlength="10"
109.             [ngClass]="{'form-control': true, 'is-
   invalid': phone.touched && phone.invalid}"
110.             name="phone"
111.             [(ngModel)]="userData.phone"
112.           />
113.           <div
```

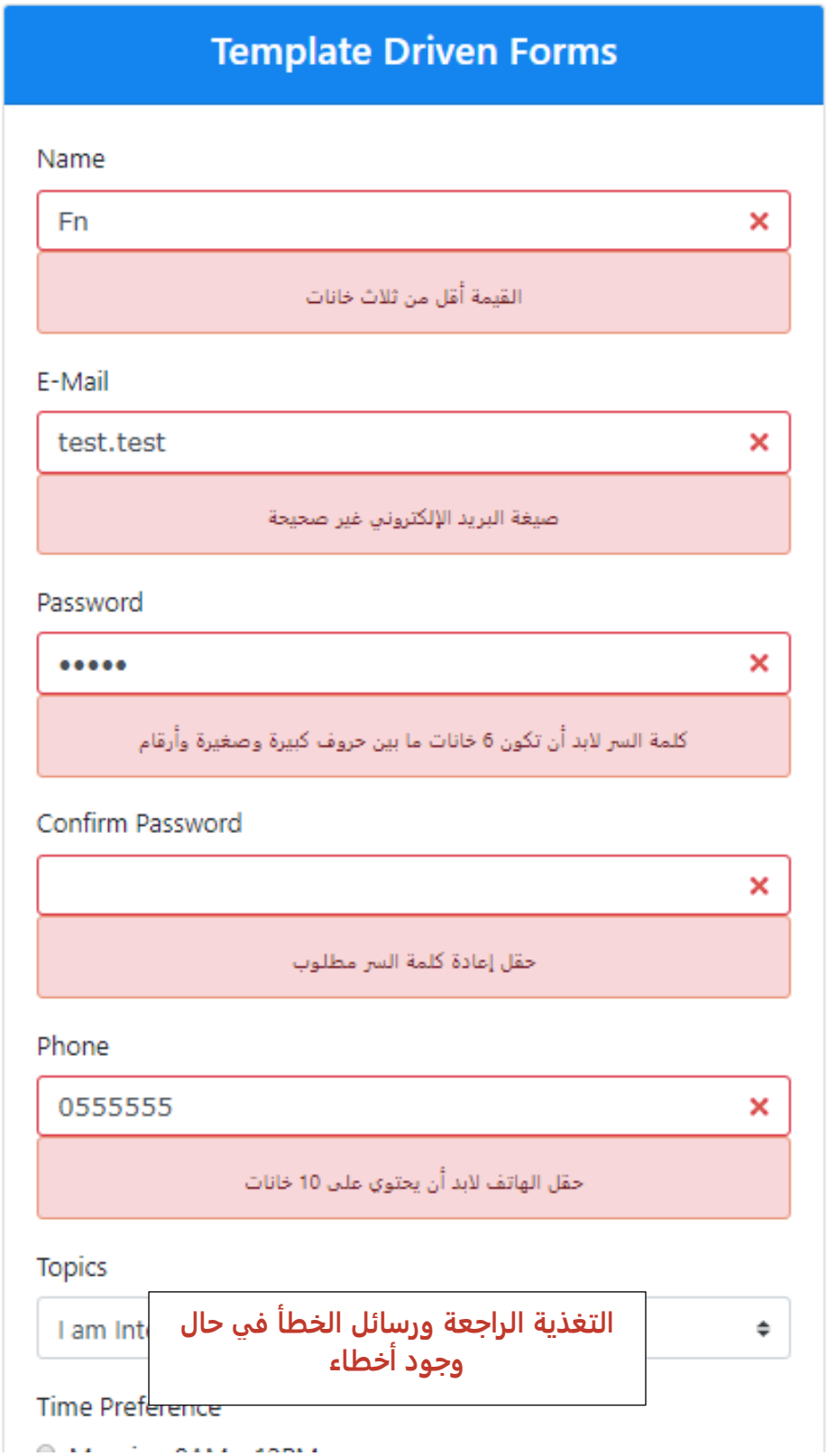
```
114.              [ngClass]="{'alert': true, 'alert-danger': true, 'text-
   center': true, 'w-100': true, 'd-none': phone.untouched || phone.valid}"
115.            >
116.              <small *ngIf="phone.errors?.required">حقل الهاتف مطلوب</small>
117.              <small *ngIf="phone.errors?.pattern"
118.                > حقل الهاتف لابد أن يحتوي على 10 خانات</small
119.              >
120.            </div>
121.          </div>
122.
123.          <div class="form-group">
124.            <label>Topics</label>
125.            <select
126.              class="custom-select"
127.              #topic="ngModel"
128.              name="topics"
129.              [(ngModel)]="userData.topic"
130.            >
131.              <option value="">I am Interested in ..</option>
132.              <option *ngFor="let topic of topics">{{topic}}</option>
133.            </select>
134.          </div>
135.
136.          <div class="mb-3">
137.            <label>Time Preference</label>
138.            <div class="form-check">
139.              <input
140.                class="form-check-input"
141.                type="radio"
142.                #timePreference="ngModel"
143.                required
144.                name="TimePreference"
145.                [(ngModel)]="userData.timePreference"
146.                value="Morning"
147.              />
148.              <label class="form-check-label">Morning 9AM - 12PM</label>
149.            </div>
150.            <div class="form-check">
151.              <input
152.                class="form-check-input"
153.                type="radio"
154.                #timePreference="ngModel"
155.                required
156.                name="TimePreference"
157.                [(ngModel)]="userData.timePreference"
158.                value="Evining"
159.              />
160.              <label class="form-check-label">Evining 5PM - 8PM</label>
161.            </div>
162.          </div>
163.
164.          <div class="form-check mb-3">
165.            <input
```

```
              class="form-check-input"
              type="checkbox"
              #subscribePhone="ngModel"
              name="subscribePhone"
              [(ngModel)]="userData.subscribePhone"
            />
            <label class="form-check-label">
              Send Me Promotional Offers by phone</label
            >
            <br />
            <input
              class="form-check-input"
              type="checkbox"
              #subscribeEmail="ngModel"
              name="subscribeEmail"
              [(ngModel)]="userData.subscribeEmail"
            />
            <label class="form-check-label">
              Send Me Promotional Offers by Email</label
            >
          </div>

          <div class="card-footer text-muted">
            <button class="btn btn-primary w-100" type="submit">
              Submit Form
            </button>
          </div>
        </form>
      </div>
    </div>
  </div>

```

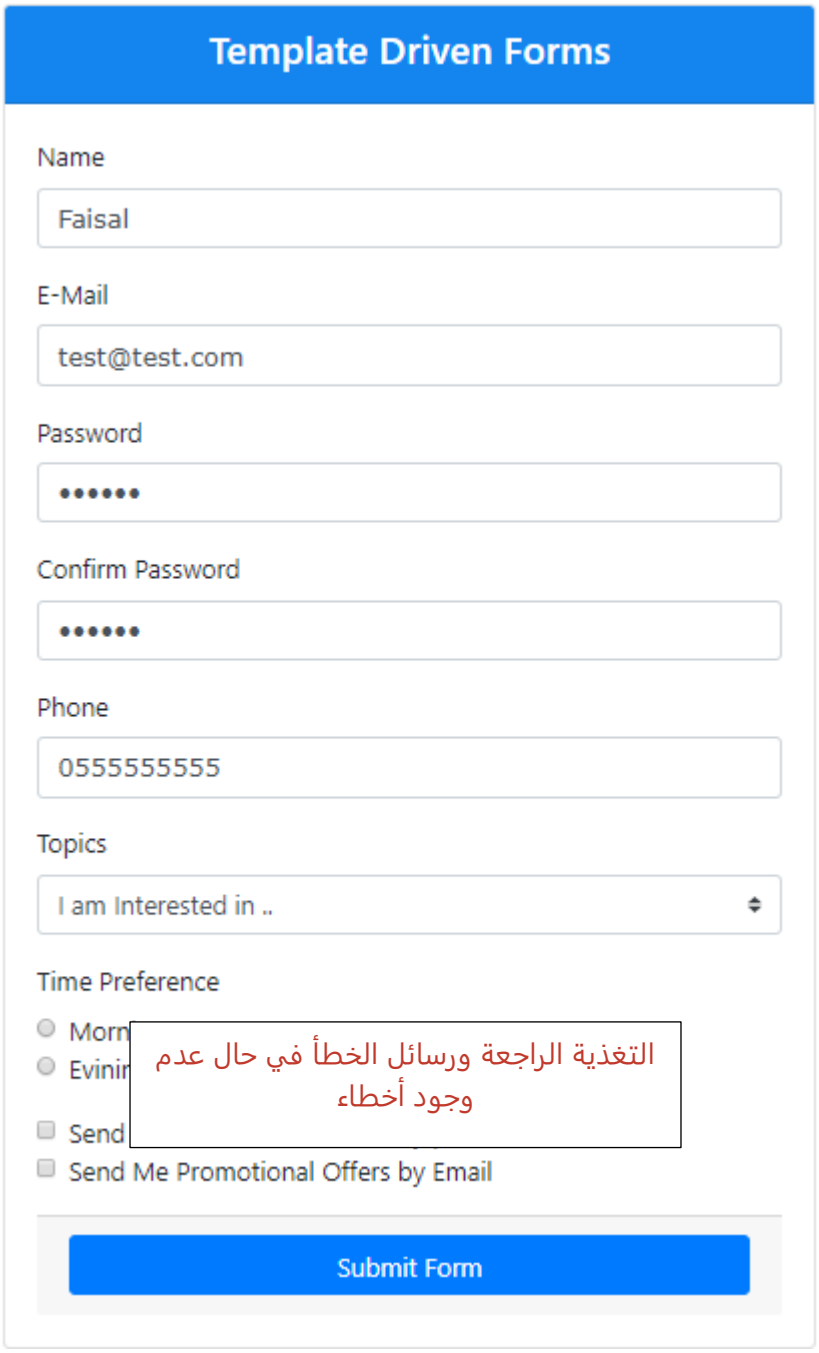
راجع الاسطر (من 23 – 30) – (44 – 53) – (68 – 78) – (92 – 98) – (113 – 120)

ملاحظة: هنالك بعض الأدوات لم نضع لها validation لأن لها تعامل معين وسوف نتركها للجزء التالي custom validation بإذن الله.

الآن لنرى ما قمنا به من تعديلات على النموذج Form:



التغذية الراجعة ورسائل الخطأ في حال وجود أخطاء



إلى هنا نكون أنهينا هذا الجزء ومتبقي لنا نوع آخر من validation سوف نتكلم عنه في الجزء التالي بإذن الله.

4.2. Custom Validations:

هنالك بعض الأحيان انواع من التحقق من الصحة لا توفرها لنا angular TDF بشكل جاهز ولابد لنا من بنائها بأنفسنا مثل منع المستخدم من إدخال بعض الاسماء او أن تكون إعادة أدخال كلمة السر مساوية لكلمة السر...الخ، لذلك يتوجب على المطور بناء التحقق من الصحة الخاص به او ما يسمى custom validation، وهذا النوع ليس له قواعد ثابته وانما يرجع لطريقة المبرمج في بناء الكود الخاص به لتحقق من صحة البيانات المدخلة لنموذج Form ولكن في نهاية الأمر لابد أن يُرجع الكود قيمة منطقية true أو false لكي نستفيد منها في شروط اظهار واخفاء رسائل الخطأ كما كنا نفعل فيما سبق.

الآن لنبدأ بأول أداة وهي أداة الأسم والمطلوب هو منع المستخدم من أدخال الأسم admin والاسم administrator في الحقل وايضاً الا تكون القيمة رقمية فقط، وللقيام بذلك نقوم بكتابة الأكواد التالية في ملف class - ملف app.component.ts، كالتالي:

أولاً: انشاء متغير من النوع boolean وليكن اسمه nameHasError وأُعطيه قيمة مبدأية false:

```
nameHasError: boolean = false;
```

ثانياً: انشاء مصفوفة نصية وليكن اسمها names تحتوي على هذين الأسمين:

```
names: string[] = ['admin', 'administrator'];
```

ثالثاً: انشاء دالة وليكن اسمها validateName وتستقبل باراميتر واحد وهو قيمة أداة أدخال الأسم، ومهمة هذه الدالة هو البحث عن القيمة التي استقبلتها من الأداة في المصفوفة فإذا وجدت قيمة تجعل قيمة المتغير nameHasError يساوي true وإن لم تجد تجعل قيمته تساوي false، وايضاً تقوم بمهمة أخرى وهي التأكد من أن القيمة ليست رقمية فقط ولا تبدأ برقم ويمكن ذلك من خلال الاستفادة من الدالة isNaN() التي تقدمها لنا javascript حيث أن هذه الدالة تُرجع true إذا كانت القيمة المدخلة نصية، وتُرجع false إذا كانت القيمة رقمية، بصياغة أخرى تقوم الدالة validateName بالبحث في المصفوفة عن قيمة الأداة فإذا وجدت قيمة مطابقة (أو) الدالة isNaN ارجعت القيمة false (و) لم تكن فارغة أجعل قيمة المتغير nameHasError يساوي true:

```
  validateName (value) {
    this.names.find(val => val === value || (!isNaN(value) && value !== '') ? this.nameHasError = true :
this.nameHasError = false);
  }
```

لا تقلق عزيزي المتعلم من السطر السابق فقد تجد فيه نوع من الصعوبة في حال كنت مبتدأ في برمجة Javascript بشكل عام وTyprscript على وجه الخصوص، لأن هذا الكود هو كود مختصر أو ما يسمى sugar syntax للكود التالي:

```
validateName (value) {
  if ( value === ''  ) {
     this.nameHasError = false;
  } else if (isNaN(value) === false) {
     this.nameHasError = true;
  } else {
    this.names.find( (val) => {
      if (val === value) {
        return this.nameHasError = true;
      } else {
        return this.nameHasError = false;
      }
    });
```

```
}
```

رابعاً: تنفذ الدالة في الحدث input الخاص في أداة إدخال الاسم مع تمرير قيمة الأداة لهذه الدالة:

```
(input) = "validateName (name.value)"
```

الآن لنضيف الأكواد في الخطوات أولاً وثانياً وثالثاً في ملف class - ملف app.component.ts والخطوة رابعاً في ملف template - ملف app.component.html.

ملف app.component.ts

```
1. import {Component, OnInit} from '@angular/core';
2. import { User } from './user';
3.
4. @Component({
5.   selector: 'app-root',
6.   templateUrl: './app.component.html',
7.   styleUrls: ['./app.component.css']
8. })
9.
10.export class AppComponent implements OnInit {
11.
12.  names: string[] = ['admin', 'administrator'];
13.  nameHasError: boolean = false;
14.  topics = ['Angular', 'React', 'Vue'];
15.  userData = new User ('', '', null, null, '', '', false, false);
16.
17.  constructor() { }
18.
19.  ngOnInit() {}
20.
21.  validateName( value ) {
22.    this.names.find(val => val === value || (!isNaN(value) && value !== '') ? this.name
   HasError = true : this.nameHasError = false);
23.  }
24.
25.}
26.
```

راجع الاسطر (12 – 13 – 21 – 22 – 23)

جزء من ملف app.component.html

```
1. <div class="form-group">
2.   <label for="">Name</label>
3.   <input
4.     type="text"
5.     #name="ngModel"
6.     (input)="validateName (name.value)"
7.     required
8.     minlength="3"
9.     [ngClass]="{'form-control': true, 'is-invalid': name.touched && name.invalid}"
10.    name="userName"
11.    [(ngModel)]="userData.name"
12.  />
```

```
13.   <div
14.     [ngClass]="{'alert': true, 'alert-danger': true, 'text-center': true, 'w-
    100': true, 'd-none': name.untouched || name.valid}"
15.   >
16.     <small *ngIf="name.errors?.required">حقل الأسم مطلوب</small>
17.     <small *ngIf="name.errors?.minlength">القيمة أقل من ثلاث خانات</small>
18.   </div>
19. </div>
```

راجع السطر 6

الآن أصبح لدينا متغير nameHasError تصبح قيمته true في حال وجود خطأ وتصبح قيمته false في حالة عدم وجود خطأ، وسوف نستفيد منه في اظهار رسائل الخطأ للمستخدم، وسوف أضيف هذا المتغير في جزئين الأول في الشروط الخاصة بإضافة وحذف الكلاس is-invalid والثاني لإظهار رسالة خطأ:

```
[ngClass]="{'form-control': true, 'is-invalid': (name.touched && name.invalid) || nameHasError}"
```

```
<label class="alert alert-danger text-center w-100" *ngIf="nameHasError">القيمة غير صالحة</label>
```

الآن لنضيف هذين السطرين إلى الأداة المستهدفة:

app.component.ts جزء من ملف

```
1. <div class="form-group">
2.   <label for="">Name</label>
3.   <input
4.     type="text"
5.     #name="ngModel"
6.     (input)="validateName (name.value)"
7.     required
8.     minlength="3"
9.     [ngClass]="{'form-control': true, 'is-
    invalid': (name.touched && name.invalid) || nameHasError}"
10.    name="userName"
11.    [(ngModel)]="userData.name"
12.  />
13.  <div
14.    [ngClass]="{'alert': true, 'alert-danger': true, 'text-center': true, 'w-
    100': true, 'd-none': name.untouched || name.valid}"
15.  >
16.    <small *ngIf="name.errors?.required">حقل الأسم مطلوب</small>
17.    <small *ngIf="name.errors?.minlength">القيمة أقل من ثلاث خانات</small>
18.  </div>
19.  <label class="alert alert-danger text-center w-100" *ngIf="nameHasError">
20.    القيمة غير صالحة
21.  </label>
22. </div>
```

راجع الاسطر (9 – 19 – 20 – 21)

لنشاهد التعديلات على form:

Template Driven Forms

Name

admin

القيمة غير صالحة

E-Mail

Password

Confirm Password

Phone

Topics

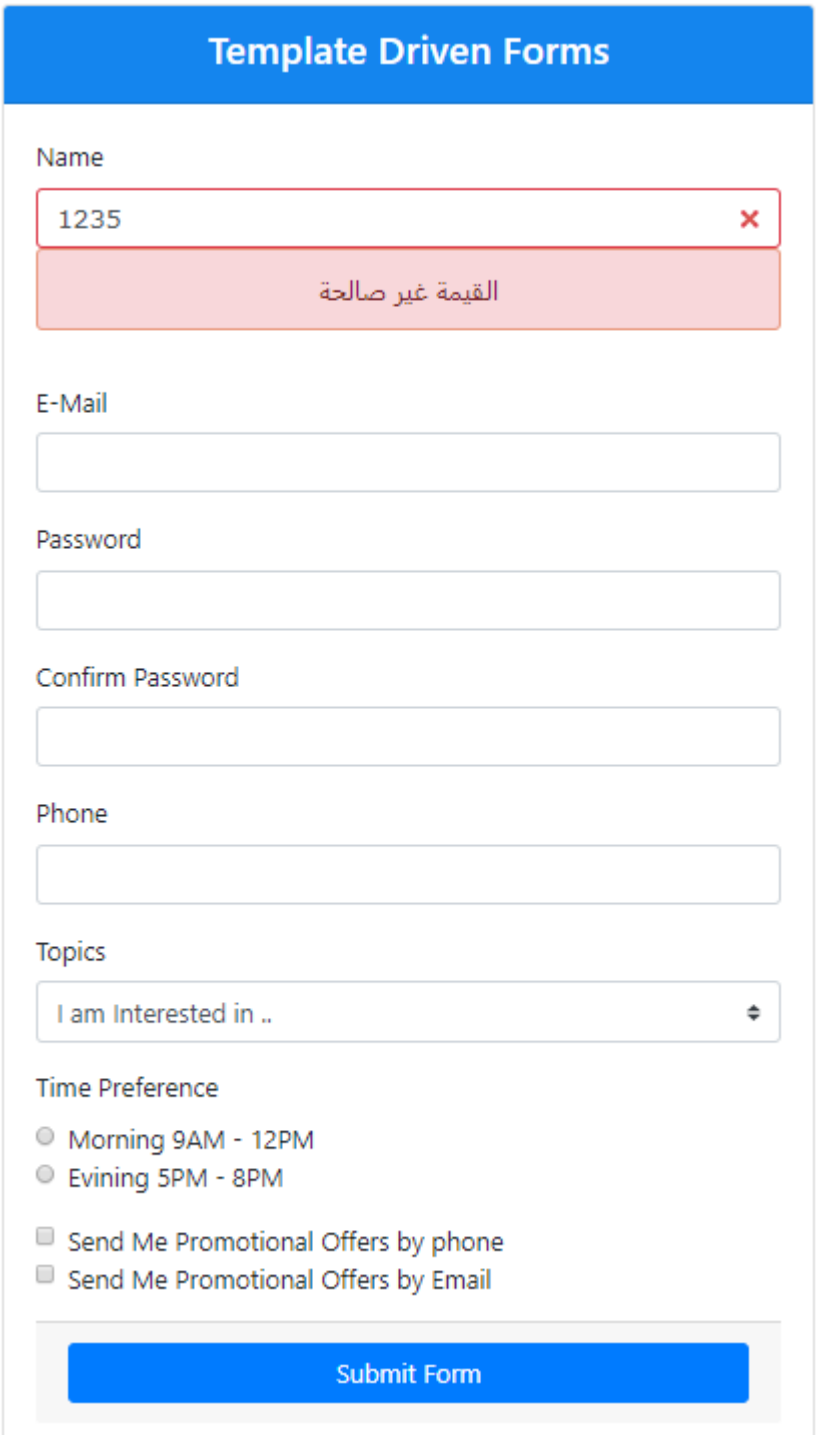
I am Interested in ..

Time Preference

- Morning 9AM - 12PM
- Evining 5PM - 8PM

- Send Me Promotional Offers by phone
- Send Me Promotional Offers by Email

Submit Form



هذا بالنسبة إلى أداة الاسم، اما بالنسبة لتحقق من أن كلمة السر مساوية لإعادة إدخال كلمة السر، فهنالك عدة طرق منها ما هو شبه تكرار لما عملناه سابقاً وهو انشاء متغير منطقي boolean ثم إنشاء دالة في الحدث input لكلا الأداتين وهذه الدالة تستقبل باراميترين هما قيمتا الأداتين ومن ثم تقارن بينهما فإذا كانتا متطابقتين جعلت قيمة المتغير true وان لم يكن جعلة قيمته false، كالتالي:

اولاً: تعريف متغير منطقي من النوع boolean وليكن اسمه passwordHasError

ثانياً: إنشاء دالة وليكن اسمها validatePassword وتستقبل باراميترين هما قيمة الأداة password وقيمة الأداة confirm password، ثم تقارن بينهما فإذا لم تتطابق القيمتين تجعل قيمة المتغير true وإلا تكون قيمته false

ثالثاً: تنفذ الدالة في الحدث input لكلا الأداتين

رابعاً: نضع شرط إنه في حال كانت قيمة المتغير وتم لمس الأداة confirm password اظهر رسالة الخطأ ولاننسى نضع ايضاً شرط المتغير في التحكم بإظهار وأخفاء الكلاس is-invalid للأداة confirm password

ملف app.component.ts

```
1. import { Component, OnInit } from '@angular/core';
2. import { User } from './user';
3.
4. @Component({
5.     selector: 'app-root',
6.     templateUrl: './app.component.html',
7.     styleUrls: ['./app.component.css']
8. })
9.
10.export class AppComponent implements OnInit {
11.
12.    names: string[] = ['admin', 'administrator'];
13.    nameHasError: boolean = false;
14.
15.    passwordHasError: boolean = false;
16.
17.    topics = ['Angular', 'React', 'Vue'];
18.    userData = new User('', '', null, null, '', '', false, false);
19.
20.    constructor() { }
21.
22.    ngOnInit() { }
23.
24.    validateName(value) {
25.        this.names.find(val => val === value || (!isNaN(value) && value !== '') ? this.
   nameHasError = true : this.nameHasError = false);
26.    }
27.
28.    validatePaaword(confirmPassVal, passVal) {
29.        passVal !== confirmPassVal ? this.passwordHasError = true : this.passwordHasErr
   or = false;
30.    }
31.
32.}
```

راجع السطر 15 - الاسطر (من 28 إلى 30)

جزء من ملف app.component.html

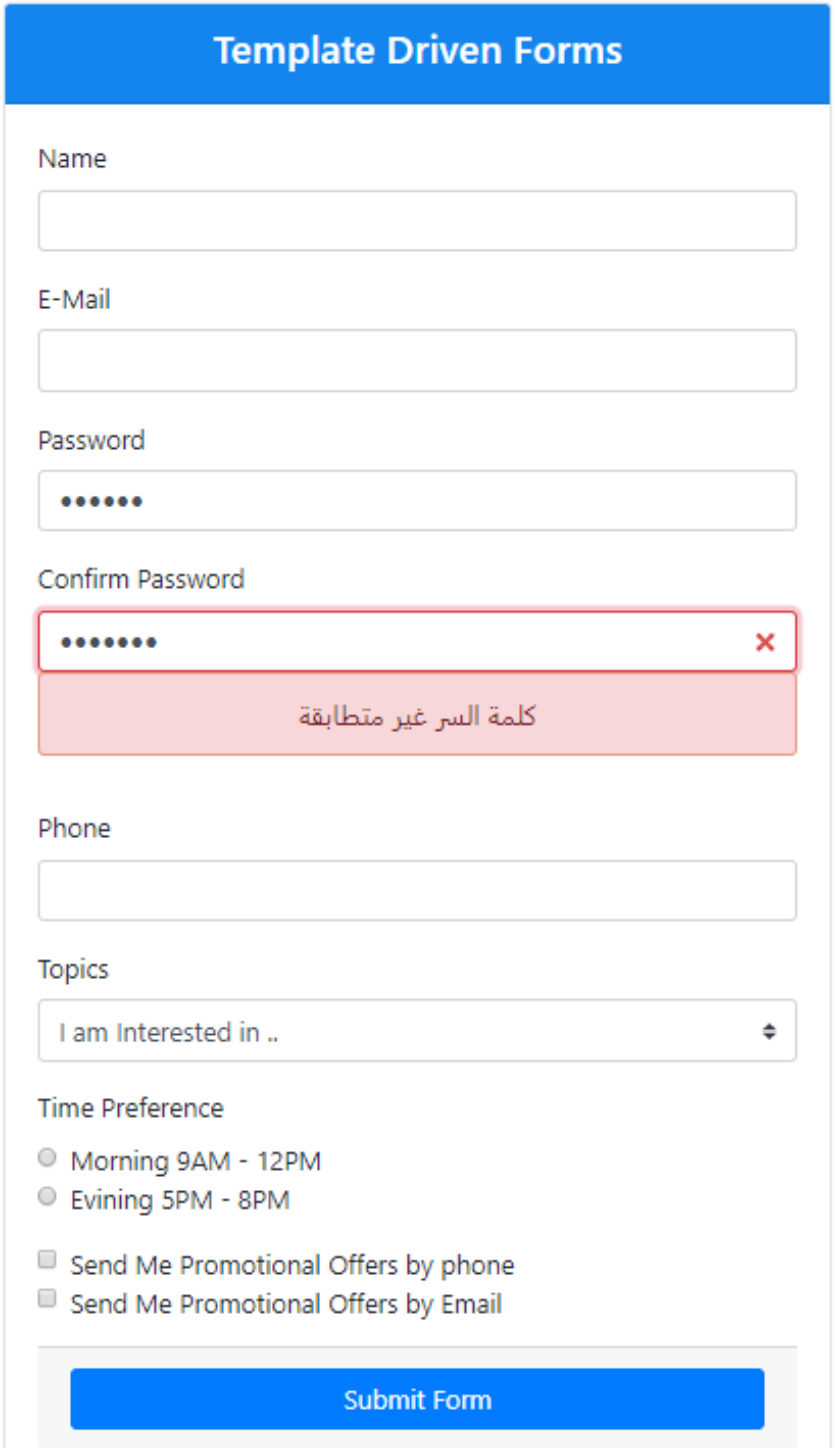
```
1. <div class="form-group">
2.   <label>Password</label>
3.   <input
4.     type="password"
5.     #password="ngModel"
6.     (input)="validatePaaword(password.value, confirmPassword.value)"
7.     autocomplete
```

```
8.      required
9.      pattern="(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{6,}"
10.     [ngClass]="{
11.         'form-control': true,
12.         'is-invalid': password.touched && password.invalid
13.     }"
14.     name="password"
15.     [(ngModel)]="userData.password"
16.   />
17.   <div
18.     [ngClass]="{
19.         'alert': true,
20.         'alert-danger': true,
21.         'text-center': true,
22.         'w-100': true,
23.         'd-none': password.untouched || password.valid
24.     }"
25.   >
26.     <small *ngIf="password.errors?.required">حقل الرقم السري مطلوب</small>
27.     <small *ngIf="password.errors?.pattern">
28.       كلمة السر لابد ان تحتوي على الأقل ست خانات
29.     </small>
30.   </div>
31. </div>
32.
33. <div class="form-group">
34.   <label>Confirm Password</label>
35.   <input
36.     type="password"
37.     #confirmPassword="ngModel"
38.     (input)="validatePaaword(password.value, confirmPassword.value)"
39.     autocomplete
40.     required
41.     [ngClass]="{
42.         'form-control': true,
43.         'is-
    invalid': (confirmPassword.invalid || passwordHasError) && confirmPassword.touched
44.     }"
45.     name="confirmPassword"
46.     ngModel
47.   />
48.   <div
49.     [ngClass]="{
50.         'alert': true,
51.         'alert-danger': true,
52.         'text-center': true,
53.         'w-100': true,
54.         'd-none': confirmPassword.untouched || confirmPassword.valid
55.     }"
56.   >
57.     <small *ngIf="confirmPassword.errors?.required">
58.       حقل إعادة كلمة السر مطلوب
59.     </small>
```

```
60.   </div>
61.   <label
62.     class="alert alert-danger text-center w-100"
63.     *ngIf="passwordHasError && confirmPassword.touched"
64.   >
65.     كلمة السر غير متطابقة
66.   </label>
67. </div>
```

راجع الاسطر (6 – 38 – 43) – (من 61 إلى 66)

الآن لنرى ما قمنا به من تعديلات على النموذج:



الآن ننتقل إلى الأداة <select> التي لم نتعامل معها إلى الآن:

جزء من ملف app.component.html

```
1. <div class="form-group">
2.   <label>Topics</label>
3.   <select
4.     class="custom-select"
5.     #topic="ngModel"
6.     name="topics"
7.     [(ngModel)]="userData.topic"
8.   >
9.     <option selected value="">I am Interested in ..</option>
10.    <option *ngFor="let topic of topics" [ngValue]="topic">{{topic}}</option>
11.  </select>
12.</div>
```

وهذه الأداة ذات التاغ select يمكن التعامل معها بطريقتين الطريقة الأولى بإستخدام الدايكرتيف والخصائص الجاهزة التي تقدمها لنا angular forms TDF وهي طريقة جيدة وتعمل بدون مشاكل في حالة إن الخاصية value الموجودة في السطر 9 في الكود السابق قيمتها فارغة ولكن ماذا لو لم تكن فارغة وتم اسناد لها القيمة default (هي قيمة تخبر المتصفح أن القيمة الابتدائية - في حالتنا هذه هي الجملة iam inteerested in- هي قيمة مقبولة للأداة في حال أختارها المستخدم من القائمة، بحيث يتم التعامل معها على حسب الهدف من النظام. وهو النظام المتبع في الأنظمة الحقيقية وخصوصاً إذا كانت البيانات قادمة من السيرفر، وفي حالتنا هذي يعتبر هذا الأمر غير مقبول فهذه الجملة لا تعتبر من البيانات الخاصة بالمستخدم وفي حال أختارها المستخدم لابد من إظهار رسالة خطأ له)، لذلك لابد من استخدام الطريقة الثانية وهي custom validation، وفي الحقيقة هو ايضاً تكرار لما قمنا به سابقاً، كالتالي:

أولاً: ننشأ متغير منطقي من النوع boolean وليكن اسمه topicHasError ونعطيه قيمة مبدأية true (السبب أن هذه الأداة لا تحتوي على الخاصية required لذلك لابد من التعامل مع هذه الأداة افتراضياً على انها invalid قيمتها غير صالحة او غير مقبولة)

ثانياً: ننشأ دالة وليكن اسمها validateTopic وتستقبل قيمة هذه الأداة ومهمتها التأكد من قيمة الأداة فإذا كانت فارغة او قيمتها default تجعل قيمة المتغير true وإلا تجعل قيمته false

ثالثاً: تُنفذ هذه الأداة في الحدث blur والحدث change للأداة

رابعاً: عن طريق المتغير يتم التعامل مع رسائل الخطأ والتغذية الراجعة للمستخدم

ملف app.component.ts

```
1. import { Component, OnInit } from '@angular/core';
2. import { User } from './user';
3.
4. @Component({
5.     selector: 'app-root',
6.     templateUrl: './app.component.html',
7.     styleUrls: ['./app.component.css']
```

```
8. })
9.
10.export class AppComponent implements OnInit {
11.
12.    names: string[] = ['admin', 'administrator'];
13.    nameHasError: boolean = false;
14.
15.    passwordHasError: boolean = false;
16.
17.    topicHasError: boolean = true;
18.
19.    topics = ['Angular', 'React', 'Vue'];
20.
21.    userData = new User('', '', null, null, '', '', false, false);
22.
23.    constructor() { }
24.
25.    ngOnInit() { }
26.
27.    validateName(value) {
28.        this.names.find(val => val === value || (!isNaN(value) && value !== '') ? this.
   nameHasError = true : this.nameHasError = false);
29.    }
30.
31.    validatePaaword(confirmPassVal, passVal) {
32.        passVal !== confirmPassVal ? this.passwordHasError = true : this.passwordHasErr
   or = false;
33.    }
34.
35.    validateTopic(value) {
36.        value === 'default' || value === '' ? this.topicHasError = true : this.topicHas
   Error = false;
37.    }
38.
39.}
40.
```

راجع السطر 17 - الأسطر (من 35 إلى 37)

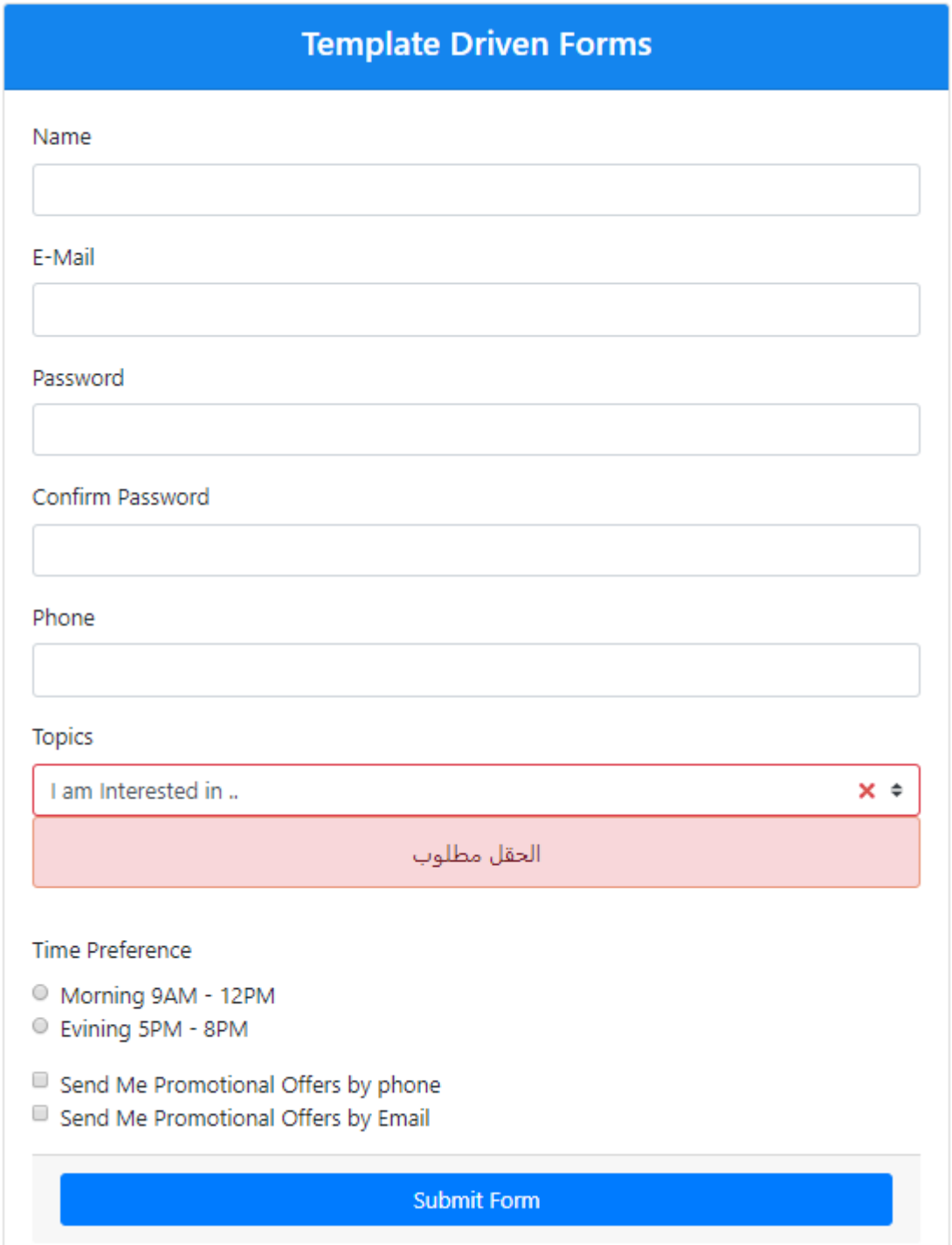
جزء من ملف app.component.html

```
1. <div class="form-group">
2.   <label>Topics</label>
3.   <select
4.     #topic="ngModel"
5.     (blur)="validateTopic(topic.value)"
6.     (change)="validateTopic(topic.value)"
7.     [ngClass]="{'custom-select': true, 'is-invalid': topicHasError && topic.touched}"
8.     name="topics"
9.     [(ngModel)]="userData.topic"
10.  >
11.    <option selected value="default">I am Interested in ..</option>
12.    <option *ngFor="let topic of topics" [ngValue]="topic">{{topic}}</option>
13.  </select>
```

```
14.   <label
15.     class="alert alert-danger text-center w-100"
16.     *ngIf="topicHasError && topic.touched"
17.   >
18.     الحقل مطلوب
19.   </label>
20. </div>
```

راجع الاسطر (5 – 6 – 7) – (من 14 إلى 19)

الآن لنرى التعديلات على النموذج:



أما آخر أداتين وهما radio وcheckbox فالتعامل معهما بسيط وسوف نتعامل معها اثناء أرسال النموذج او ما يسمى submit form، ولذلك سوف نؤجل شرحها لاحقاً.

بذلك بكون انتهى هذا الجزء، وكما قلت سابقاً custom validation ليس له طريقة ثابتة وانما هو اسلوب برمجي logic يختلف من مبرمج إلى مبرمج، انا هنا استخدمت ابسط الطرق وقد يكون هنالك مبرمج آخر يستخدم طرق أكثر احترافية وذلك عن طريق إنشاء directive خاص به ويُنشئ بداخله الخصائص والدوال ومن ثم ويقوم بتجميع كل validation بداخله، ومن ثم يضيف هذا الدايركتيف وخصائصه إلى عناصر HTML والأدوات المستهدفة، وايضاً قد يكون هنالك طرق أخرى ابسط مما استخدمنا هنا. لذلك اعرف ماذا تحتاج واستخدم المطلوب فقط بأبسط الطرق، فطريقة الدايكرتيف يُستفاد منها إذا كان لدي أكثر من نموذج في أكثر من component لذلك تجنباً لتكرار الكود يُكتب مرة واحدة ويتم استدعاءه في جميع النماذج.

5.2. Form validation and submit form:

جميع ما قمنا به سابقاً هو في مستوى control validation وهو التحقق من الصحة على مستوى الأدوات ولكن الـangular forms تقدم لنا خصائص جاهزة على مستوى form validation وهو التحقق من الصحة الخاص بالform كاملاً، وهو ما سوف نتكلم عنه في هذا الجزء مع كيفية التعامل مع زر ارسال النموذج submit form.

لو رجعنا إلى أول كلامنا عن angular TDF ا لو جدنا اننا انشأنا متغير باسم #userForm او ما يسمى template reference لـ Form الخاص بنا وربطناه بالدايركتيف ngForm وبالتالي أصبح هذا المتغير يمتلك حق الوصول لجميع الخصائص التي يقدمها هذا الدايكرتيف وما يهمنا منها حالياً هو الخاصية userForm.form.invalid وهذه الخاصية ترجع true في حال أن هنالك خطأ من التحقق من الصحة لأي أداة من أدوات النموذج – Form – ويمكن أن نستفيد من هذه الخاصية في تفعيل وتعطيل زر أرسال البيانات حيث إذا كانت قيمة الخاصية true نعطل الزر وإذا كانت false يفعل الزر بمعنى أنه لا يوجد أخطاء، ونستطيع أن نقوم بذلك عن طريقة ميزة property binding (لمعلومات أكثر عن Property Binding الرجاء مراجعة الكتاب الثاني من هذه السلسلة) التي تقدمها لنا الـangular عن طريق استخدام الخاصية disabled التي نضيفها في أداة زر ارسال البيانات، كالتالي:

```
<div class="card-footer text-muted">
    <button class="btn btn-primary w-100" [disabled]="userForm.form.invalid" type="submit">
       Submit Form
    </button>
</div>
```

حيث أن خاصية invalid التابعة لـForm تقوم بمتابعة جميع خصائص HTML5 (مثل required - minlength - pattern) التي تمت إضافتها للأدوات التي يحتويها هذا النموذج وفي حال أن أي خاصية منها كانت قيمتها تساوي true تلقائياً سوف تصبح قيمة خاصية invalid التابعة للform ايضاً تساوي true، ومن هذا المنطلق نستطيع التعامل مع الأداة radio وذلك عن طريق خاصية required التي اضفناها لها فيما سبق، بمعنى أن الزر لن يتم تفعيله في حالة انه لايوجد اي اختيار من خيارات اداة radio، أما أداة checkbox فلا تمتلك خاصية required (السبب انه في حال وضعنا لهذه الخاصية في الأداة

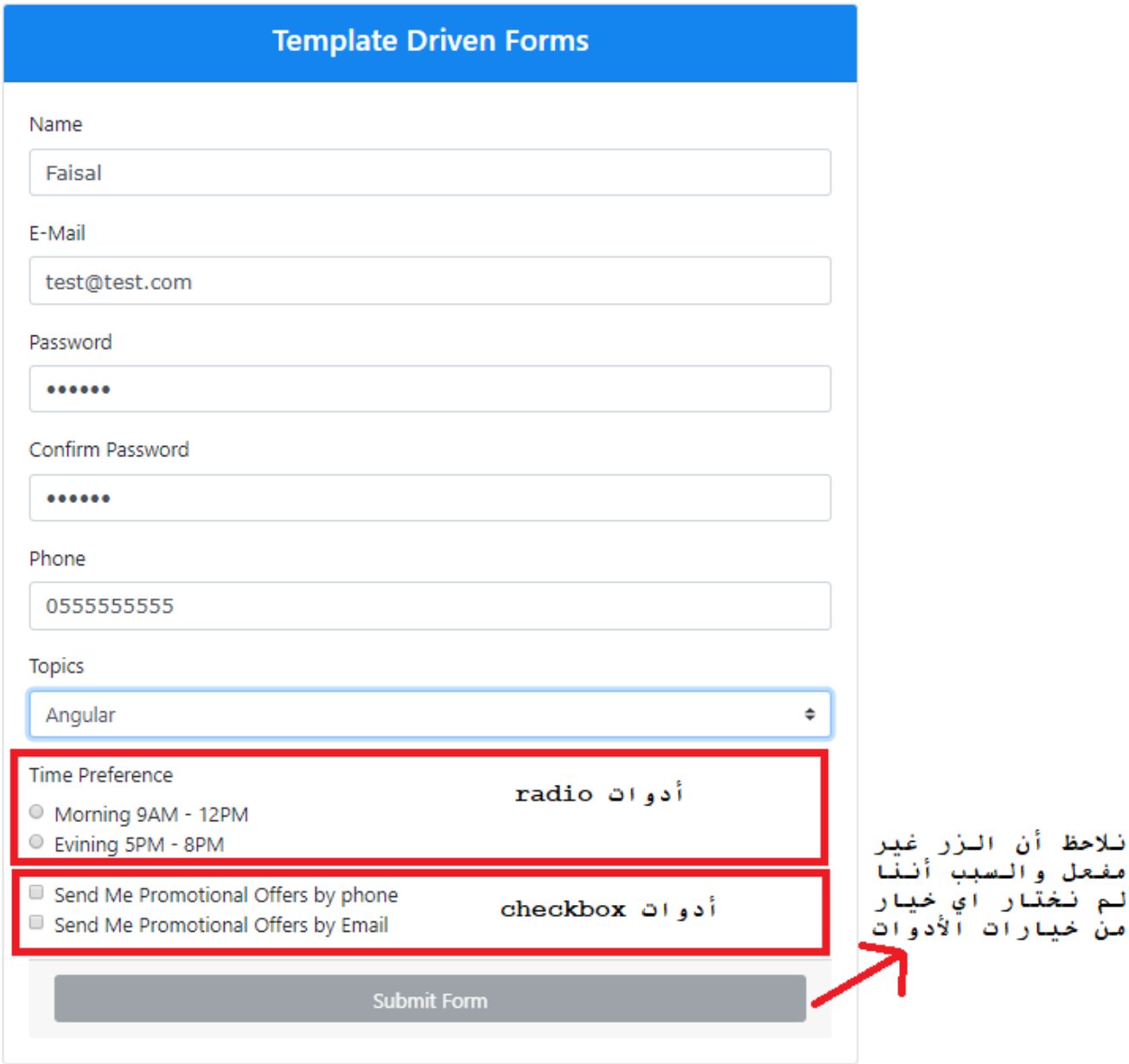
سوف يجبر المستخدم على اختيار جميع الخيارات وهذا غير منطقي) ولا اي خاصية تحقق من الصحة أخرى لذلك لحلها نكتب فقط الأمر البرمجي التالي:

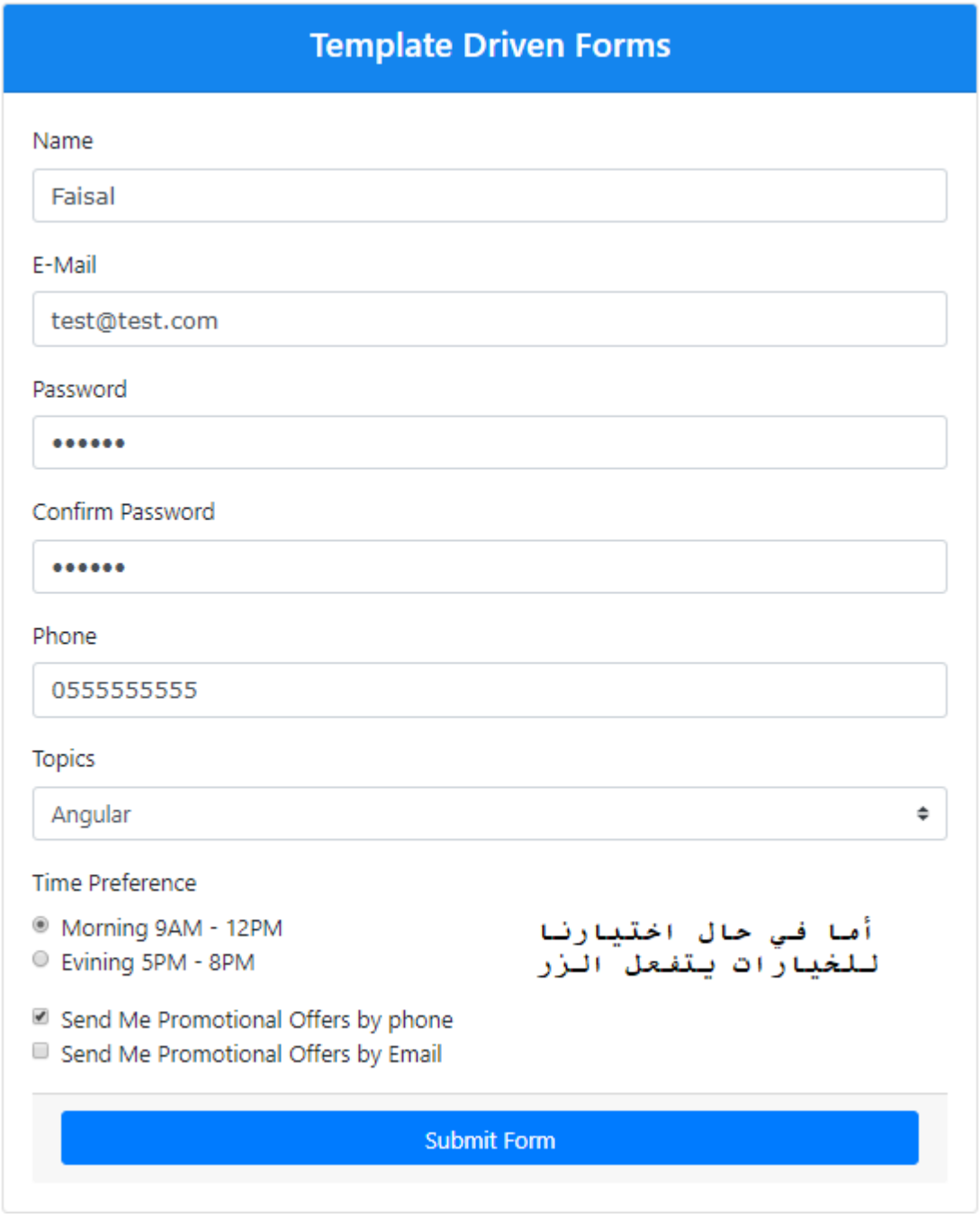
```
!subscribePhone.value && !subscribeEmail.value
```

وهذا الأمر بكل بساطة يجلب لنا قيمة أداة checkbox الأولى وقيمة أداة checkbox الثانية (وكما هو معروف ان هذه الانواع من الأدوات تُرجع قيم منطقية true او false)، الآن لنضيف هذا الأمر البرمجي إلى زر ارسال البيانات لـForm:

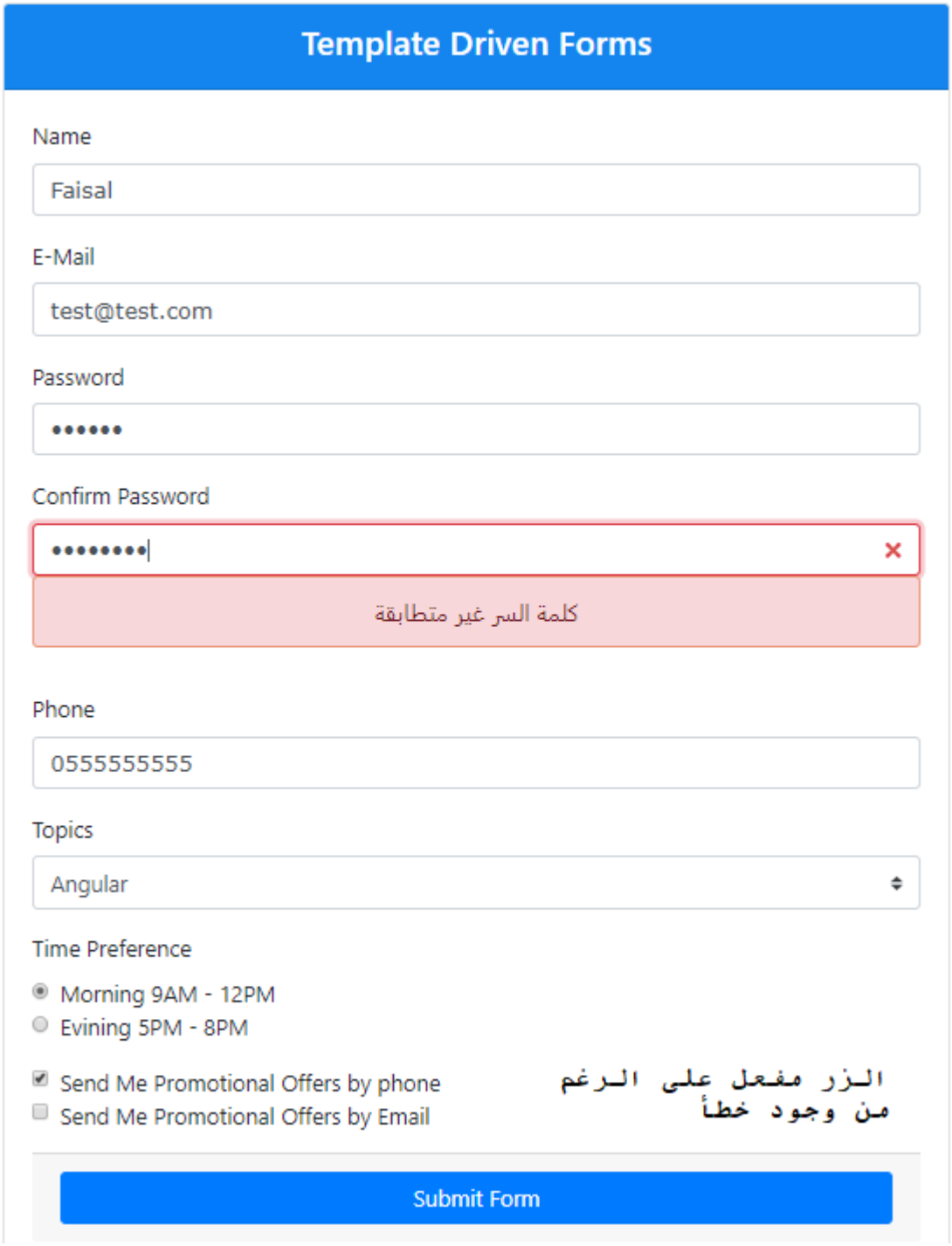
```
<div class="card-footer text-muted">
    <button class="btn btn-primary w-100" [disabled]="userForm.form.invalid || (!subscribePhone.value
&& !subscribeEmail.value)" type="submit">Submit Form</button>
</div>
```

ومعنى الكود السابق اجعل الزر في حالة تعطيل disabled إذا كان form في حالة invalid او قيمة كلا الأداتين تساوي false اي لم يتم اختيار اي خيار منهما، الآن لنقوم بمشاهدة ما قمنا به من تعديلات على النموذج:





وبذلك أنهينا التحقق من الصحة لجميع الأدوات داخل النموذج، ولكن بقي لنا أن نرتكب خطأ مقصود وهو أحد اخطاء custom validation وليكن كلمة إعادة السر لا تساوي كلمة السر ولنرى ماذا يحدث لزر:



والسبب في ذلك يعود إلى أن الشروط التي اعطيناها للزر ليبقى في حالة تعطيل هي ان يكون النموذج في حالة invalid وهذه الحالة مرتبطة بالخصائص الجاهزة مثل required وغيره وليست مرتبطة بي custom validation وايضاً الشرط الثاني يتعلق بأدوات checkbox، لذلك لابد من إضافة شروط أخرى وهي المتغيرات التي قمنا بتعريفها سابقاً للزر، كالتالي:

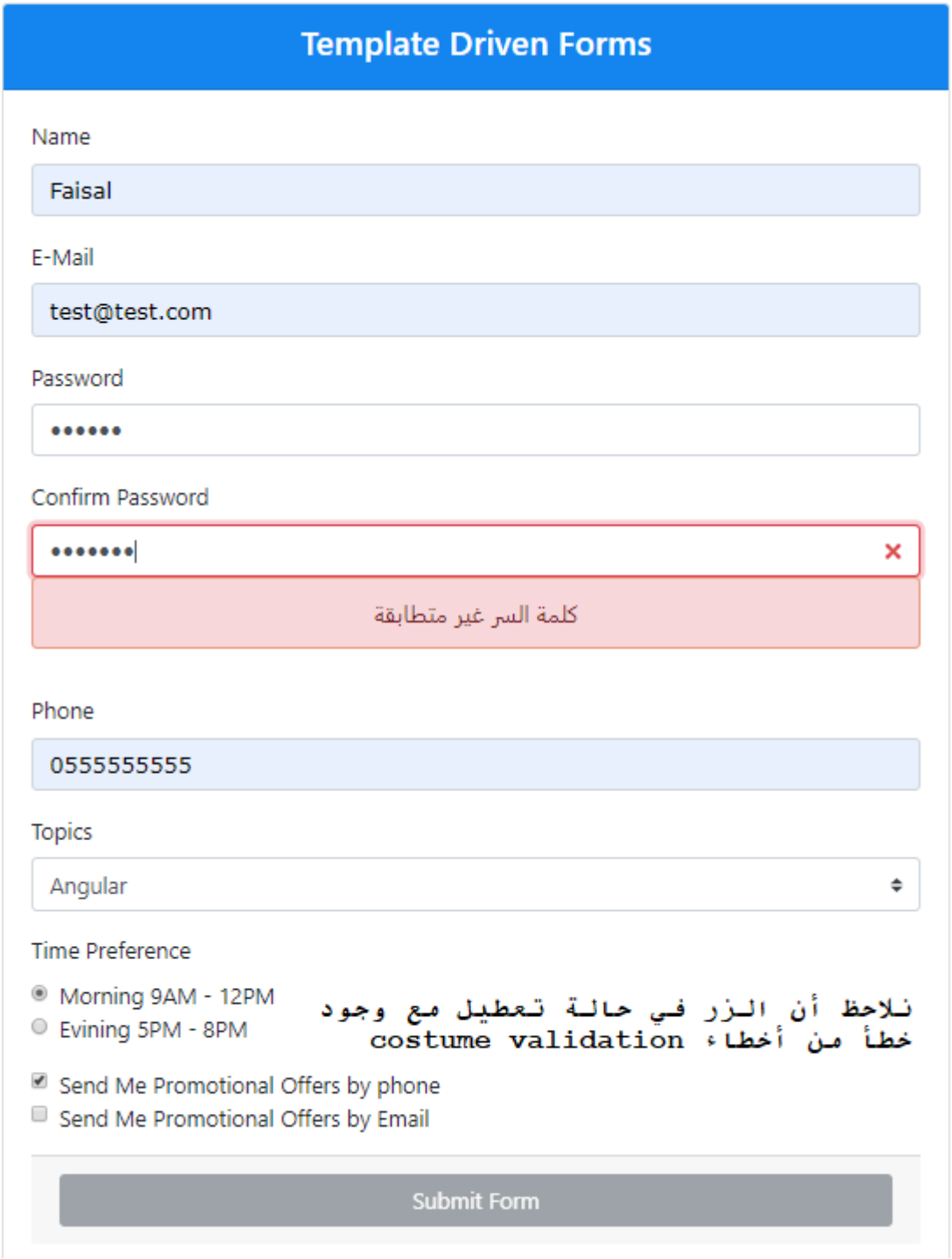
جزء من ملف app.component.html

```
1. <div class="card-footer text-muted">
2.   <button
3.     class="btn btn-primary w-100"
4.     [disabled]="
5.         userForm.form.invalid ||
6.         (!subscribePhone.value && !subscribeEmail.value) ||
7.         nameHasError ||
```

```
8.          passwordHasError ||
9.          topicHasError
10.     "
11.     type="submit"
12.   >
13.     Submit Form
14.   </button>
15.</div>
```

راجع الاسطر من 5 إلى 9

الآن لنرى ما قمنا به من تعديلات على النموذج:



بقي أخيراً أن نتعامل مع submit form، وللقيام بهذا الأمر عن طريق angular forms نقوم اولاً بإضافة الخاصية novalidate إلى تاغ <form> والسبب لوضعنا هذه الخاصية لأننا نتحقق من الصحة عن طريق الـangular forms ولا نريد من المتصفح أن يقوم بذلك عن طريق الـForms الخاصة بـHTML5 والـvalidation الخاص به، والأمر الثاني الذي نقوم به هو إضافة الحدث (ngSubmit) ايضاً في تاغ <form> حيث يقوم هذا الحدث بتنفيذ كود معين عندما يتم الضغط على زر submit الخاص بـForm وفي حالتنا هذه نريد تنفيذ دالة باسم onSubmit() تقوم باستعراض محتويات الكائن userData الذي عملنا له instance من الكلاس User وربطناه بالأدوات الخاصة بـForm الخاص، كالتالي:

ملف app.component.html

```
1. <div class="container-fluid">
2.   <div
3.     class="card mx-auto col-sm-12 col-md-6 col-lg-6 col-xl-6"
4.     style="padding-top: 0"
5.   >
6.     <div class="card-header">
7.       <h4 class="text-center">Template Driven Forms</h4>
8.     </div>
9.
10.    <div class="card-body">
11.      <form #userForm="ngForm" novalidate (ngSubmit)="onSubmit()">
12.        <div class="form-group">
13.          <label for="">Name</label>
14.          <input
15.            type="text"
16.            #name="ngModel"
17.            (input)="validateName (name.value)"
18.            required
19.            minlength="3"
20.            [ngClass]="{
21.                'form-control': true,
22.                'is-invalid': (name.touched && name.invalid) || nameHasError
23.            }"
24.            name="userName"
25.            [(ngModel)]="userData.name"
26.          />
27.          <div
28.            [ngClass]="{'alert': true, 'alert-danger': true, 'text-center': true, 'w-
   100': true, 'd-none': name.untouched || name.valid}"
29.          >
30.            <small *ngIf="name.errors?.required">حقل الأسم مطلوب</small>
31.            <small *ngIf="name.errors?.minlength">
32.              القيمة أقل من ثلاث خانات
33.            </small>
34.          </div>
35.          <label
36.            class="alert alert-danger text-center w-100"
37.            *ngIf="nameHasError"
38.            >القيمة غير صالحة</label
39.          >
40.        </div>
```

```
41.
42.         <div class="form-group">
43.           <label>E-Mail</label>
44.           <input
45.             type="email"
46.             #email="ngModel"
47.             required
48.             pattern="[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,}$"
49.             [ngClass]="{
50.                 'form-control': true,
51.                 'is-invalid': email.touched && email.invalid
52.             }"
53.             name="email"
54.             [(ngModel)]="userData.email"
55.           />
56.           <div
57.             [ngClass]="{
58.                 'alert': true,
59.                 'alert-danger': true,
60.                 'text-center': true,
61.                 'w-100': true,
62.                 'd-none': email.untouched || email.valid
63.             }"
64.           >
65.             <small *ngIf="email.errors?.required">
66.               حقل البريد الإلكتروني مطلوب
67.             </small>
68.             <small *ngIf="email.errors?.pattern"
69.               >صيغة البريد الإلكتروني غير صحيحة</small
70.             >
71.           </div>
72.         </div>
73.
74.         <div class="form-group">
75.           <label>Password</label>
76.           <input
77.             type="password"
78.             #password="ngModel"
79.             (input)="validatePaaword(password.value, confirmPassword.value)"
80.             autocomplete
81.             required
82.             pattern="(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{6,}"
83.             [ngClass]="{
84.                 'form-control': true,
85.                 'is-invalid': password.touched && password.invalid
86.             }"
87.             name="password"
88.             [(ngModel)]="userData.password"
89.           />
90.           <div
91.             [ngClass]="{
92.                 'alert': true,
93.                 'alert-danger': true,
```

```
                'text-center': true,
                'w-100': true,
                'd-none': password.untouched || password.valid
              }"
            >
              <small *ngIf="password.errors?.required">
                   حقل الرقم السري مطلوب
                </small>
                <small *ngIf="password.errors?.pattern">
                  خانات ما بين حروف كبيرة وصغيرة وأرقام 6كلمة السر لابد أن تكون
                </small>
              </div>
            </div>

            <div class="form-group">
              <label>Confirm Password</label>
              <input
                type="password"
                #confirmPassword="ngModel"
                (input)="validatePaaword(password.value, confirmPassword.value)"
                autocomplete
                required
                [ngClass]="{
                    'form-control': true,
                    'is-
   invalid': (confirmPassword.invalid || passwordHasError) && confirmPassword.touched
                  }"
                name="confirmPassword"
                ngModel
              />
              <div
                [ngClass]="{
                    'alert': true,
                    'alert-danger': true,
                    'text-center': true,
                    'w-100': true,
                    'd-none': confirmPassword.untouched || confirmPassword.valid
                }"
              >
                <small *ngIf="confirmPassword.errors?.required">
                  حقل إعادة كلمة السر مطلوب
                </small>
              </div>
              <label
                class="alert alert-danger text-center w-100"
                *ngIf="passwordHasError && confirmPassword.touched"
                >كلمة السر غير متطابقة</label
              >
            </div>

            <div class="form-group">
              <label>Phone</label>
              <input
```

```
146.              type="tel"
147.              #phone="ngModel"
148.              required
149.              pattern="^\d{10}$"
150.              maxlength="10"
151.              [ngClass]="{'form-control': true, 'is-
   invalid': phone.touched && phone.invalid}"
152.              name="phone"
153.              [(ngModel)]="userData.phone"
154.            />
155.            <div
156.              [ngClass]="{
157.                  'alert': true,
158.                  'alert-danger': true,
159.                  'text-center': true,
160.                  'w-100': true,
161.                  'd-none': phone.untouched || phone.valid
162.              }"
163.            >
164.              <small *ngIf="phone.errors?.required">حقل الهاتف مطلوب</small>
165.              <small *ngIf="phone.errors?.pattern">
166.                 خانات 10حقل الهاتف لابد أن يحتوي على
167.              </small>
168.            </div>
169.          </div>
170.
171.          <div class="form-group">
172.            <label>Topics</label>
173.            <select
174.              #topic="ngModel"
175.              (blur)="validateTopic(topic.value)"
176.              (change)="validateTopic(topic.value)"
177.              [ngClass]="{'custom-select': true, 'is-
   invalid': topicHasError && topic.touched}"
178.              name="topics"
179.              [(ngModel)]="userData.topic"
180.            >
181.              <option selected value="default">I am Interested in ..</option>
182.              <option *ngFor="let topic of topics" [ngValue]="topic">
183.                {{topic}}
184.              </option>
185.            </select>
186.            <label
187.              class="alert alert-danger text-center w-100"
188.              *ngIf="topicHasError && topic.touched"
189.              >الحقل مطلوب</label
190.            >
191.          </div>
192.
193.          <div class="mb-3">
194.            <label>Time Preference</label>
195.            <div class="form-check">
196.              <input
```

```
197.                 class="form-check-input"
198.                 type="radio"
199.                 #timePreference="ngModel"
200.                 required
201.                 name="TimePreference"
202.                 [(ngModel)]="userData.timePreference"
203.                 value="Morning"
204.               />
205.               <label class="form-check-label">Morning 9AM - 12PM</label>
206.             </div>
207.             <div class="form-check">
208.               <input
209.                 class="form-check-input"
210.                 type="radio"
211.                 #timePreference="ngModel"
212.                 required
213.                 name="TimePreference"
214.                 [(ngModel)]="userData.timePreference"
215.                 value="Evining"
216.               />
217.               <label class="form-check-label">Evining 5PM - 8PM</label>
218.             </div>
219.           </div>
220.
221.           <div class="form-check mb-3">
222.             <input
223.               class="form-check-input"
224.               type="checkbox"
225.               #subscribePhone="ngModel"
226.               name="subscribePhone"
227.               [(ngModel)]="userData.subscribePhone"
228.             />
229.             <label class="form-check-label">
230.               Send Me Promotional Offers by phone
231.             </label>
232.             <br />
233.             <input
234.               class="form-check-input"
235.               type="checkbox"
236.               #subscribeEmail="ngModel"
237.               name="subscribeEmail"
238.               [(ngModel)]="userData.subscribeEmail"
239.             />
240.             <label class="form-check-label">
241.               Send Me Promotional Offers by Email</label
242.             >
243.           </div>
244.
245.           <div class="card-footer text-muted">
246.             <button
247.               class="btn btn-primary w-100"
248.               [disabled]="
249.                   userForm.form.invalid ||
```

```
                (!subscribePhone.value && !subscribeEmail.value) ||
                nameHasError ||
                passwordHasError ||
                topicHasError
            "
            type="submit"
          >
            Submit Form
          </button>
        </div>
      </form>
    </div>
  </div>
</div>
```

راجع السطر 11

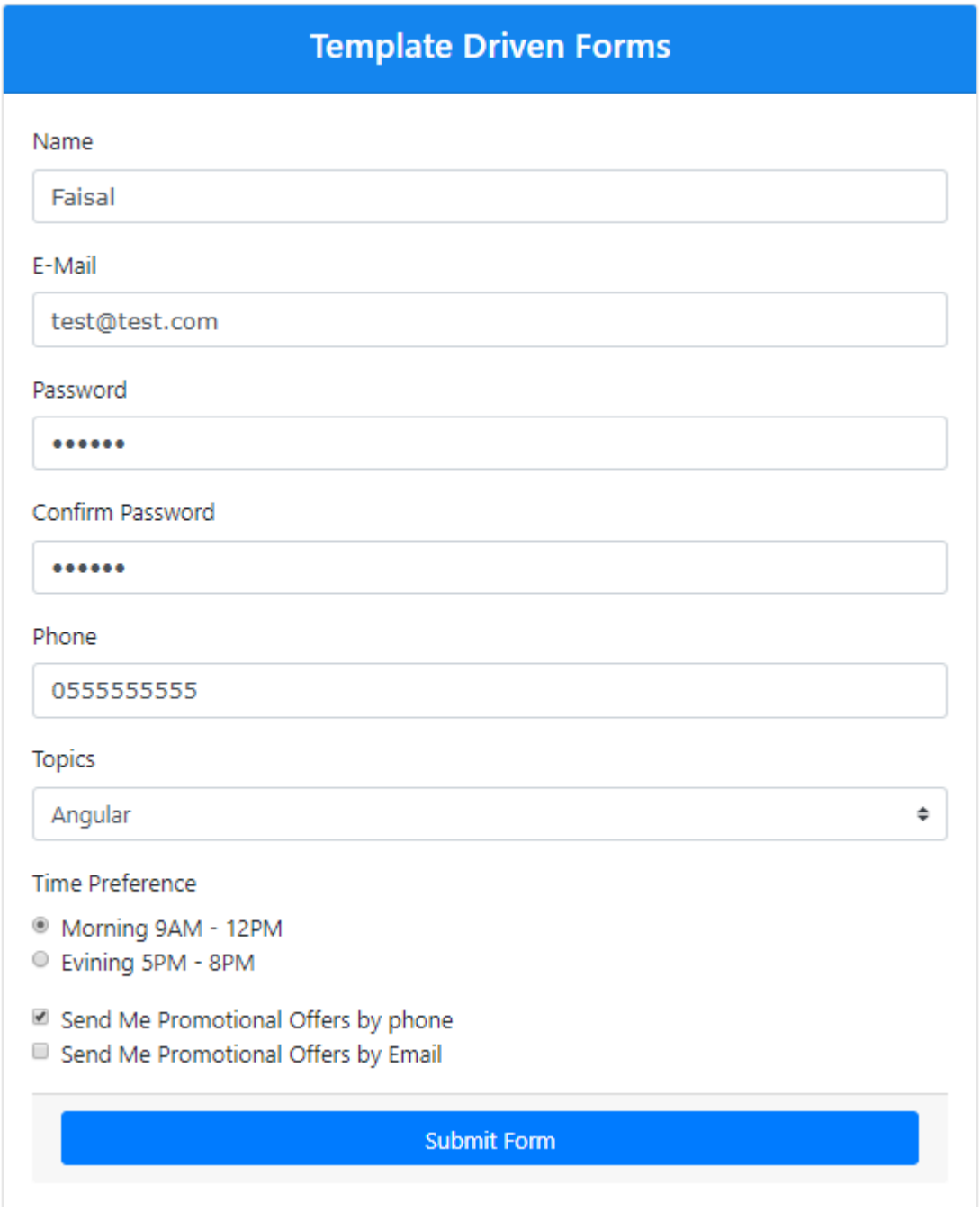
وفي ملف class نكتب محتوى الدالة:

```
import { Component, OnInit } from '@angular/core';
import { User } from '../../user';

@Component({
    selector: 'app-root',
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css']
})

export class AppComponent implements OnInit {
    names: string[] = ['admin', 'administrator'];
    nameHasError: boolean = false;

    passwordHasError: boolean = false;

    topicHasError: boolean = true;

    topics = ['Angular', 'React', 'Vue'];

    userData = new User('', '', null, null, '', '', false, false);

    constructor() { }

    ngOnInit() { }

    validateName(value) {
        this.names.find(val => val === value || (!isNaN(value) && value !== '') ? this.nameHasError = true : this.nameHasError = false);
    }

    validatePaaword(confirmPassVal, passVal) {
```

```
31.         passVal !== confirmPassVal ? this.passwordHasError = true : this.passwordHasErr
    or = false;
32.     }
33.
34.     validateTopic(value) {
35.         value === 'default' || value === '' ? this.topicHasError = true : this.topicHas
    Error = false;
36.     }
37.
38.     onSubmit() {
39.         console.log(this.userData);
40.     }
41.
42. }
```

راجع الاسطر (من 38 إلى 40)

الآن لنقوم بتعبئة النموذج ببيانات صحيحة ومن ثم نقوم بالضغط على زر submit:



الآن نقوم بضغط الزر ثم نفتح أدوات المطور في المتصفح ونذهب إلى التبويب console لنرى النتيجة: (المتصفح الذي استخدمه google chrome):

```
▼User {name: "Faisal", email: "test@test.com", password: "Aa1234", phone: "0555
555555", topic: "Angular", …}
    email: "test@test.com"
    name: "Faisal"
    password: "Aa1234"
    phone: "0555555555"
    subscribeEmail: false
    subscribePhone: true
    timePreference: "Morning"
    topic: "Angular"
  ▶__proto__: Object
```

نلاحظ أن جميع قيم الـForm تم حفظها في الكائن الذي قمنا سابقاً بربط خصائصه بعناصر النموذج، لذلك نستطيع أخذ هذا الكائن وأرساله إلى ثاعدة البيانات او أرسال جزء منه بحسب احتياجنا.

وبذلك نكون أنهينا النوع الأول من أنواع النماذج التي تقدمها لنا Angular Forms وهو TDF، وفي الفصول القادمة بإذن الله سوف نتطرق إلى النوع الثاني وهو Reactive Forms.

الفصل الثالث

مدخل إلى

Angular Reactive Forms

1.3. مقدمة:

في هذا القسم سوف نتكلم عن التقنية الثانية التي يقدمها لنا إطار عمل angular وهي angular reactive forms وهي تقنية أكثر مرونة ومتخصصة في النماذج المعقدة والمتداخلة والديناميكية وتعتمد على logic أكثر من اعتمادها على الدايركتيف التي نضعها في تاغات وعناصر html كما كنا نفعل في النوع الأول TDF، وسوف استفيض في الكلام في هذه التقنية محاول شرح جميع أبعادها وتقنياتها وأضيف بعض الأفكار وأحاول إيجاد حلول لبعض المشاكل البرمجية، أما طريقة الشرح فسوف يكون تقريباً مشابه لما سبق حيث سنبدأ بمثال بسيط ثم كل ما نتعلم تقنية جديدة سوف نضيفها إلى هذا المثال إلى أن يكتمل النموذج بشكله النهائي مغطي بذلك على أغلب مفاهيم هذه التقنية.

2.3. بناء وربط النموذج برمجياً:

بعد إنشاء مشروع angular جديد وإضافة إطار العمل bootstrap إلى المشروع، نقوم ببناء النموذج كالعادة في ملف app.component.html، حيث نضيف اكواد HTML5 التالية:

ملف app.component.html

```
1. <div class="container-fluid">
2.   <div class="card mx-auto col-sm-12 col-md-6 col-lg-6 col-xl-6 pt-sm-4">
3.     <form>
4.       <div class="card-header">
5.         <h4 class="text-center">Reactive Forms</h4>
6.       </div>
7.
8.       <div class="card-body">
9.         <!-- الأداة الخاصة بإدخال اسم المستخدم -->
10.        <div class="form-group">
11.          <label>User Name</label>
12.          <input class="form-control" />
13.        </div>
14.
15.        <!-- الأداة الخاصة بإدخال الإيميل -->
16.        <div class="form-group">
17.          <label>Email</label>
18.          <input type="email" class="form-control" />
19.        </div>
20.
21.        <!-- الأداة الخاصة بإدخال الرقم السري -->
22.        <div class="form-group">
23.          <label>Password</label>
24.          <input type="password" class="form-control" />
25.        </div>
26.
27.        <!-- الأداة الخاصة بإدخال إعادة كتابة الرقم السري -->
28.        <div class="form-group">
29.          <label>Confirm Password</label>
30.          <input type="password" class="form-control" />
31.        </div>
32.
33.        <!-- الأداة الخاصة بإدخال النوع ذكر أو أنثى -->
```

```
34.         <label class="pr-2">Gender</label>
35.         <div class="form-check form-check-inline">
36.           <input
37.             class="form-check-input"
38.             type="radio"
39.             name="gender"
40.             id="maleGender"
41.             value="male"
42.           />
43.           <label class="form-check-label" for="gender">Male</label>
44.         </div>
45.         <div class="form-check form-check-inline">
46.           <input
47.             class="form-check-input"
48.             type="radio"
49.             name="gender"
50.             id="femaleGender"
51.             value="fmail"
52.           />
53.           <label class="form-check-label" for="femaleGender">Femail</label>
54.         </div>
55.
56.         <!-- نموذج فرعي يحتوي على ادوات العنوان -->
57.         <fieldset class="scheduler-border">
58.           <legend class="scheduler-border">Address Informition</legend>
59.           <!-- الأداة الخاصة بإدخال المدينة -->
60.           <div class="form-group">
61.             <label>City</label>
62.             <input class="form-control" />
63.           </div>
64.           <!-- الأداة الخاصة اختيار المنطقة او الولاية -->
65.           <div class="form-group">
66.             <label>State</label>
67.             <select class="form-control">
68.               <option selected [ngValue]="null">Choose...</option>
69.               <option *ngFor="let item of states" [value]="item">
70.                 {{ item }}
71.               </option>
72.             </select>
73.           </div>
74.           <!-- الأداة الخاصة بإدخال الرمز البريدي -->
75.           <div class="form-group">
76.             <label>Zip Code</label>
77.             <input class="form-control" />
78.           </div>
79.         </fieldset>
80.       </div>
81.
82.       <!-- زر الحفظ save -->
83.       <div class="card-footer">
84.         <button class="btn btn-primary">Save</button>
85.       </div>
86.     </form>
```

```
87.   </div>
88. </div>
89.
```

نلاحظ أن اكواد HTML5 هي عبارة عن نموذج يحتوي على مجموعة أدوات بالإضافة إلى نموذج فرعي يحتوي ايضاً بداخله مجموعة أدوات لإدخال بيانات العنوان.

اما في ملف app.component.ts، فنقوم بكتابة الأكواد التالية:

ملف app.component.ts

```
1. import { Component, OnInit } from "@angular/core";
2.
3. @Component({
4.     selector: "app-root",
5.     templateUrl: "./app.component.html",
6.     styleUrls: ["./app.component.css"]
7. })
8. export class AppComponent implements OnInit {
9.   private states: string[] = ["AR", "AL", "CA", "DC"];
10.
11.    constructor() {
12.
13.    }
14.
15.    ngOnInit() {
16.
17.    }
18.
19.}
20.
```

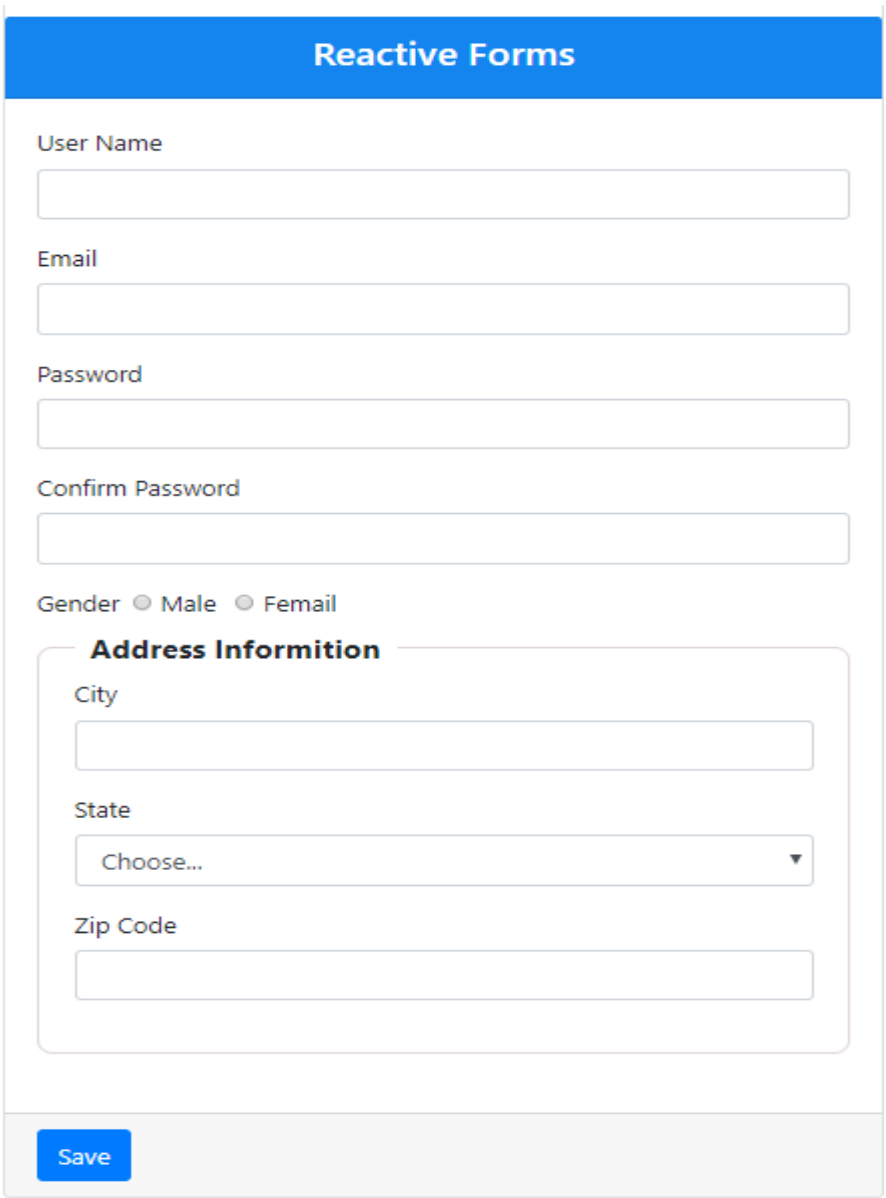
اما ملف app.component.css، نضيف الأكواد التالية:

ملف app.component.css

```
1. .card {
2.   margin-top: 50px;
3.   padding-top: 20px;
4.   padding-right: 0px;
5.   padding-left: 0px;
6. }
7.
8. .card-header {
9.   background-color: rgb(20, 133, 238);
10.  color: white;
11.}
12.
13.input {
14.  font-family: FontAwesome, "Open Sans", Verdana, sans-serif;
15.}
16.
17.fieldset.scheduler-border {
```

```
18.   border: 1px solid rgba(218, 205, 205, 0.815) !important;
19.   border-radius: 10px;
20.   padding: 0 1.4em 1.4em 1.4em !important;
21.   margin: 0 0 1.5em 0 !important;
22.   -webkit-box-shadow: 0px 0px 0px 0px #000;
23.   box-shadow: 0px 0px 0px 0px #000;
24. }
25.
26. legend.scheduler-border {
27.   font-size: 1.2em !important;
28.   font-weight: bold !important;
29.   text-align: left !important;
30.   width: auto;
31.   padding: 0 10px;
32.   border-bottom: none;
33. }
34.
```

الآن لنقوم بتشغيل المشروع وذلك عن طريق كتابة الأمر ng serve –o في terminal لمشاهدة النتيجة، كالتالي:



إلا الآن لم نقم بالتعامل مع reactive forms فكل الأكواد السابقة هي عبارة عن اكواد HTML5 وCSS3 وبعض الأكواد البسيطة من angular، لذلك للتعامل مع reactive forms ينبغي أولاً إضافة المديول ReactiveFormsModule في الملف app.module.ts حيث أن هذا المديول يعطينا عدد كبير من الكلاسات والدايركتيف والسيرفيسس التي تسهل لنا التعامل مع reactive forms، كالتالي:

ملف app.module.ts

```
1. import { BrowserModule } from '@angular/platform-browser';
2. import { NgModule } from '@angular/core';
3. import { ReactiveFormsModule } from '@angular/forms';
4. import { AppComponent } from './app.component';
5.
6. @NgModule({
7.     declarations: [
8.         AppComponent
9.     ],
10.    imports: [
11.        BrowserModule,
12.        NgModule
13.        ReactiveFormsModule
14.    ],
15.    providers: [],
16.    bootstrap: [AppComponent]
17.})
18.
19.export class AppModule { }
20.
```

راجع السطر 3 والسطر 13

بعد بناءنا للنموذج عن طريق ملف app.component.html وبعد إضافتنا أيضاً للمديول ReactiveFormsModule نقوم ببناء هذا النموذج برمجياً في ملف app.component.ts، حيث لو رجعنا إلى النموذج لوجدنا أنه يحتوي على أربع أدوات أداة لإدخال اسم المستخدم وأداة أخرى لإدخال الإيميل وأداتين لكلمة السر وإعادة كتابة كلمة السر، بالإضافة إلى نموذج فرعي يحتوي على ثلاث أدوات لإدخال بيانات العنوان، لذلك لبناء النموذج برمجياً نقوم بالخطوات التالية في ملف app.component.ts، كالتالي:

١) تعريف متغير وليكن اسمه form ونوعه الكلاس FormGroup بحيث يشير هذا المتغير إلى النموذج الخاص بنا.

٢) نستدعي service ذات الاسم FormBuilder ونعمل لها injection في constructor حيث نستخدمها لبناء النموذج.

٣) نقوم ببناء النموذج برمجياً حيث يكون على شكل كائن object يحتوي على مجموعة خصائص وكل خاصية تقابل أداة من أدوات النموذج وتشير إليها، ولك حرية اختيار أسماء هذه الخصائص ولكن يُفضل أن تكون معبرة عن الأداة التي تشير إليها، مع العلم ان كتابة الأكواد الخاصة ببناء النموذج تتم في الدالة ngOnInit التي تقدمها لنا angular.

ملف app.component.ts

```
1. import { Component, OnInit } from '@angular/core';
2.
3. import { FormGroup, FormBuilder } from '@angular/forms';
4.
5. @Component({
6.     selector: 'app-root',
7.     templateUrl: './app.component.html',
8.     styleUrls: ['./app.component.css']
9. })
10.
11.export class AppComponent implements OnInit {
12.
13.    states: String[] = ['AR', 'AL', 'CA', 'DC'];
14.
15.    form: FormGroup;
16.
17.    constructor(private fb: FormBuilder) { }
18.
19.    ngOnInit() {
20.        this.form = this.fb.group({
21.            userName: [],
22.            email: [],
23.            password: [],
24.            confirmPassword: [],
25.            address: this.fb.group({
26.                city: [],
27.                state: [],
28.                zipCode: []
29.            })
30.        });
31.    }
32.
33.}
34.
```

راجع السطر 1 (استدعاء الinterface ذات الاسم ngOnInit التي تحتوي على الدالة ngOnInit)

راجع السطر 3 (استدعاء الكلاس FormGroup والservice ذات الاسم FormBuilder)

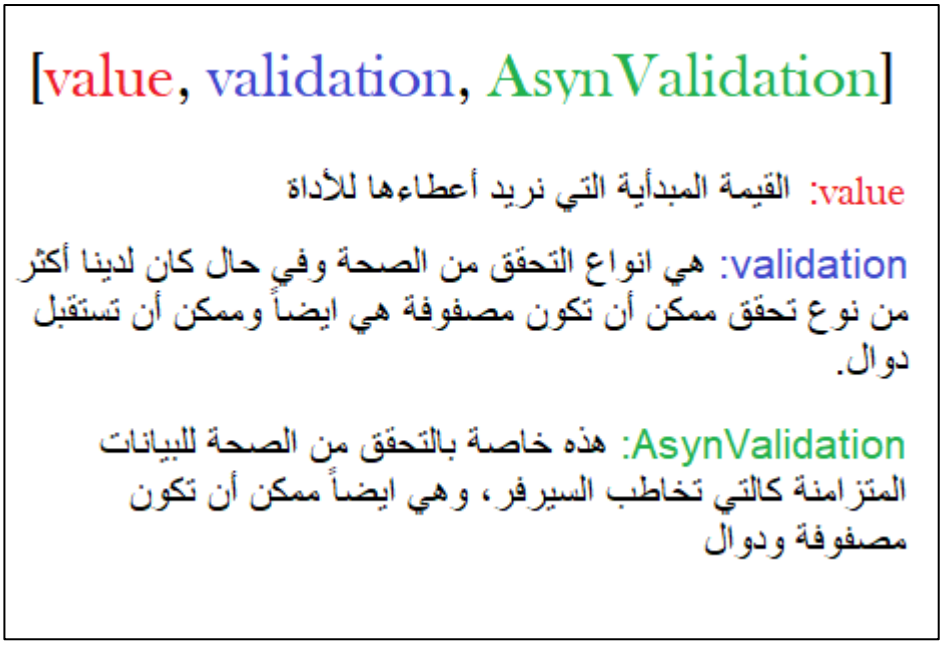
راجع السطر 11 (نعمل implement لـOnInit interface)

راجع السطر 15 (تعريف متغير باسم form)

راجع السطر 17 (نعمل injection او ما يسمى حقن لـservice في constructor حيث تم حقنها في متغير وليكن اسمه fb)

راجع الاسطر من 19 إلى 31 (أكواد بناء النموذج برمجياً)

نلاحظ أن القيمة لكل خاصية هي اقواس المصفوفة حيث أن هذه الأقواس تتكون من ثلاث أجزاء، كالتالي:



ما يهمنا الآن هو الجزء الأول value حيث سوف نقوم بإعطاء قيمة مبدأية او افتراضية null عند بداية تشغيل النموذج، كالتالي:

```
1.    ngOnInit() {
2.      this.form = this.fb.group({
3.        userName: [null],
4.        email: [null],
5.        password: [null],
6.        confirmPassword: [null],
7.        gendar: [null],
8.        address: this.fb.group({
9.          city: [null],
10.         state: [null],
11.         zipCode: [null]
12.       })
13.     });
14.   }
```

الآن لنقوم بربط النموذج البرمجي بالنموذج الأساسي، بحيث نستخدم الدايكرتيف Directive [FormGroup] لربط النموذج الأساسي بالمتغير form الذي انشأناه بحيث يمثل كامل النموذج، اما كل خاصية نربطها بالأداة المكافئة لها عن طريق الدايكرتيف formControlName، أما إذا كان لدينا نموذج فرعي فنقوم بإضافة الدايكرتيف formGroupName لـ<tag> الذي يحوي أدوات النموذج الفرعي ونربطه في الخاصية للنموذج الفرعي الذي تم بنائه برمجياً وفي مثالنا هنا لايوجد إلا نموذج فرعي واحد وأنشأنا له خاصية تقابله واسميناها address، أما باقي أدوات النموذج الفرعي فتعامل بشكل عادي عن طريق الدايركتيف formControlName، كالتالي:

ملف app.component.html:

app.component.html ملف

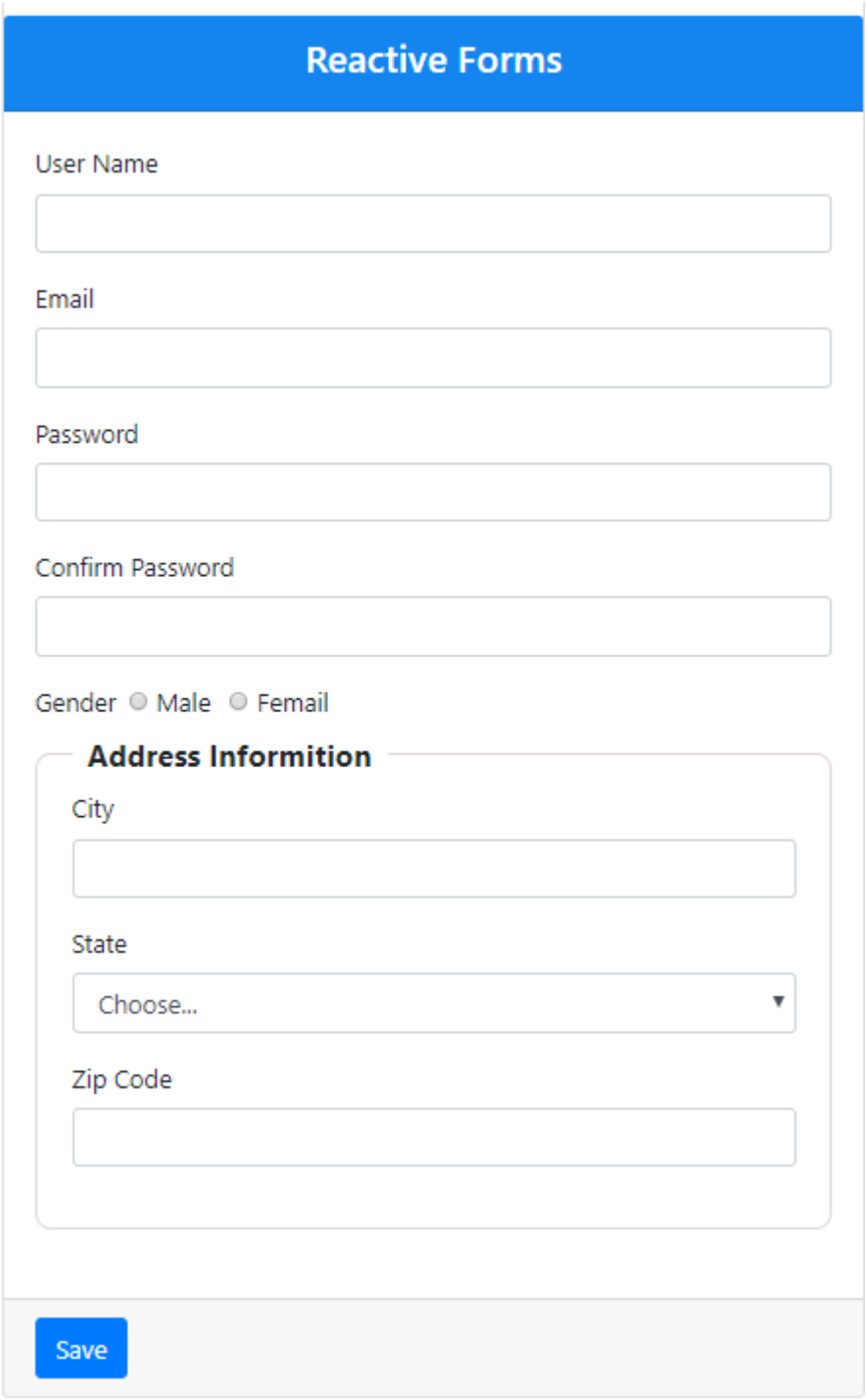
```
1. <div class="container-fluid">
2.   <div class="card mx-auto col-sm-12 col-md-6 col-lg-6 col-xl-6 pt-sm-4">
3.     <form [formGroup]="form">
4.       <div class="card-header">
5.         <h4 class="text-center">Reactive Forms</h4>
```

```
6.        </div>
7.
8.        <div class="card-body">
9.          <!-- الأداة الخاصة بإدخال اسم المستخدم -->
10.         <div class="form-group">
11.           <label>User Name</label>
12.           <input class="form-control" formControlName="userName" />
13.         </div>
14.
15.         <!-- الأداة الخاصة بإدخال الإيميل -->
16.         <div class="form-group">
17.           <label>Email</label>
18.           <input type="email" class="form-control" formControlName="email" />
19.         </div>
20.
21.         <!-- الأداة الخاصة بإدخال الرقم السري -->
22.         <div class="form-group">
23.           <label>Password</label>
24.           <input
25.             type="password"
26.             class="form-control"
27.             formControlName="password"
28.           />
29.         </div>
30.
31.         <!-- الأداة الخاصة بإدخال إعادة كتابة الرقم السري -->
32.         <div class="form-group">
33.           <label>Confirm Password</label>
34.           <input
35.             type="password"
36.             class="form-control"
37.             formControlName="confirmPassword"
38.           />
39.         </div>
40.
41.         <!-- الأداة الخاصة بإدخال النوع ذكر أو أنثى -->
42.         <label class="pr-2">Gender</label>
43.         <div class="form-check form-check-inline">
44.           <input
45.             class="form-check-input"
46.             type="radio"
47.             name="gender"
48.             id="maleGender"
49.             value="male"
50.             formControlName="gender"
51.           />
52.           <label class="form-check-label" for="gender">Male</label>
53.         </div>
54.         <div class="form-check form-check-inline">
55.           <input
56.             class="form-check-input"
57.             type="radio"
58.             name="gender"
```

```
59.             id="femaleGender"
60.             value="femail"
61.             formControlName="gender"
62.           />
63.           <label class="form-check-label" for="femaleGender">Femail</label>
64.         </div>
65.
66.         <!-- نموذج فرعي يحتوي على ادوات العنوان -->
67.         <fieldset class="scheduler-border" formGroupName="address">
68.           <legend class="scheduler-border">Address Information</legend>
69.           <!-- الأداة الخاصة بإدخال المدينة -->
70.           <div class="form-group">
71.             <label>City</label>
72.             <input class="form-control" formControlName="city" />
73.           </div>
74.           <!-- الأداة الخاصة اختيار المنطقة او الولاية -->
75.           <div class="form-group">
76.             <label>State</label>
77.             <select class="form-control" formControlName="state">
78.               <option selected [ngValue]="null">Choose...</option>
79.               <option *ngFor="let item of states" [value]="item">
80.                 {{ item }}
81.               </option>
82.             </select>
83.           </div>
84.           <!-- الأداة الخاصة بإدخال الرمز البريدي -->
85.           <div class="form-group">
86.             <label>Zip Code</label>
87.             <input class="form-control" formControlName="zipCode" />
88.           </div>
89.         </fieldset>
90.       </div>
91.
92.       <!-- زر الحفظ save -->
93.       <div class="card-footer">
94.         <button class="btn btn-primary">Save</button>
95.       </div>
96.     </form>
97.   </div>
98. </div>
99.
```

راجع الاسطر (3 – 12 – 18 – 27 – 37 – 50 – 61 – 67 – 72 – 77 - 87)

الآن لنقوم بتشغيل النموذج ونرى ما قمنا به من تعديلات:



نلاحظ أن النموذج لم يختلف عن بداية بنائنا له في صفحة app.component.html السبب في ذلك أننا اعطينا القيمة الافتراضية للخصائص null، وكما نستطيع تعبئة النموذج برمجياً وهذا ما سوف نتكلم عنه في الجزء التالي.

3.3. تعبأة النموذج برمجياً (setValue – patchValue):

نستطيع تعبأة النموذج برمجياً كأن يكون لدينا بيانات قادمة من قاعدة البيانات ونريد عرضها للمستخدم لكي يقوم بالتعديل عليها ثم يحفظها لكي تُرسل لقاعدة البيانات مرة أخرى، ونستطيع القيام بذلك عن طريق الدالتين setValue و patchValue وهما دالتين من ضمن الكلاس FormGroup وبما أننا عرفنا المتغير form على انه نوع من هذا الكلاس لذلك اصبح يمتلك حق الوصول إلى هذه الدالتين، أما مهمة هاتين الدالتين لتعبئة مجموعة بيانات للنموذج عن طريق اسناد هذه القيم إلى الخصائص التي قمنا ببنائها برمجياً وربطناها مع حقول النموذج سواء كانت هذه القيم ديناميكية بمعنى قادمة

من قاعدة بيانات او ثابتة نحن قمنا بتعبأتها يدوياً، أما من ناحية الفرق بينهما فالدالة setValue تُلزم اسناد قيم لجميع الخصائص ولو كانت هذه القيم فارغة اما الدالة patchValue فتسمح اسناد قيم لبعض الخصائص او اسناد قيم لجميع هذه الخصائص، وبما أننا ليس لدينا قاعدة بيانات فسوف نسند القيم يدوياً كما انني سوف استخدم الدالة patchValue، وللقيام بذلك سوف اضيف زر في ملف app.component.html وليكن هذا الزر اسمه Laod Data ويحمل دالة بداخله تحمل نفس الاسم loadData() ولا تستقبل أي باراميتر ومهمة هذه الدالة تعبئة بعض البيانات للنموذج، وسوف تكون بجانب زر save، أما محتوى الدالة التي يحتويها الزر فتكون بداخل ملف app.componet.ts، كالتالي:

جزء من ملف app.component.html

```
1. <div class="card-footer">
2.   <button class="btn btn-primary">Save</button>
3.   <button class="btn btn-primary ml-3" (click)="laodData()">Load Data</button>
4. </div>
5.
```

راجع السطر 3

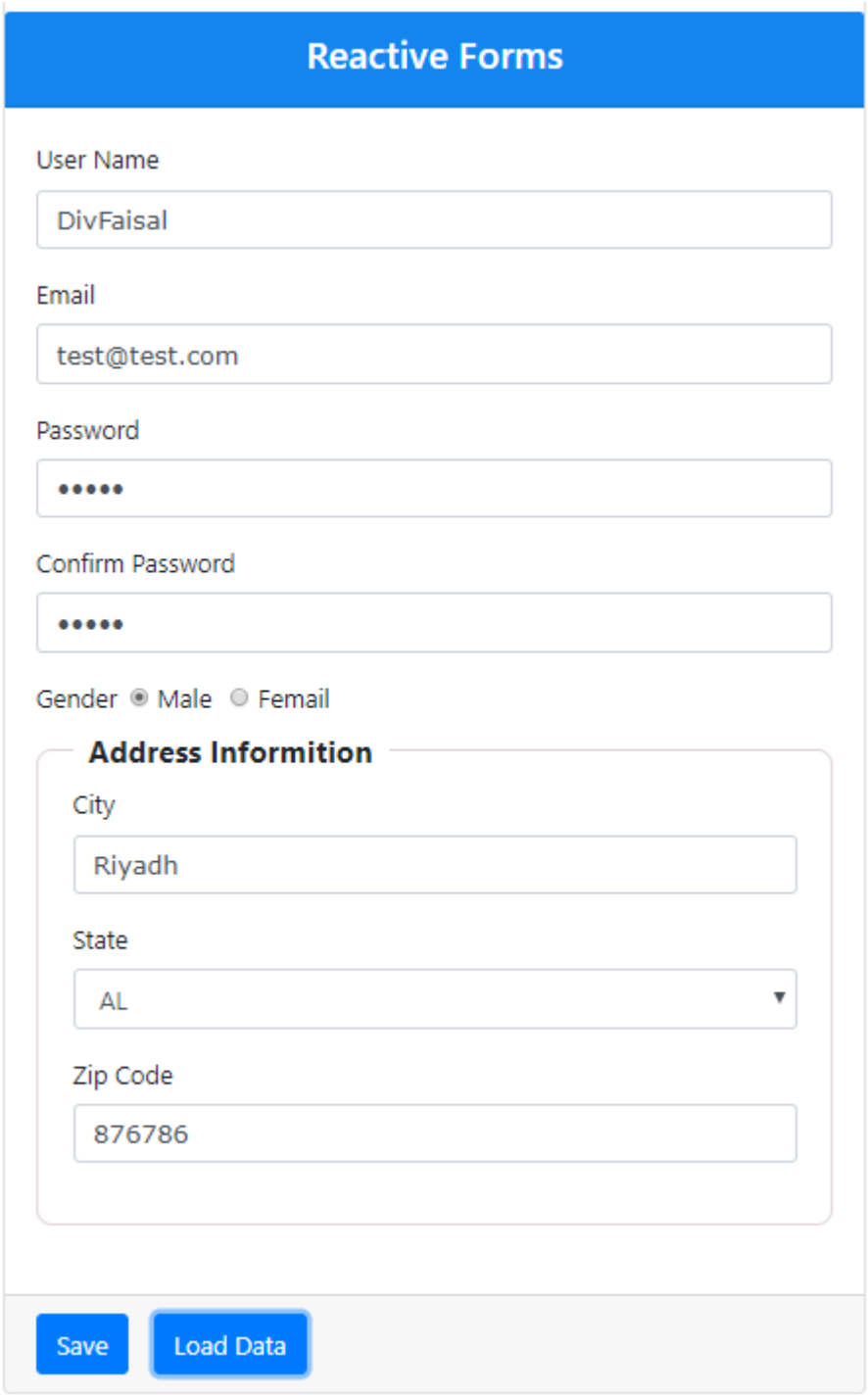
ملف app.component.ts:

جزء من ملف app.component.ts

```
1. laodData() {
2.     this.form.patchValue({
3.         userName: 'DivFaisal',
4.         email: 'test@test.com',
5.         password: '12345',
6.         confirmPassword: '12345',
7.         gender: 'male',
8.         address: {
9.             city: 'Riyadh',
10.            state: 'AL',
11.            zipCode: '876786'
12.        }
13.    });
14.}
15.
```

محتوى الدالة، laodData حيث استخدمنا الدالة patchValue لتمرير بيانات إلى الخصائص.

الآن لنقم بتشغيل النموذج ومن ثم الضغط على الزر Load Data ولنرى النتيجة:



نشاهد أن البيانات تم تعبأتها إلى النموذج.

4.3. مراقبة التعديل على قيم النموذج برمجياً valueChanges:

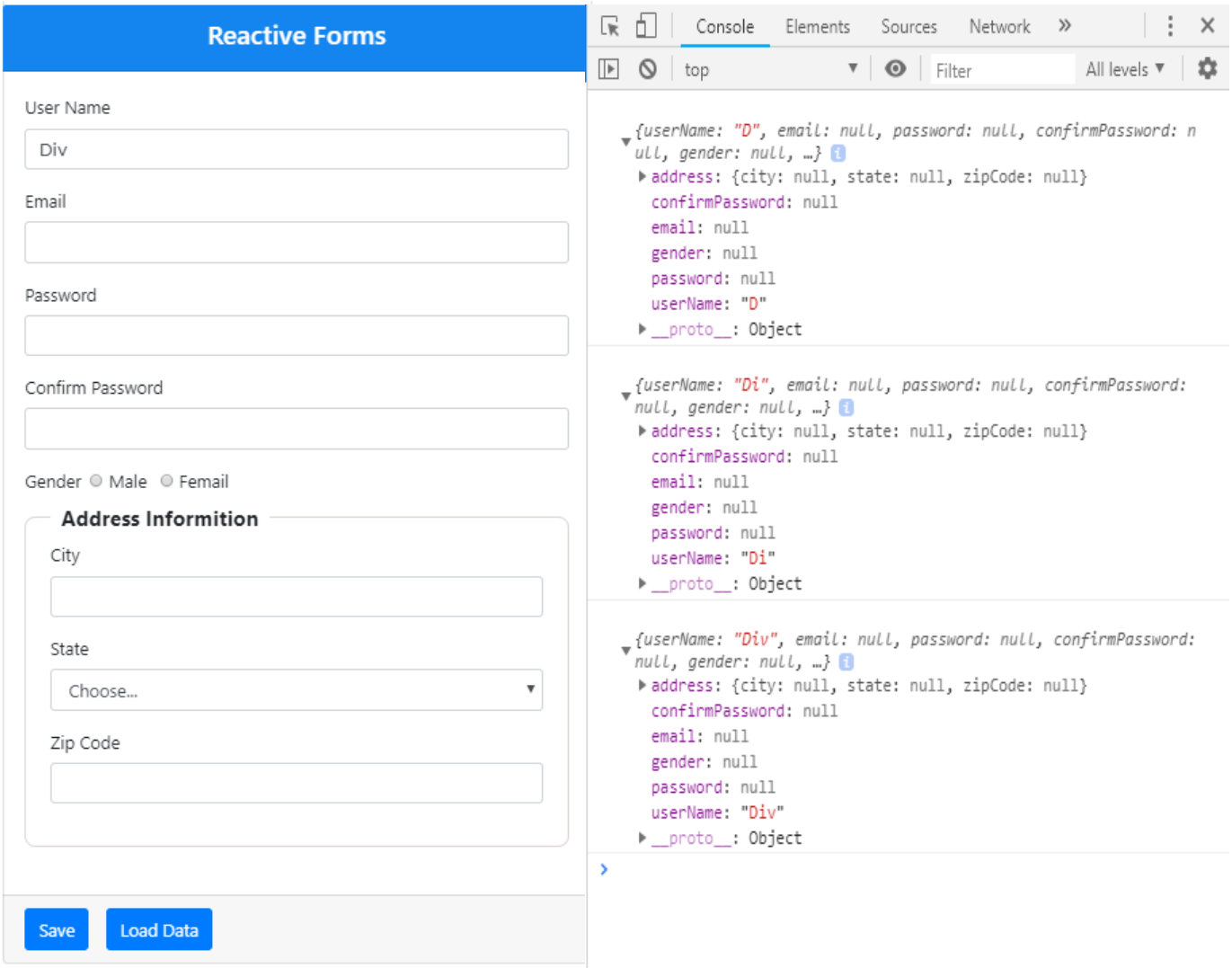
من المميزات الرائعة التي قدمتها لنا angular reactive forms هي الدالة valueChanges حيث أن النموذج او النماذج الفرعية أو حتى الأدوات جميعها تمتلك حق الوصول إلى هذه الدالة، وهذه الدالة تُرجع لنا قيمة من النوع Observable لذلك لابد أن نعمل لها subscribe، فإذا كانت على مستوى النموذج كامل فإنها تُرجع لنا قيم أدوات النموذج كاملاً على شكل كائن object مع أي تغيير يطرأ على أي قيمة من قيم أي أداة، أما إذا كانت على مستوى النموذج الفرعي فهي ايضاً تُرجع لنا كائن لقيم أدوات النموذج الفرعي، أما إذا كانت على مستوى أداة فهي تُرجع لنا قيمة هذه الأداة فقط عند أي تغيير يطرأ على قيمتها، وهكذا بقية أجزاء النموذج.

وسوف أعطي مثالاً لكل حالة من هذه الحالات الثلاثة، مع العلم أنني سوف اكتب الكود في الدالة ngOnInit الي تكلمنا عنها سابقاً، كالتالي:

على مستوى النموذج كاملا، نكتب الكود التالي:

```
this.form.valueChanges.subscribe(value => {
  console.log(value);
});
```

ولنرى النتيجة اولاً نقوم بتشغيل المشروع على المتصفح وثانياً لابد أن نفتح أدوات المطور في المتصفح – استخدم متصفح chrome – ونذهب على التبويب console، بحيث تكون النتيجة كالتالي:



نلاحظ أنه في البداية قمنا بكتابة حرف D فأرجع لنا كائن يحتوي على جميع قيم النموذج وعند كتابتنا للحرف الثاني i أيضاً قام بإرجاع الكائن السابق مع التعديل الجديد وهكذا، لذلك هو يقوم بمراقبة جميع أدوات النموذج وفي حال التعديل على أي قيمة من قيم هذه الأدوات سواء إضافة او حذف يقوم بإرجاع جميع قيم هذا النموذج.

اما على مستوى النموذج الفرعي – مع العلم اننا قمنا بتسمية النموذج الفرعي عند انشاءه برمجياً باسم address -، فيكون الكود بالشكل التالي:

```
this.form.get('address').valueChanges.subscribe(value => {
  console.log(value);
});
```

اما النتيجة فهي مشابهة لما سبق ولكن هنا تُرجع كائن لقيم أدوات النموذج الفرعي فقط.

اما على مستوى الأداة – لنأخذ أداة username كمثال وباقي الأدوات نفس الطريقة – فيكون الكود كالتالي:

```
this.form.get('userName').valueChanges.subscribe(value => {
  console.log(value);
});
```

اما النتيجة فهي مشابهة لما سبق ولكن هنا تقوم بإرجاع قيمة الأداة فقط في كل مرة يتم التعديل عليها.

أما للوصول إلى أداة داخل نموذج فرعي – ولنأخذ الأداة city كمثال – فنكتب الكود التالي:

```
this.form.get('address').get('city').valueChanges.subscribe(value => {
  console.log(value);
});
```

وهذه الميزة valueChanges لها فوائد كثيرة وسوف نستخدمها عند التطرق إلى التحقق من الصحة المشروط لاحقاً بإذن الله، ولكن هنا سوف نوضح فكرة بسيطة وهي وضع عداد يحسب عدد الحروف المدخلة في حقل userName، وللقيام بهذا الأمر نقوم اولاً في ملف app.component.ts بتعرف متغير وليكن اسمه userNameLength ونعطيه قيمة مبدأيه تساوي الصفر بحيث يكون هو العداد الذي يخزن عدد الحروف، ومن ثم نكتب الكود التالي:

ملف app.component.ts

```
import { Component, OnInit } from '@angular/core';

import { FormGroup, FormBuilder } from '@angular/forms';

@Component({
    selector: 'app-root,
    templateUrl: './ app.component.html',
    styleUrls: ['./ app.component.css']
})

export class AppComponent implements OnInit {

    private states: string[] = ['AR', 'AL', 'CA', 'DC'];

    form: FormGroup;

    userNameLength = '0';

    constructor(private fb: FormBuilder) { }

```

```
21.     ngOnInit() {
22.         this.form = this.fb.group({
23.             userName: [null],
24.             email: [null],
25.             password: [null],
26.             confirmPassword: [null],
27.             gender: [null],
28.             address: this.fb.group({
29.                 city: [null],
30.                 state: [null],
31.                 zipCode: [null]
32.             })
33.         });
34.
35.         this.form.get('userName').valueChanges.subscribe((value: string) => {
36.             this.userNameLength = value.length;
37.         });
38.
39.     }
40.
41.     laodData() {
42.         this.form.patchValue({
43.             userName: 'DivFaisal',
44.             email: 'test@test.com',
45.             password: '12345',
46.             confirmPassword: '12345',
47.             gender: 'male',
48.             address: {
49.                 city: 'Riyadh',
50.                 state: 'AL',
51.                 zipCode: '876786'
52.             }
53.         });
54.     }
55.
56. }
57.
```

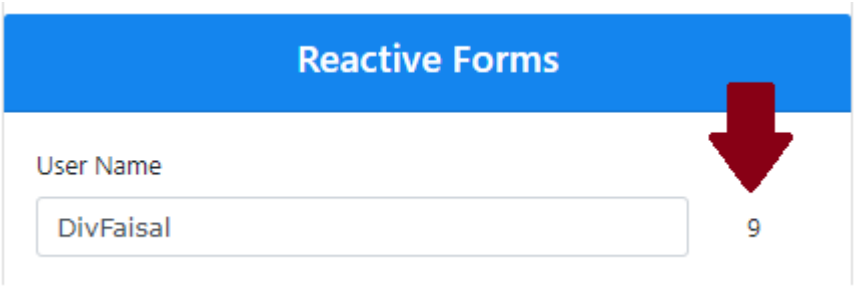
راجع السطر 17 – الأسطر من 35 إلى 37

أما في ملف app.component.html وفي الجزء الخاص بكود أداة إدخال اسم المستخدم نقوم بالتعديلات التالية:

جزء من ملف app.component.html

```
1. <!-- الأداة الخاصة بإدخال اسم المستخدم -->
2. <div class="form-group">
3.   <label>User Name</label>
4.   <div class="form-inline">
5.     <input class="form-control col-10" formControlName="userName" />
6.     <label class="col-2">{{ userNameLength }}</label>
7.   </div>
8. </div>
9.
```

أما عند تشغيل النموذج على المتصفح فسوف يكون شكل الأداة، كالتالي:



ملاحظة: valueChanges تراقب جميع التعديلات سواء كانت من خلال النموذج أو برمجياً.

5.3. إنشاء مرجع لأسماء الأدوات:

في القسم السابق نلاحظ اننا لو أردنا الوصول إلى أحد الأدوات داخل النموذج برمجياً يجب كتابة اسم المتغير الذي يشير الى النموذج ومن ثم الدالة get ولو كانت هذه الأداة داخل نموذج فرعي لابد ان نصل إلى النموذج الفرعي ثم الأداة، وهذه الطريقة صحيحة ولكن للاختصار يُفضل إنشاء دوال تشير إلى مسار هذه الأدوات بدلاً من كتابة المسار في كل مرة، ونستطيع القيام بذلك عن طريق الدالة get التي تقدمها لنا Typescript ومن ثم كتابة اسم دالة ويُضل ان تكون نفس الاسم البرمجي للأداة، ومهمة هذه الدالة ارجاع مسار الوصول إلى هذه الأداة، فمثلاً لو اردنا الوصول إلى الأداة userName للحصول على قيمتها نكتب: this.form.get("userName").value لذلك سوف ننشأ دالة get لهذه الأداة بالطريقة التالية:

```
1.  get userName() {
2.    return this.form.get("userName");
3.  }
```

وبالتالي نستطيع كتابة اسم هذه الأداة بدلاً من كتابة مسار الوصول إلى الأداة في كل مرة.

مع ملاحظة أن اسم الدالة لك حرية اختيار الاسم الذي تُريده ولكن يُفضل أن يكون مشابه للاسم البرمجي للأداة.

الآن في ملف app.component.ts لنقم بكتابة الدالة get لكل الأدوات، كالتالي:

جزء من ملف app.component.ts

```
1. get userName() {
2.     return this.form.get('userName');
3. }
4. get email() {
5.     return this.form.get('email');
6. }
7. get password() {
8.     return this.form.get('password');
9. }
10.get confirmPassword() {
11.     return this.form.get('confirmPassword');
12.}
13.get gender() {
14.     return this.form.get('gender');
15.}
16.get address() {
```

```
17.     return this.form.get('address');
18. }
19. get city() {
20.     return this.form.get('address').get('city');
21. }
22. get state() {
23.     return this.form.get('address').get('state');
24. }
25. get zipCode() {
26.     return this.form.get('address').get('zipCode');
27. }
28.
```

6.3. عمل Loop على أدوات النموذج برمجياً:

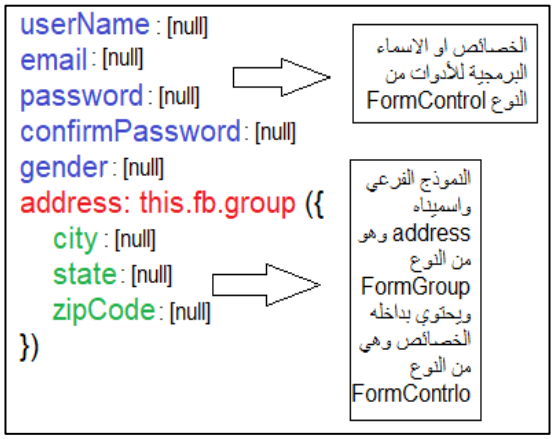
بما أننا نتعامل مع النموذج برمجياً نستطيع عمل logic أو طريقة برمجية لعمل Loop على أدوات النموذج والحصول على الخصائص التي تشير إلى أدوات النموذج مع القيم الخاصة بها، ويمكن الاستفادة من هذه الطريقة بأمور عديدة منها على سبيل المثال التأكد أن جميع القيم داخل الأدوات صحيحة عند الضغط على زر حفظ (ولكن نؤجل الكلام إلى حين وصولنا إلى الجزء الخاص بالتحقق من الصحة)، الآن لنقوم بإنشاء دالة مهمة هذه الدالة تقوم بعمل Loop على جميع أدوات النموذج والحصول على الاسم البرمجي – الخاصية – لكل أداة مع القيمة الخاصة بها، وليكن أسم الدالة loopThroughControls وهذه الدالة تستقبل باراميتر من النوع FormGroup الذي هو بالحقيقة النموذج الخاص بنا والذي إنشانا له الاسم البرمجي form، ويكون تنفيذ هذه الدالة في الزر save، كالتالي:

جزء من ملف app.component.ts

```
1. save() {
2.     this.loopThroughControls(this.form);
3. }
4.
5. loopThroughControls(formGroup: FormGroup) {
6.     Object.keys(formGroup.controls).forEach((key: string) => {
7.         const contrloName = formGroup.get(key);
8.         if (contrloName instanceof FormGroup) {
9.             Object.keys(contrloName.controls).forEach((subKey: string) => {
10.                 console.log(`${subKey} - ${contrloName.get(subKey).value}`);
11.             });
12.         } else {
13.             console.log(`${key} - ${abstractCTRL.value}`);
14.         }
15.     });
16. }
17.
```

ولفهم هذه الكود يجب أولاً فهم النموذج الذي بنيناه برمجياً، فالنموذج هو عبارة عن خصائص برمجية تشير إلى أدوات النموذج وهذه الخصائص من النوع FormControl وهو كلاس من الكلاسات التي تقدمها angular reactive forms وهي تقدم مجموعة من الدوال والخصائص التي تساعد في التعامل مع هذه الأدوات كالحصول على قيمها والتأكد من صحتها وغيرها الكثير، والجزء الثاني من النموذج هو النموذج الفرعي باسم address ويحتوي داخله مجموعة من الأدوات وهو من

النوع FormGroup وهو ايضاً كلاس يتعامل مع النموذج الرئيسي او النماذج الفرعية، والنموذج البرمجي هو بالحقيقة عبارة عن كائن object ومعروف ان الكائن يتكون من خاصية او مفتاح key من النوع نص string وقيمة value، لذلك سوف نقوم بعمل loop مرتين الأولى لأدوات كامل النموذج والثاني لأدوات النموذج الفرعي ونقوم بالحصول على key الذي يمثل الخاصية والقيمة لكل key التي تمثل قيمة الموجودة في الأداة.



الآن لنقوم بشرح الأكواد السابقة:

السطر من 1 إلى 3 (الدالة save وهي الدالة التي يتم تنفيذها عند الضغط على زر save، ومهمتها فقط استدعاء الدالة loopThroughControls وممرنا لها المتغير form الذي يُمثل النموذج الخاص بنا)

السطر 5 (بداية الدالة loopThroughControls وتستقبل باراميتر من النوع FormGroup واسميناه formGroup وهو في الحقيقة النموذج form الذي مررناه لدالة في السطر السابقة الذكر)

السطر 6 (كما قلنا سابقاً ان النموذج برمجياً هو في الحقيقة كائن object يحتوي على key وهو يمثل الخصائص أو الأسماء البرمجية للأدوات وvalue وهو يمثل قيم هذه الخصائص، لذلك نقوم بالحصول على key من هذا object من خلال تمرير جميع الأدوات controls الموجودة في النموذج formGroup، ومن ثم عن طريق الدالة forEach نقوم بعمل loop على هذا الكائن، وهي تستقبل باراميتر وهو عبارة key في كل مرة تقوم فيها بعمل تكرار أو loop على الكائن)

السطر 7 (وهو اول سطر في حلقة التكرار loop وهنا نقوم بتخزين الخاصية في ثابت ويمكن الوصول إلى الخاصية أو الاسم البرمجي للأداة من خلال تمرير key للأمر formGroup.get(key) حيث لو كانت قيمة key تساوي username فإن الثابت سوف يمثل الاسم البرمجي او الخاصية username في النموذج وهكذا إلى الانتهاء من جميع keys في الكائن)

السطر من 8 الى 11 (في هذا السطر من القيمة الموجودة في الثايت controlName هل هي من النوع FormGroup أو لا فإذا كانت من نفس النوع فإنها بهذه الحالة تشير إلى النموذج الفرعي address وبذلك نقوم بعمل loop أخرى للحصول على الأدوات الموجودة بداخله مع عرض اسم الأداة وقيمتها في console الخاص بالمتصفح)

السطر من 12 إلى 14 (هنا أن لم يكن نوع الثابت FormGroup فهو إذن FormControl لذلك يقوم بطباعة اسم الأداة وقيمتها)

ولو نلاحظ أننا قمنا بعمل Loop مرتين لذلك نستطيع اختصار هذا الكود بالطريقة التالية:

```
save() {
  this.loopThroughControls(this.form);
}

loopThroughControls(formGroup: FormGroup) {
  Object.keys(formGroup.controls).forEach((key: string) => {
    const controlName = formGroup.get(key);
    if (controlName instanceof FormGroup) {
      this.loopThroughControls(controlName);
    } else {
      console.log(`${key} - ${controlName.value}`);
    }
  });
}
```

الآن لنضيف هذا الكود إلى ملف app.component.ts:

ملف app.component.ts

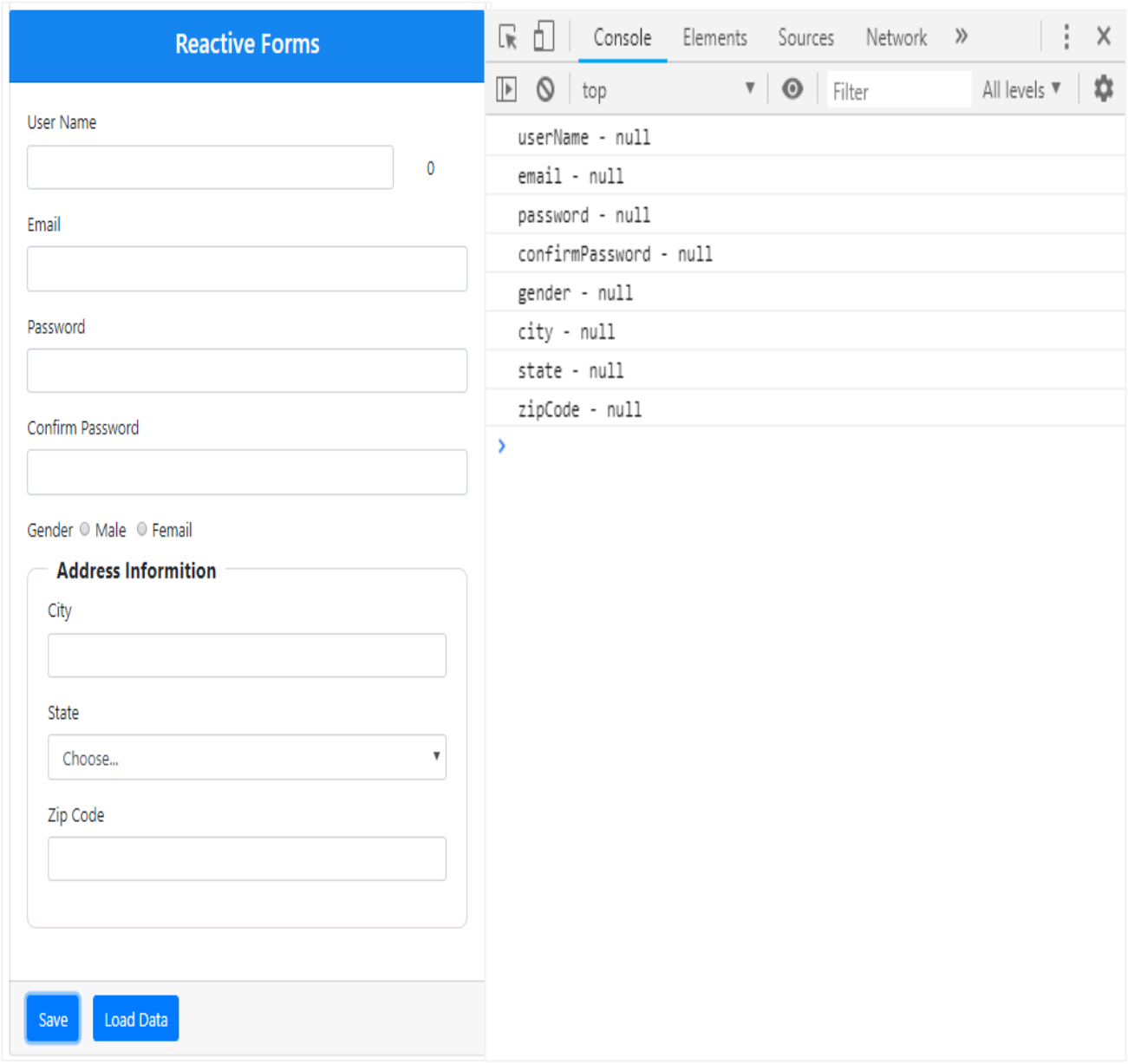
```
import { Component, OnInit } from "@angular/core";

import { FormGroup, FormBuilder } from "@angular/forms";

@Component({
    selector: "app-root",
    templateUrl: "./app.component.html",
    styleUrls: ["./app.component.css"]
})

export class AppComponent implements OnInit {
    private states: string[] = ["AR", "AL", "CA", "DC"];
    form: FormGroup;
    userNameLength: any = "0";

    constructor(private fb: FormBuilder) { }

    ngOnInit() {
        this.form = this.fb.group({
            userName: [null],
            email: [null],
            password: [null],
            confirmPassword: [null],
            gender: [null],
            address: this.fb.group({
                city: [null],
                state: [null],
                zipCode: [null]
            })
        });

        this.userName.valueChanges.subscribe((value: string) => {
            this.userNameLength = value.length;
        });
    }
```

```
36.
37.save() {
38.         this.loopThroughControls(this.form);
39.     }
40.
41.     loopThroughControls(formGroup: FormGroup) {
42.         Object.keys(formGroup.controls).forEach((key: string) => {
43.             const controlName = formGroup.get(key);
44.             if (controlName instanceof FormGroup) {
45.                 this.loopThroughControls(controlName);
46.             } else {
47.                 console.log(`${key} - ${controlName.value}`);
48.             }
49.         });
50.     }
51.
52.     laodData() {
53.         this.form.patchValue({
54.             userName: "DivFaisal",
55.             email: "test@test.com",
56.             password: "12345",
57.             confirmPassword: "12345",
58.             gender: "male",
59.             address: {
60.                 city: "Riyadh",
61.                 state: "AL",
62.                 zipCode: "876786"
63.             }
64.         });
65.     }
66.
67.     get userName() {
68.         return this.form.get("userName");
69.     }
70.     get email() {
71.         return this.form.get("email");
72.     }
73.     get password() {
74.         return this.form.get("password");
75.     }
76.     get confirmPassword() {
77.         return this.form.get("confirmPassword");
78.     }
79.     get gender() {
80.         return this.form.get("gender");
81.     }
82.     get address() {
83.         return this.form.get("address");
84.     }
85.     get city() {
86.         return this.form.get("address").get("city");
87.     }
88.     get state() {
```

```
89.         return this.form.get("address").get("state");
90.     }
91.     get zipCode() {
92.         return this.form.get("address").get("zipCode");
93.     }
94. }
95.
```

راجع الاسطر من 37 إلى 50

الآن لنقم بتشغيل المشروع على المتصفح مع فتح أدوات المطور والذهاب لconsole ونرى النتيجة:



نلاحظ أنه بعد الضغط على زر save ظهرت لنا النتيجة في console على اليمين وهي عبارة عن أسماء الأدوات وقيم كل أداة وبما أن النموذج فارغ فبالتأكيد أن القيم تكون null.

الفصل الرابع

Validations in Angular Reactive Forms

1.4. المقدمة:

بعكس Template Driven Forms التي يتم التحقق من الصحة بالاعتماد على مجموعة من الدايكرتيف والخصائص الجاهزة والتي يتم كتابتها بشكل مباشر في تاغات HTML، هنا يتم التحقق برمجياً في ملف ts للcomponent وفي مثالنا هنا اسمه app.component.ts، وقبل البدأ في دراسة هذا القسم لابد من الرجوع إلى التحقق من الصحة في template driven forms لأننا سوف نستخدم خصائص مشابهه لما تم شرحه في القسم السابق (touched – dirty - minlength – required ...etc. -) لذلك ومنعاً لتكرار يجب مراجعتها قبل الدخول في هذا الجزء، اما من ناحية التحقق من الصحة validation في angular reactive forms هو ينقسم إلى عدة أنواع، كالتالي:

- ✓ Built-In Validation
- ✓ Custom Validation
- ✓ Conditional Validation
- ✓ Asynchronous Validation
- ✓ Dynamic Validation

وسوف نتكلم عن جميع هذه الأنواع بشيء من التفصيل ماعدا النوع الأخير Dynamic Validation سوف نؤجل التكلم عنه إلى حين الوصول إلى الفصل الخامس Dynamic Forms.

2.4. التحقق من الصحة المبني ضمناً Built-In Validation:

على الرغم من أن التحقق من الصحة يتم برمجياً إلى أن angular reactive forms قدمت لنا مجموعة من الدوال الجاهزة والمبنية ضمناً في الكلاس Validator وهذه الدوال مشابهه للخصائص التي تكلمنا عنها فيما سبق في angular template driven forms مثل required – pattern – touched – dirty – ...etc.، ولو رجعنا إلى الجزء الخاص في بناء النموذج برمجياً لوجدنا أني أشرت أن الخاصية البرمجية لكل أداة تستقبل قيمة على شكل مصفوفة وهذه المصفوفة تحتوي على ثلاث أجزاء الجزء الأول هو القيمة الافتراضية للأداة والثاني هو التحقق من الصحة validation وهذا هو الذي يهمنا في هذا الفصل، حيث هنا نقوم بكتابة دوال التحقق من الصحة، كما في الشكل التالي:

كما هو موضح في الشكل السابق حيث في الخاصية البرمجية username الخاصة بأداة إدخال اسم المستخدم، نضع اكواد التحقق من صحة القيم، وفي حال كان لدينا أكثر من validation للأداة الواحدة تكون هي ايضاً على هيئة مصفوفة، كالتالي:

userName: [null, [validation1, validation2, ...etc.]]

ملاحظة: منعاً لتكرار، الرجاء مراجعة الفصول السابقة وبالتحديد في فصل template driven farms validation لأن دوال التحقق من الصحة هنا مشابهه في عملها واسمائها لما هو موجود هناك.

الآن للاستفادة من هذه الدوال يجب استدعاء الكلاس Validator ثم إضافة الدوال لكل خاصية بحسب المطلوب وما نريد التحقق منه، كما في الجدول التالي:

دالة التحقق	نوع التحقق	الخاصية البرمجية للأداة
Validators.required	الحقل لا يقبل قيمة فارغة	userName
Validators. minlength(3) **كما يمكن استخدام pattern**	اقل عدد خانات مسموح ٣ خانات	
Validators.required	الحقل لا يقبل قيمة فارغة	email
Validators. Email **كما يمكن استخدام pattern**	صحة صيغة البريد الإلكتروني	
Validators.required	الحقل لا يقبل قيمة فارغة	password
Validators.pattern('(?=.*[A-Za-z])(?=.*[A-Z])(?=.*[0-9])[A-Za-z0-9\d$@].{5,}')	كلمة السر على الأقل ست خانات وتحتوي على حروف كبيرة وصغيرة وأرقام وبدون مسافات	
Validators.required	الحقل لا يقبل قيمة فارغة	confirmPassword
Validators.required	الحقل لا يقبل قيمة فارغة	gender
Validators.required	الحقل لا يقبل قيمة فارغة	city
Validators.required يفضل لإضافة الخاصية [ngValue]= 'null' إذا كانت هنالك قيمة افتراضية غير مطلوبة في أداة الاختيار	الحقل لا يقبل قيمة فارغة	state
Validators.required	الحقل لا يقبل قيمة فارغة الحقل لا يقبل إلا قيم رقمية وعلى	zipCode
Validators.pattern('^[0-9]{5}$')	الأقل خمس خانات	

الآن لنضيف الأكواد إلى ملف app.component.ts وفقاً لشروط الصحة التي ذكرناها في الجدول السابق:

ملف app.component.ts

```
1. import { Component, OnInit } from '@angular/core';
2.
3. import { FormGroup, FormBuilder, Validators } from '@angular/forms';
4.
5. @Component({
6.     selector: 'app-root',
7.     templateUrl: './app.component.html',
8.     styleUrls: ['./app.component.css']
9. })
10.export class AppComponent implements OnInit {
11.     private states: string[] = ['Riyadh', 'Makkah', ' AlSharqiyah', ' AlQasim'];
12.     form: FormGroup;
13.     userNameLength: any = '0';
14.
15.     constructor(private fb: FormBuilder) { }
16.
17.     ngOnInit() {
18.         this.form = this.fb.group({
19.             userName: [null, [Validators.required, Validators.minLength(3)]],
20.             email: [null, [Validators.required, Validators.email]],
21.             password: [
22.                 null,
23.                 [
24.                     Validators.required,
25.                     Validators.pattern(
26.                         '(?=.*[A-Za-z])(?=.*[A-Z])(?!.* )(?=.*[0-9])[A-Za-z0-
   9d$@].{5,}'
27.                     )
28.                 ]
29.             ],
30.             confirmPassword: [null, Validators.required],
31.             gender: [null, Validators.required],
32.             address: this.fb.group({
33.                 city: [null, Validators.required],
34.                 state: [null, Validators.required],
35.                 zipCode: [null, [Validators.required, Validators.pattern('^[0-
   9]{5}$')]]
36.             })
37.         });
38.
39.         this.userName.valueChanges.subscribe((value: string) => {
40.             this.userNameLength = value.length;
41.         });
42.     }
43.
44.     save() {
45.         this.loopThroughControls(this.form);
46.         console.log(this.form);
47.     }
```

```
48.
49.     loopThroughControls(formGroup: FormGroup) {
50.         Object.keys(formGroup.controls).forEach((key: string) => {
51.             const controlName = formGroup.get(key);
52.             if (controlName instanceof FormGroup) {
53.                 this.loopThroughControls(controlName);
54.             } else {
55.                 console.log(`${key} - ${controlName.value}`);
56.             }
57.         });
58.     }
59.
60.     laodData() {
61.         this.form.patchValue({
62.             userName: 'DivFaisal',
63.             email: 'test@test.com',
64.             password: '12345',
65.             confirmPassword: '12345',
66.             gender: 'male',
67.             address: {
68.                 city: 'Riyadh',
69.                 state: 'AL',
70.                 zipCode: '876786'
71.             }
72.         });
73.     }
74.
75.     get userName() {
76.         return this.form.get('userName');
77.     }
78.     get email() {
79.         return this.form.get('email');
80.     }
81.     get password() {
82.         return this.form.get('password');
83.     }
84.     get confirmPassword() {
85.         return this.form.get('confirmPassword');
86.     }
87.     get gender() {
88.         return this.form.get('gender');
89.     }
90.     get address() {
91.         return this.form.get('address');
92.     }
93.     get city() {
94.         return this.form.get('address').get('city');
95.     }
96.     get state() {
97.         return this.form.get('address').get('state');
98.     }
99.     get zipCode() {
100.         return this.form.get('address').get('zipCode');
```

```
101.       }
102.   }
103.
```

راجع السطر 3 (استدعاء الكلاس)

راجع الاسطر من 17 إلى 37 (كتابة أكواد التحقق من الصحة)

أما في ملف app.component.html فسوف نقوم بإظهار وإخفاء رسائل الخطأ تبعاً لشروط التحقق من الصحة التي قمنا في كتابتها قبل قليل في ملف app.component.ts، ومنعاً لتكرار فهي مشابهه لما قمنا به في قسم template driven forms validation لذلك لن أقوم بشرح الخطوات وإنما الاكتفاء بكتابة الاكواد فقط وفي حال الرغبة بفهم اكثر الرجاء الرجوع إلى الشرح الذي قمنا به في الفصول السابقة الذكر.

app.component.html

```
1. <div class="container-fluid">
2.   <div class="card mx-auto col-sm-12 col-md-6 col-lg-6 col-xl-6 pt-sm-4">
3.     <form [formGroup]="form">
4.       <div class="card-header">
5.         <h4 class="text-center">Reactive Forms</h4>
6.       </div>
7.
8.       <div class="card-body">
9.         <!-- أداة إدخال اسم المستخدم -->
10.        <div class="form-group">
11.          <label>User Name</label>
12.          <div class="form-inline">
13.            <input
14.              formControlName="userName"
15.              [ngClass]="{
16.                'col-10': true,
17.                'form-control': true,
18.                'is-invalid': userName.invalid && userName.touched
19.              }"
20.            />
21.            <label class="col-2">{{ userNameLength }}</label>
22.          </div>
23.          <!-- جزء التحقق من الصحة الخاص باسم المستخدم -->
24.          <div *ngIf="userName.invalid && userName.touched">
25.            <small class="text-danger" *ngIf="userName.hasError('required')">
26.              اسم المستخدم مطلوب
27.            </small>
28.            <small class="text-danger" *ngIf="userName.hasError('minlength')">
29.              اسم المستخدم على الاقل ثلاث خانات
30.            </small>
31.          </div>
32.        </div>
33.
34.        <!-- أداة إدخال البريد الإلكتروني -->
35.        <div class="form-group">
36.          <label>Email</label>
```

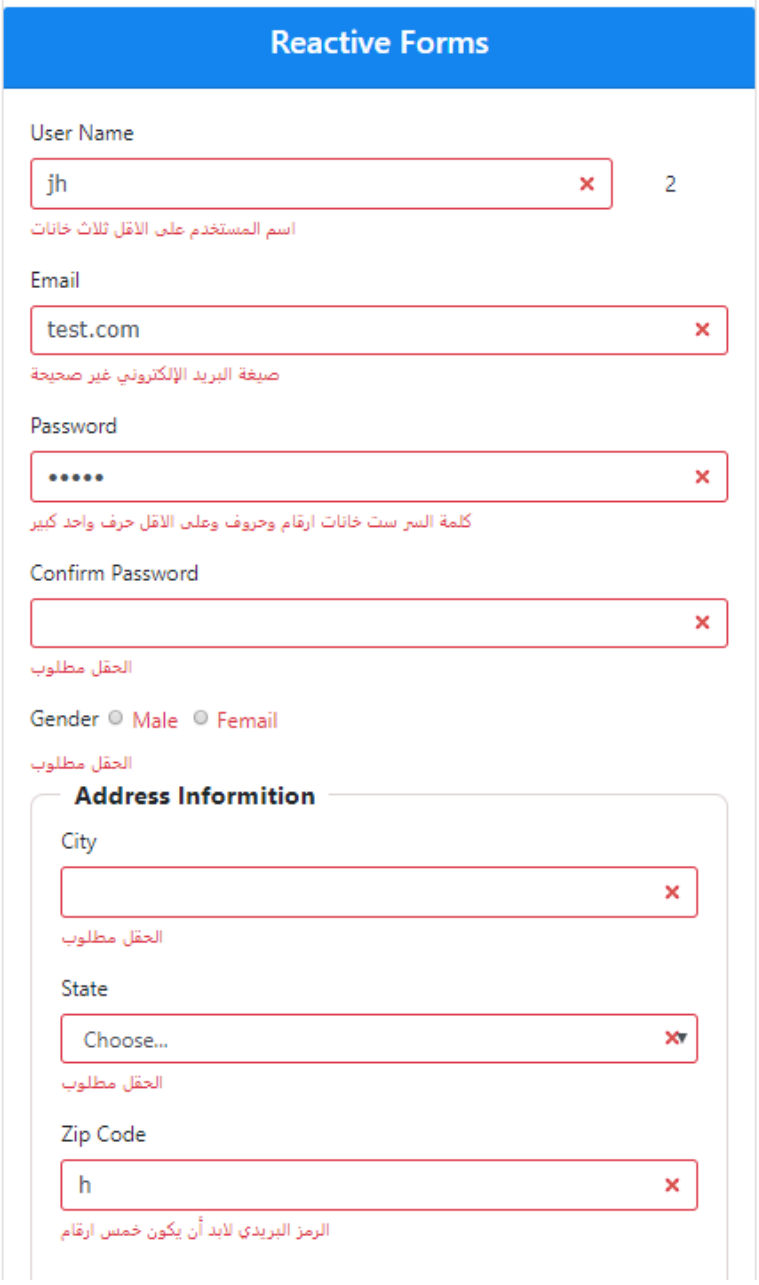
```
37.          <input
38.            type="email"
39.            formControlName="email"
40.            [ngClass]="{
41.              'form-control': true,
42.              'is-invalid': email.invalid && email.touched
43.            }"
44.          />
45.          <!-- جزء التحقق من الصحة الخاص بالبريد الإلكتروني -->
46.          <div *ngIf="email.invalid && email.touched">
47.            <small class="text-danger" *ngIf="email.hasError('required')">
48.              البريد الإلكتروني مطلوب
49.            </small>
50.            <small class="text-danger" *ngIf="email.hasError('email')">
51.              صيغة البريد الإلكتروني غير صحيحة
52.            </small>
53.          </div>
54.        </div>
55.
56.        <!-- أداة إدخال كلمة السر -->
57.        <div class="form-group">
58.          <label>Password</label>
59.          <input
60.            type="password"
61.            formControlName="password"
62.            [ngClass]="{
63.              'form-control': true,
64.              'is-invalid': password.invalid && password.touched
65.            }"
66.          />
67.          <!-- جزء التحقق من الصحة الخاص بكلمة السر -->
68.          <div *ngIf="password.invalid && password.touched">
69.            <small class="text-danger" *ngIf="password.hasError('required')">
70.              كلمة االسر مطلوبة
71.            </small>
72.            <small class="text-danger" *ngIf="password.hasError('pattern')">
73.              بدون مسافات كلمة السر ست خانات ارقام وحروف وعلى الاقل حرف واحد
74.              كبير
75.            </small>
76.          </div>
77.        </div>
78.
79.        <!-- أداة إعادة إدخال كلمة السر -->
80.        <div class="form-group">
81.          <label>Confirm Password</label>
82.          <input
83.            type="password"
84.            formControlName="confirmPassword"
85.            [ngClass]="{
86.              'form-control': true,
87.              'is-invalid': confirmPassword.invalid && confirmPassword.touched
88.            }"
89.          />
```

```
90.           <!-- جزء التحقق من الصحة الخاص بأداة إعادة إدخال كلمة السر -->
91.           <div *ngIf="confirmPassword.invalid && confirmPassword.touched">
92.             <small
93.               class="text-danger"
94.               *ngIf="confirmPassword.hasError('required')"
95.             >
96.               الحقل مطلوب
97.             </small>
98.           </div>
99.         </div>
100.
101.          <!-- أداة تحديد نوع الجنس ذكر أو أنثى -->
102.          <label class="pr-2">Gender</label>
103.          <div class="form-check form-check-inline">
104.            <input
105.              type="radio"
106.              formControlName="gender"
107.              id="maleGender"
108.              value="male"
109.              [ngClass]="{
110.                'form-check-input': true,
111.                'is-invalid': gender.invalid && gender.touched
112.              }"
113.            />
114.            <label class="form-check-label" for="gender">Male</label>
115.          </div>
116.          <div class="form-check form-check-inline">
117.            <input
118.              type="radio"
119.              formControlName="gender"
120.              id="femaleGender"
121.              value="femail"
122.              [ngClass]="{
123.                'form-check-input': true,
124.                'is-invalid': gender.invalid && gender.touched
125.              }"
126.            />
127.            <label class="form-check-label" for="femaleGender">Femail</label>
128.          </div>
129.          <!-- الجزء الخاص بتحقق من الصحة لأداة تحديد نوع الجنس -->
130.          <div *ngIf="gender.invalid && gender.touched">
131.            <small class="text-danger" *ngIf="gender.hasError('required')">
132.              الحقل مطلوب
133.            </small>
134.          </div>
135.
136.          <!-- بداية النموذج الفرعي -->
137.          <fieldset class="scheduler-border" formGroupName="address">
138.            <legend class="scheduler-border">Address Information</legend>
139.            <div class="form-group">
140.              <!-- أداة إدخال اسم المدينة -->
141.              <label>City</label>
142.              <input
```

```
143.                 formControlName="city"
144.                 [ngClass]="{
145.                   'form-control': true,
146.                   'is-invalid': city.invalid && city.touched
147.                 }"
148.               />
149.               <!-- جزء التحقق من الصحة لأداة إدخال اسم المدينة -->
150.               <div *ngIf="city.invalid && city.touched">
151.                 <small class="text-danger" *ngIf="city.hasError('required')">
152.                   الحقل مطلوب
153.                 </small>
154.               </div>
155.             </div>
156.             <!-- أداة اختيار اسم المنطقة او الولاية -->
157.             <div class="form-group">
158.               <label>State</label>
159.               <select
160.                 formControlName="state"
161.                 [ngClass]="{
162.                   'form-control': true,
163.                   'is-invalid': state.invalid && state.touched
164.                 }"
165.               >
166.                 <option selected [ngValue]="null">Choose...</option>
167.                 <option *ngFor="let item of states" [value]="item">
168.                   {{ item }}
169.                 </option>
170.               </select>
171.               <!-- جزء التحقق من الصحة لأداة أختيار اسم المنطقة -->
172.               <div *ngIf="state.invalid && state.touched">
173.                 <small class="text-danger" *ngIf="state.hasError('required')">
174.                   الحقل مطلوب
175.                 </small>
176.               </div>
177.             </div>
178.             <!-- أداة إدخال الرمز البريدي -->
179.             <div class="form-group">
180.               <label>Zip Code</label>
181.               <input
182.                 formControlName="zipCode"
183.                 [ngClass]="{
184.                   'form-control': true,
185.                   'is-invalid': zipCode.invalid && zipCode.touched
186.                 }"
187.               />
188.               <!-- جزء التحقق من الصحة لأداة إدخال الرمز البريدي -->
189.               <div *ngIf="zipCode.invalid && zipCode.touched">
190.                 <small class="text-danger" *ngIf="zipCode.hasError('required')">
191.                   الحقل مطلوب
192.                 </small>
193.                 <small class="text-danger" *ngIf="zipCode.hasError('pattern')">
194.                   الرمز البريدي لابد أن يكون خمس ارقام
195.                 </small>
```

```
196.             </div>
197.           </div>
198.         </fieldset>
199.       </div>
200.
201.       <!-- بداية أدوات الأزرار -->
202.       <div class="card-footer">
203.         <button class="btn btn-primary" (click)="save()">Save</button>
204.         <button class="btn btn-primary ml-3" (click)="laodData()">
205.           Load Data
206.         </button>
207.       </div>
208.     </form>
209.   </div>
210. </div>
211.
```

الآن لنقوم بتشغيل النموذج ونرى ما قمنا به من تعديلات عليه من خلال المتصفح:



1.2.4. نقل رسائل التحقق من الصحة إلى ملف class للComponent:

في هذا الجزء سوف نتكلم عن طريقة نقل رسائل الخطأ من ملف template (html) إلى ملف ts (class) بمعنى في مثالنا هذا من ملف app.component.html إلى ملف app.component.ts، وذلك لعدة أسباب:

- الإبقاء على ملف tamplate نظيفاً من أكواد الـangular البرمجية او logic البرمجي بشكل عام على قدر المستطاع بحيث يحتوي على أكواد html فقط، لأننا نتعامل مع النموذج برمجياً.
- سهولة عمل Testing للكود بما انه في ملف ts.
- ممكن ان تكون رسائل الخطأ قادمة من السيرفر وهذه الطريقة هي المناسبة.
- إذا أردنا أن تكون الرسائل ديناميكية فلابد من اللجوء إلى هذه الطريقة.

وتكمن الفكرة للقيام بهذا الأمر من خلال إنشاء كائنين الكائن الأول يحتوي على جميع رسائل التحقق من الصحة والكائن الثاني يحتوي على الرسائل التي تظهر للمستخدم عند وقوع خطأ منه، بحيث يكون الكائن الأول كمخزن لجميع دوال التحقق من الصحة ورسائل الخطأ الخاصة بكل دالة والكائن الثاني كمخزن مؤقت نضع فيه رسالة الخطأ في حال تحقق أي من دوال التحقق من الصحة وتحذف منه في حال عدم وجود أي خطأ وهذه الكائن هو الذي نربطه في ملف html او ملف app.component.html، وسوف تكون اشكال الكائنين كالتالي:

الكائن الثاني

```
currentMessageValidation = {
  userName: '',
  email: '',
  password: '',
  confirmPassword: '',
  gender: '',
  city: '',
  state: '',
  zipCode: ''
};
```

الكائن الأول

```
messageValidation = {
    userName: {
      required: 'اسم المستخدم مطلوب',
      minlength: 'اسم المستخدم على الاقل ثلاث خانات'
    },
    email: {
      required: ' البريد الإلكتروني مطلوب',
      email: 'صيغة البريد الإلكتروني غير صحيحة'
    },
    password: {
      required: 'كلمة االسر مطلوبة',
      pattern: 'كلمة السر ست خانات ارقام وحروف وعلى الاقل حرف واحد كبير  بدون مسافات'
    },
    confirmPassword: {
      required: ' الحقل مطلوب'
    },
    gender: {
      required: 'الحقل مطلوب'
    },
    city: {
      required: 'حقل اسم المدينة مطلوب'
    },
    state: {
      required: 'حقل المنطقة مطلوب'
    },
    zipCode: {
      required: 'حقل الرمز البريدي مطلوب',
      pattern: 'الرمز البريدي لابد أن يكون قيمة رقمية من خمس خانات'
    }
  };
```

نلاحظ أن الكائن الأول الذي اسميناه massageValidation يحتوي على keys تشبه الأسماء البرمجية للأدوات وكل key يحتوي بداخله خصائص لها نفس اسماء دوال التحقق لكل أداة وكل دالة اضفنا لها رسالة الخطأ الخاصة بها، أما الكائن الثاني اسميناه currentMessageValidation ويحتوي على keys لها نفس الأسماء البرمجية للأدوات وقيمها فارغة، بحيث كما قلنا سابقاً الكائن الأول يكون مخزن ثابت لكل دوال التحقق من الصحة ورسائل الخطأ الخاصة بها والكائن الثاني نخزن فيه رسائل الخطأ في حال تحقق أي خطأ من الأخطاء – أي من دوال التحقق من الصحة ارجعت القيمة true – وفي حال عدم وجود هذه الخطأ تحذف الرسالة من هذا الكائن او تستبدل مكانها رسالة خطأ أخرى في حال وجود أخطاء، مع العلم أن keys التي في الكائن الثاني تُرجع القيمة true في حال وجود رسالة خطأ بداخله وسوف نستفيد من هذه القيمة في ملف html من ناحية إظهار أو إخفاء كلاسات bootstrap.

وهذه الطريقة تساعدنا وتسهل لنا في حال أردنا إضافة او استبدال او حذف أي من دوال التحقق من الصحة فإذا أردنا مثلاً استبدال الدالة minlength في userName إلى الدالة pattern فكل الذي نعمله نغير الدالة اولاً في موقعها الأساسي في مرحلة انشاء النموذج برمجياً ثم نقوم بتغييره في الكائن massageValidation ونفس الطريقة في حال الإضافة او الحذف، كالتالي:

app.component.ts جزء من ملف

```
this.form = this.fb.group({
    userName: [null, [Validators.required, Validators.pattern('.{3,}')]],
    email: [null, [Validators.required, Validators.email]],
    password: [
        null,
        [
            Validators.required,
            Validators.pattern(
                '(?=.*[A-Za-z])(?=.*[A-Z])(?=.*[0-9])[A-Za-z0-9d$@].{5,}'
            )
        ]
    ],
    confirmPassword: [null, Validators.required],
    gender: [null, Validators.required],
    address: this.fb.group({
        city: [null, Validators.required],
        state: [null, Validators.required],
        zipCode: [null, [Validators.required, Validators.pattern('^[0-9]{5}$')]]
    })
});
```

app.component.ts جزء من ملف

```
messageValidation = {
    userName: {
        required: 'اسم المستخدم مطلوب',
        pattern: 'اسم المستخدم على الاقل ثلاث خانات'
    },
    email: {
        required: ' البريد الإلكتروني مطلوب',
        email: 'صيغة البريد الإلكتروني غير صحيحة'
```

```
    },
    password: {
        required: 'كلمة االسر مطلوبة',
        pattern: 'بدون مسافات كلمة السر ست خانات ارقام وحروف وعلى الاقل حرف واحد كبير'
    },
    confirmPassword: {
        required: ' الحقل مطلوب'
    },
    gender: {
        required: 'الحقل مطلوب'
    },
    city: {
        required: 'حقل اسم المدينة مطلوب'
    },
    state: {
        required: 'حقل المنطقة مطلوب'
    },
    zipCode: {
        required: 'حقل الرمز البريدي مطلوب',
        pattern: 'الرمز البريدي لابد أن يكون قيمة رقمية من خمس خانات'
    }
};
```

والشكل التالي يوضح طريقة سير رسالة الخطأ في حال حدوثها وليكن مثلاً الخطأ هو أن صيغة البريد الإلكتروني غير صحيح:

ملف app.component.ts

```
messageValidation = {
  userName: {
    required: 'اسم المستخدم مطلوب',
    pattern: 'اسم المستخدم على الاقل ثلاث خانات'
  },
  email: {
    required: ' البريد الإلكتروني مطلوب',
    email: 'صيغة البريد الإلكتروني غير صحيحة'
  },
  password: {
    required: 'كلمة السر مطلوبة',
```

```
currentMessageValidation = {
    userName: '',
    email: 'صيغة البريد الإلكتروني غير صحيحة',
    password: '',
    confirmPassword: '',
    gender: '',
    city: '',
    state: '',
    zipCode: ''
  };
```

في حال وجود خطأ بمعنى أصبحت دالة التحقق من الصحة email تساوي true

في هذه الحالة يتم نقل رسالة الخطأ إلى الكائن الثاني وتخزين الرسالة مؤقتاً في KEY المناسب له تمهيداً لعرضها للمستخدم في ملف APP.COMPONENT.HTML

ونقوم بعمل ربط للكائن الثاني مع ملف html في الكود الخاص بالأداة حيث يُرجع key قيمة true في حال احتوائه على رسالة خطأ ونستفيد منه في إظهار وإخفاء رسائل الخطأ وكلاسات البوتستراب

ملف app.component.html

```
<input type="email" formControlName="email" [ngClass]="{ 'form-control': true,
'is-invalid': currentMessageValidation.email}"/>
<small class="text-danger" *ngIf="currentMessageValidation.email">
  {{ currentMessageValidation.email }}
</small>
```

شرط إظهاروإخفاء كلاسات البوتستراب

محتوى رسالة الخطأ

شرط إظهاروإخفاء رسالة الخطأ

الآن بعد استعراضنا لآلية عمل وطريقة إظهار رسالة الخطأ، سوف نقوم بكتابة الكود logic الذي يقوم بهذا العمل، وللقيام بهذا الأمر سوف نستفيد من دالة loopThroughControls() التي أنشأناها سابقاً وقمنا بشرحها بشكل مفصل في الجزء الخاص بعمل Loop على أدوات النموذج برمجياً، وكانت الدالة بشكل التالي:

```
1.  loopThroughControls(formGroup: FormGroup) {
2.    Object.keys(formGroup.controls).forEach((key: string) => {
3.      const controlName = formGroup.get(key);
4.      if (controlName instanceof FormGroup) {
5.        this.loopThroughControls(controlName);
6.      } else {
7.
8.
9.
10.
11.
12.     }
13.   });
14. }
```

هنا نكتب logic بعد else

بما أن هذا الكود هو حلقة تكرار فأن الثابت controlName في كل دورة يمثل أداة من أدوات النموذج لذلك نستطيع الاستفادة منه في التحقق من أنه يحتوي على قيمة وان قيمته غير صحيحة وتم لمسه أو التعديل عليه فإذا تحققت هذه الشروط ينفذ ما بداخل الشرط بمعنى أن هنالك خطأ وقع، بحيث يكون شكل السطر البرمجي:

```
1.  if (controlName && controlName.invalid && (controlName.touched || controlName.dirty)) {
2.
3.
4.  }
```

بعد أن عرفنا أن هنالك خطأ بقي علينا أن نعرف ما هو هذا الخطأ أو مجموعة الأخطاء في هذه الأداة ولعمل ذلك اولاً لابد من معرفة الاسم البرمجي لهذه الأداة هل هي UserName أم email ام...الخ، ولو لاحظنا أن هذه الأسماء موجودة في المتغير key في حلقة التكرار، فإذا كانت controlName تمثل الحقل password فإن الاسم البرمجي لها يكون مخزن في المتغير النصي الذي اسمه key وفي الدورة الثانية لحلقة التكرار إذا كان controlName يمثل city فإن الاسم البرمجي لها مخزن في المتغير النصي key وهكذا في بقية الأدوات، ومن هذا المنطلق نستطيع معرفة أي key في الكائن الأول massageValidation هو المقصود في الخطأ لأن رسائل الخطأ كما قلنا سابقاً تم تخزينها في هذا الكائن لذلك سوف نقوم بتعريف ثابت باسم massages ومهمته تخزين رسائل الخطأ الخاصة بكل أداة في كل دورة، فمثلا لو كانت controlName تمثل أسم المستخدم والمتغير النصي ذو الاسم key يحتوي على الاسم البرمجي والذي هو userName فإن الثابت massages تكون قيمته:

```
{
        Pattern: "اسم المستخدم على الاقل ثلاث خانات"
        required: "اسم المستخدم مطلوب"
}
```

وهكذا مع بقية الأدوات في كل دورة في حلقة التكرار تتغير قيمة massages بناءً على قيم key وcontrolName، الآن لنترجم هذا الشرح المطول على شكل سطر برمجي واحد بحيث يكون كالتالي:

```
1.  if (controlName && controlName.invalid && (controlName.touched || controlName.dirty)) {
2.
3.          const messages = this.messageValidation[key];
4.
5.  }
```

الآن بعد أن وصلنا إلى الأداة التي فيها الخطأ في الكائن الأول وخزنا جميع رسائل الخطأ في المتغير massages بقي علينا أن ننقل هذه الرسائل إلى الكائن الثاني لعرضها للمستخدم، ونستطيع القيام بهذا الأمر من خلال عمل حلقة تكرار for لجميع رسائل الخطأ في الأداة وتخزينها في ثابت ومن ثم اضافتها إلى الكائن الثاني currentMessageValidation عن طريق المتغير النصي key الذي يحتوي على الاسم البرمجي للأداة كما قلنا سابقاً لكي نحدد أي key في هذا الكائن تتم إضافة رسالة الخطأ إليه لأن جميع key في الكائن الأول والثاني والاسم البرمجي لها نفس الاسم، اما أي دالة من دوال التحقق من الصحة التي حدث فيها الخطأ فنستطيع معرفتها بكل بساطة من خلال الثابت massages لأننا وضعنا أسماء الرسائل مشابهه لدول التحقق من الصحة، بحيث يصبح الكود كالتالي:

```
if (controlName && controlName.invalid && (controlName.touched || controlName.dirty)) {

        const messages = this.messageValidation[key];
        for (const controlError in controlName.errors) {
           if (controlError) {
              this.currentMessageValidation[key] += messages[controlError] + ' ';
           }
        }

}
```

بقي أخيراً شيئين الأول هو تفريغ الكائن الثاني مع كل دورة لكيلا تتكرر الرسائل نفسها في كل مرة والشيء الثاني هو وضع قيمة افتراضية للبراميتر FormGroup الذي مررناه للدالة وهذه القيمة هي النموذج نفسه الذي اشرنا إليه بالمتغير form:

```
loopThroughControls(formGroup: FormGroup = this.form) {
  Object.keys(formGroup.controls).forEach((key: string) => {
    const controlName = formGroup.get(key);
    if (controlName instanceof FormGroup) {
      this.loopThroughControls(controlName);
    } else {
      this.currentMessageValidation[key] = '';
      if (
        controlName && controlName.invalid && (controlName.touched || controlName.dirty)
      ) {
        const messages = this.messageValidation[key];
        for (const controlError in controlName.errors) {
          if (controlError) {
            this.currentMessageValidation[key] += messages[controlError] + ' ';
          }
        }
      }
    }
  });
}
```

بقي أن نقوم بمراقبة التعديلات على النموذج عن طريق valueChanges ومن ثم تنفيذ الدالة السابقة في حال وجود أي تعديل على النموذج بحيث يكون الكود بالشكل التالي:

```
this.form.valueChanges.subscribe(data => {
  this.loopThroughControls(this.form);
});
```

الآن لنضيف هذه الأكواد جميعها إلى ملف app.component.ts:

ملف app.component.ts

```
import { Component, OnInit } from '@angular/core';

import { FormGroup, FormBuilder, Validators } from '@angular/forms';

@Component({
    selector: 'app-root',
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css']
})

export class AppComponent implements OnInit {
    states: string[] = ['ALRiyadh', 'Makkah', 'AlSharqiyah', 'AlQasim'];
    form: FormGroup;
```

```
14.     userNameLength: any = '0';
15.
16.     messageValidation = {
17.         userName: {
18.             required: 'اسم المستخدم مطلوب',
19.             pattern: 'اسم المستخدم على الاقل ثلاث خانات'
20.         },
21.         email: {
22.             required: ' البريد الإلكتروني مطلوب',
23.             email: 'صيغة البريد الإلكتروني غير صحيحة'
24.         },
25.         password: {
26.             required: 'كلمة االسر مطلوبة',
27.             pattern: 'بدون مسافات كلمة السر ست خانات ارقام وحروف وعلى الاقل حرف واحد كبير'
28.         },
29.         confirmPassword: {
30.             required: ' الحقل مطلوب'
31.         },
32.         gender: {
33.             required: 'الحقل مطلوب'
34.         },
35.         city: {
36.             required: 'حقل اسم المدينة مطلوب'
37.         },
38.         state: {
39.             required: 'حقل المنطقة مطلوب'
40.         },
41.         zipCode: {
42.             required: 'حقل الرمز البريدي مطلوب',
43.             pattern: 'الرمز البريدي لابد أن يكون قيمة رقمية من خمس خانات'
44.         }
45.     };
46.
47.     currentMessageValidation = {
48.         userName: '',
49.         email: '',
50.         password: '',
51.         confirmPassword: '',
52.         gender: '',
53.         city: '',
54.         state: '',
55.         zipCode: ''
56.     };
57.
58.     constructor(private fb: FormBuilder) { }
59.
60.     ngOnInit() {
61.         this.form = this.fb.group({
62.             userName: [
63.                 null,
64.                 [
65.                     Validators.required,
66.                     Validators.pattern('.{3,}')
```

```
                ]
            ],
            email: [
                null,
                [
                    Validators.required,
                    Validators.email
                ]
            ],
            password: [
                null,
                [
                    Validators.required,
                    Validators.pattern(
                        '(?=.*[A-Za-z])(?=.*[A-Z])(?!.* )(?=.*[0-9])[A-Za-z0-9d$@].{5,}'
                    )
                ]
            ],
            confirmPassword: [null, Validators.required],
            gender: [null, Validators.required],
            address: this.fb.group({
                city: [null, Validators.required],
                state: [null, Validators.required],
                zipCode: [
                    null,
                    [
                        Validators.required,
                        Validators.pattern('^[0-9]{5}$')
                    ]
                ]
            })
        });

        this.userName.valueChanges.subscribe((value: string) => {
            this.userNameLength = value.length;
        });

        this.form.valueChanges.subscribe(data => {
            this.loopThroughControls(this.form);
        });

    }

    save() {
        this.loopThroughControls(this.form);
    }

    loopThroughControls(formGroup: FormGroup = this.form) {
        Object.keys(formGroup.controls).forEach((key: string) => {
            const controlName = formGroup.get(key);
            if (controlName instanceof FormGroup) {
                this.loopThroughControls(controlName);
```

```
119.             } else {
120.                 this.currentMessageValidation[key] = '';
121.                 if (
122.                     controlName &&
123.                     controlName.invalid &&
124.                     (controlName.touched || controlName.dirty)
125.                 ) {
126.                     const messages = this.messageValidation[key];
127.                     for (const controlError in controlName.errors) {
128.                         if (controlError) {
129.                             this.currentMessageValidation[key] += messages[controlEr
    ror] + ' ';
130.                         }
131.                     }
132.                 }
133.             }
134.         });
135.     }
136.
137.     laodData() {
138.         this.form.patchValue({
139.             userName: 'DivFaisal',
140.             email: 'test@test.com',
141.             password: '12345',
142.             confirmPassword: '12345',
143.             gender: 'male',
144.             address: {
145.                 city: 'Riyadh',
146.                 state: 'AL',
147.                 zipCode: '876786'
148.             }
149.         });
150.     }
151.
152.     get userName() {
153.         return this.form.get('userName');
154.     }
155.     get email() {
156.         return this.form.get('email');
157.     }
158.     get password() {
159.         return this.form.get('password');
160.     }
161.     get confirmPassword() {
162.         return this.form.get('confirmPassword');
163.     }
164.     get gender() {
165.         return this.form.get('gender');
166.     }
167.     get address() {
168.         return this.form.get('address');
169.     }
170.     get city() {
```

```
171.           return this.form.get('address').get('city');
172.       }
173.       get state() {
174.           return this.form.get('address').get('state');
175.       }
176.       get zipCode() {
177.           return this.form.get('address').get('zipCode');
178.       }
179.   }
180.
```

وبنفس الوقت نقوم بعمل التعديلات اللازمة في ملف app.component.html، مع ملاحظة إضافة الدالة loopThroughControls() إلى كل أداة في الحدث blur والسبب نريد أن تُفعل هذه الدالة في حال أن المستخدم قام بلمس الأداة ولكن لم يقوم بالتعديل لأن valueChanges تعمل في حال التعديل فقط على أدوات النموذج، بحيث يكون الكود كالتالي:

ملف app.component.html

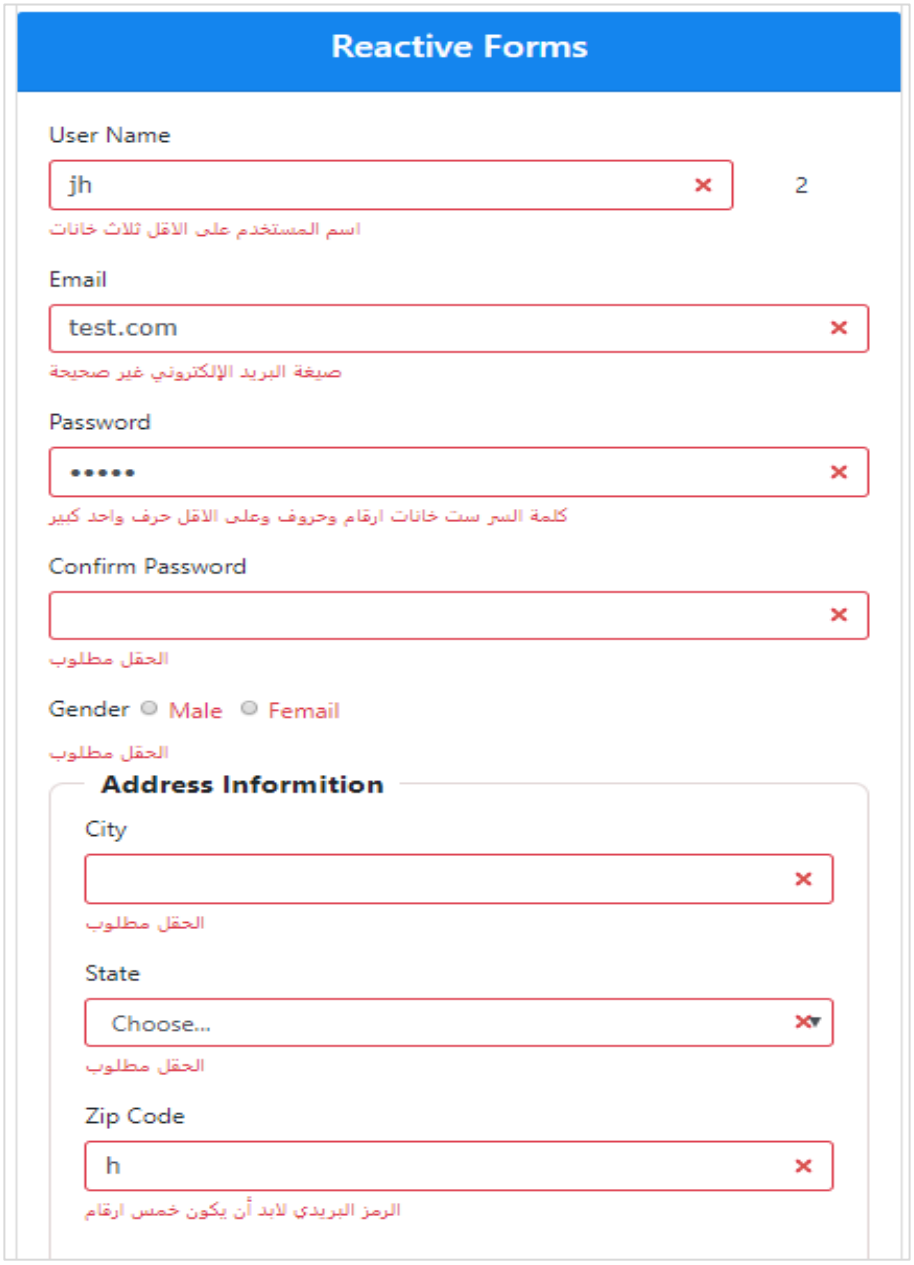
```
1. <div class="container-fluid">
2.   <div class="card mx-auto col-sm-12 col-md-6 col-lg-6 col-xl-6 pt-sm-4">
3.     <form [formGroup]="form">
4.       <div class="card-header">
5.         <h4 class="text-center">Reactive Forms</h4>
6.       </div>
7.
8.       <div class="card-body">
9.         <!-- أداة إدخال اسم المستخدم -->
10.        <div class="form-group">
11.          <label>User Name</label>
12.          <div class="form-inline">
13.            <input
14.              formControlName="userName"
15.              [ngClass]="{
16.                'col-10': true,
17.                'form-control': true,
18.                'is-invalid': currentMessageValidation.userName
19.              }"
20.              (blur)="loopThroughControls()"
21.            />
22.            <label class="col-2">{{ userNameLength }}</label>
23.          </div>
24.          <!-- جزء التحقق من الصحة الخاص باسم المستخدم -->
25.          <small class="text-danger" *ngIf="currentMessageValidation.userName">
26.            {{ currentMessageValidation.userName }}
27.          </small>
28.        </div>
29.
30.        <!-- أداة إدخال البريد الإلكتروني -->
31.        <div class="form-group">
32.          <label>Email</label>
```

```
33.           <input
34.             type="email"
35.             formControlName="email"
36.             [ngClass]="{
37.               'form-control': true,
38.               'is-invalid': currentMessageValidation.email
39.             }"
40.             (blur)="loopThroughControls()"
41.           />
42.           <!-- جزء التحقق من الصحة الخاص بالبريد الإلكتروني -->
43.           <small class="text-danger" *ngIf="currentMessageValidation.email">
44.             {{ currentMessageValidation.email }}
45.           </small>
46.         </div>
47.
48.         <!-- أداة إدخال كلمة السر -->
49.         <div class="form-group">
50.           <label>Password</label>
51.           <input
52.             type="password"
53.             formControlName="password"
54.             [ngClass]="{
55.               'form-control': true,
56.               'is-invalid': currentMessageValidation.password
57.             }"
58.             (blur)="loopThroughControls()"
59.           />
60.           <!-- جزء التحقق من الصحة الخاص بكلمة السر -->
61.           <small class="text-danger" *ngIf="currentMessageValidation.password">
62.             {{ currentMessageValidation.password }}
63.           </small>
64.         </div>
65.
66.         <!-- أداة إعادة إدخال كلمة السر -->
67.         <div class="form-group">
68.           <label>Confirm Password</label>
69.           <input
70.             type="password"
71.             formControlName="confirmPassword"
72.             [ngClass]="{
73.               'form-control': true,
74.               'is-invalid': currentMessageValidation.confirmPassword
75.             }"
76.             (blur)="loopThroughControls()"
77.           />
78.           <!-- جزء التحقق من الصحة الخاص بأداة إعادة إدخال كلمة السر -->
79.           <small
80.             class="text-danger"
81.             *ngIf="currentMessageValidation.confirmPassword"
82.           >
83.             {{ currentMessageValidation.confirmPassword }}
84.           </small>
85.         </div>
```

```
86.
87.          <!-- -أداة تحديد نوع الجنس ذكر أو أنثى -->
88.          <label class="pr-2">Gender</label>
89.          <div class="form-check form-check-inline">
90.            <input
91.              type="radio"
92.              formControlName="gender"
93.              id="maleGender"
94.              value="male"
95.              [ngClass]="{
96.                'form-check-input': true,
97.                'is-invalid': currentMessageValidation.gender
98.              }"
99.              (blur)="loopThroughControls()"
100.              />
101.              <label class="form-check-label" for="gender">Male</label>
102.            </div>
103.            <div class="form-check form-check-inline">
104.              <input
105.                type="radio"
106.                formControlName="gender"
107.                id="femaleGender"
108.                value="femail"
109.                [ngClass]="{
110.                  'form-check-input': true,
111.                  'is-invalid': currentMessageValidation.gender
112.                }"
113.                (blur)="loopThroughControls()"
114.              />
115.              <label class="form-check-label" for="femaleGender">Femail</label>
116.            </div>
117.            <!-- -الجزء الخاص بتحقق من الصحة لأداة تحديد نوع الجنس -->
118.            <small class="text-danger" *ngIf="currentMessageValidation.gender">
119.              {{ currentMessageValidation.gender }}
120.            </small>
121.
122.            <!-- -بداية النموذج الفرعي -->
123.            <fieldset class="scheduler-border" formGroupName="address">
124.              <legend class="scheduler-border">Address Informition</legend>
125.              <div class="form-group">
126.                <!-- -أداة إدخال اسم المدينة -->
127.                <label>City</label>
128.                <input
129.                  formControlName="city"
130.                  [ngClass]="{
131.                    'form-control': true,
132.                    'is-invalid': currentMessageValidation.city
133.                  }"
134.                  (blur)="loopThroughControls()"
135.                />
136.                <!-- -جزء التحقق من الصحة لأداة إدخال اسم المدينة -->
137.                <small class="text-danger" *ngIf="currentMessageValidation.city">
138.                  {{ currentMessageValidation.city }}
```

```
            </small>
          </div>
          <!-- أداة اختيار اسم المنطقة او الولاية -->
          <div class="form-group">
            <label>State</label>
            <select
              formControlName="state"
              [ngClass]="{
                'form-control': true,
                'is-invalid': currentMessageValidation.state
              }"
              (blur)="loopThroughControls()"
            >
              <option selected [ngValue]="null">Choose...</option>
              <option *ngFor="let item of states" [value]="item">
                {{ item }}
              </option>
            </select>
            <!-- جزء التحقق من الصحة لأداة أختيار اسم المنطقة -->
            <small class="text-danger" *ngIf="currentMessageValidation.state">
              {{ currentMessageValidation.state }}
            </small>
          </div>
          <!-- أداة إدخال الرمز البريدي -->
          <div class="form-group">
            <label>Zip Code</label>
            <input
              formControlName="zipCode"
              [ngClass]="{
                'form-control': true,
                'is-invalid': currentMessageValidation.zipCode
              }"
              (blur)="loopThroughControls()"
            />
            <!-- جزء التحقق من الصحة لأداة إدخال الرمز البريدي -->
            <small class="text-danger" *ngIf="currentMessageValidation.zipCode">
              {{ currentMessageValidation.zipCode }}
            </small>
          </div>
        </fieldset>
      </div>

      <!-- بداية أدوات الأزرار -->
      <div class="card-footer">
        <button class="btn btn-primary" (click)="save()">Save</button>
        <button class="btn btn-primary ml-3" (click)="laodData()">
          Load Data
        </button>
      </div>
    </form>
  </div>
</div>

```

الآن لنرى التعديلات على النموذج:



3.4. التحقق من الصحة Custom Validation:

بعض الأحيان نحتاج ان نقوم نحن ببناء التحقق من الصحة الخاص بنا، وهذا النوع ليس له دوال جاهزة وانما logic يختلف من مبرمج إلى آخر على حسب خبرته او احتياجه، فبعض الأحيان نحتاج أن نقوم بكتابة هذا logic في ملف منفصل إذا كان هنالك احتمال ان نستخدمه في أكثر من component وبعض الأحيان نحتاج إلى أكثر من ملف إذا كان التحقق من الصحة متعدد وأحيان أخرى نكتبه بنفس component إذا كان المبرمج متأكد انه لا يحتاج إلى هذا logic في أماكن أخرى من المشروع.

وعلى الرغم من ان هذا النوع من التحقق من الصحة يرجع إلى رؤية المبرمج واحتياجه ولكن هنالك أسلوب معين قدمته لنا angular لتسهيل التعامل مع هذا النوع من التحقق من الصحة. وسوف اوضحها على شكل نقاط، كالتالي:

- ✓ الكود او logic لتحقق من الصحة يتم كتابته في دالة function.

- ✓ هذه الدالة تستقبل باراميتر واحد فقط من النوع AbstractControl وهذا الكلاس هو نفسه التي ترث منه الكلاسات FormGroup و FormControl ومن هذا المنطلق يصبح هذا الباراميتر يمتلك حق الوصول لنفس الدوال والخصائص التي لدى الكلاسات السابقة.
- ✓ هذه الدالة تُرجع لنا قيمتين القيمة الأولى كائن object تحتوي على key من النوع string والقيمة value لهذا key من النوع any أو boolean، والقيمة الثانية null في حالة عدم وجود خطأ والبيانات المدخلة في الأداة صحيحة.
- ✓ إذا تمت كتابة الدالة في ملف منفصل خارج component فيتم كتابتها داخل كلاس مع عمل export لهذا الكلاس لكي يمكن استدعاءه في أي component آخر، أو كتابة الدوال بشكل مباشر في الملف مع عمل export لها، وفي هذه الحالة لا يحتاج إلى جعل الدوال static كما في النقطة التالية.
- ✓ جعل الدالة static في حال كانت في ملف خارجي وداخل الكلاس لكي يمكن الوصول لها بشكل مباشر من خلال كتابة اسم هذا الكلاس دون الحاجة لعمل instance له.
- ✓ يتم كتابة الدالة في حال كانت بنفس component او الوصول لها عن طريق كتابة اسم الكلاس في نفس الموضع الذي نكتب في دوال التحقق من الصحة المبنية ضمناً، وبدون اقواس الدالة.
- ✓ يُفضل في حال الملف الخارجي أن يُكتب اسم الملف ومن ثم نقطة ومن ثم validator ومن ثم نقطة وبعدها ts.

أما من ناحية الأمثلة التي ممكن نحتاج فيها إلى بناء التحقق من الصحة، فمنها:

- منع المستخدم من إدخال بعض أسماء المستخدم كالاسم admin.
- منع المستخدم من كتابة المسافات او الرموز او أحرف غير إنجليزية في اسم المستخدم.
- التأكد من وجود لواحق نطاقات البريد الإلكتروني.
- التأكد من أن كلمة السر وإعادة إدخال كلمة السر نفس القيمة.

وسوف نقوم ببناء جميع الاحتمالات السابقة، وكما قلت سابقاً هذا عبارة عن logic وقد لا يكون أفضل الحلول، لذلك اعرف الطرق العامة لبناء custom validation ومن ثم حاول بناء logic الخاص بك.

اما من ناحية خطوات بناء التحقق من الصحة custom validation عملياً،

أولاً/ نقوم أولا بإنشاء ملف جديد باسم:

custom.validator.ts

ثانياً/ ثم في هذه الملف ننشأ كلاس باسم CustomValidators ونعمل له export:

```
export class CustomValidators {  }
```

ثالثاً/ وفي هذا الملف نعمل استدعاء للكلاس AbstractControl:

```
import { AbstractControl } from '@angular/forms';
```

وفي داخل الكلاس CustomValidators نقوم بإنشاء الدوال بحيث كل دالة تقوم بمهمة معينة من مهام التحقق من الصحة، مع جعل كل دالة static، كالتالي:

1.3.4. دالة منع المستخدم من إدخال بعض الأسماء في حقل اسم المستخدم كالاسم admin:

كما قلنا سابقاً أن هذا النوع من الدوال يستقبل باراميتر واحد من النوع AbstractControl ويعيد كائن أما الـ key للكائن من النوع string والقيمة له any او Boolean في حال وجود خطأ اما في حال عدم وجود خطأ من المستخدم فتعيد الدالة null، لذلك سوف أقوم بأنشاء الدالة في الكلاس السابق وليكن أسمها forbiddenNames، كالتالي:

ملف custom.validator.ts

```
1. import { AbstractControl } from '@angular/forms';
2.
3. export class CustomValidator {
4.
5.     static forbiddenNames(control: AbstractControl): { [key: string]: boolean } | null {
6.
7.      }
8.
9. }
10.
```

كما نلاحظ تم إنشاء دالة باسم forbiddenNames من النوع static وتحتوي على باراميتر باسم control من النوع AbstractControl مع استدعاء هذا الكلاس في السطر الأول، والدالة نفسها تُعيد قيمتين يا كائن object او null.

وفي هذه الدالة سوف نتحقق من قيمة control هل تساوي admin ام لا إذا تساويه فمعنا هذا ان المستخدم قام بإدخال القيمة admin وهو ممنوع إدخاله وفي هذه الحالة نُعيد كائن object والkey للكائن سوف اجعل اسمه بنفس اسم الدالة – مع العلم ان لك حرية اختيار أي اسم تريده لأن هذا key هو الذي سوف نستخدمه في رسائل التحقق من الصحة في الكائن massageValidation الذي شرحناه سابقاً – وقيمة هذا key سوف تكون true اما إذا لم تساوي فسوف نُعيد null بمعنى لا يوجد أخطاء والقيمة صحيحة، كالتالي:

ملف custom.validator.ts

```
1. import { AbstractControl } from '@angular/forms';
2.
3. export class CustomValidator {
4.
5.     static forbiddenNames(control: AbstractControl): { [key: string]: boolean } | null {
6.         return control.value === 'admin' ? { 'forbiddenNames': true } : null;
7.     }
8.
9. }
10.
```

الكود الآن يعمل ولا يوجد به مشاكل ولكن لنفترض أننا نُريد أن تكون الأسماء الممنوعة ديناميكية بمعنى أن هذه الأسماء الممنوعة تكون موجودة في قاعدة بيانات مثلاً ومدير النظام يقوم بإدخال ما يشاء من الأسماء في نموذج خاص به بحيث يمنع مستخدم النظام من تسجيل هذه الأسماء، في هذه الحالة لابد أن نقوم بتعديل logic في هذه الدالة ولكن تواجهنا

مشكلة وهي أن هذه الدالة لا تقبل إلا باراميتر واحد وهو الأداة التي نُريد التحقق من صحتها لذلك لابد من عمل خُدعة بسيطة عن طريق Closure وهي طريقة برمجية في الجافا سكريبت تسمح بوجود دالة داخل دالة بحيث أن الدالة الرئيسية تُعيد الدالة الفرعية والدالة الفرعية تكون بدون اسم او ما يُسمى anonymous function مع استخدام ميزة arrow function للدالة الفرعية، بحيث أن الدالة الرئيسية تستقبل أي عدد نريده من الباراميترات والدالة الفرعية تمتلك حق الوصول والتعامل مع هذه الباراميترات، وهذا الذي نريده ويخدمنا في logic الخاص بنا حيث أن الدالة الرئيسية نُريد منها أن تستقبل مصفوفة نصية تحتوي على قائمة الأسماء وفي الدالة الفرعية التي تكون بدون اسم نتحقق هل قيمة الأداة موجودة في هذه المصفوفة أو لا إذا كانت موجودة فسوف تُعيد الكائن وأن لم تكن موجودة فسوف تُعيد null، ويُفضل أن تُعيد الدالة الرئيسية ValidatorFn، كالتالي:

ملف custom.validator.ts

```
1. import { AbstractControl, ValidatorFn } from '@angular/forms';
2.
3. export class CustomValidator {
4.
5.     static forbiddenNames(names: string[]): ValidatorFn {
6.         return (control: AbstractControl): { [key: string]: boolean } | null => {
7.             return names.includes(control.value) ? { 'forbiddenNames': true } : null;
8.         };
9.     }
10.
11.}
12.
```

نلاحظ أن الكود logic الآن أصبح أكثر ديناميكية حيث يستقبل مصفوفة نصية بأي اسم ويتحقق من إذا قيمة الأداة موجودة من ضمن قيم المصفوفة أو لا.

2.3.4. دالة منع المستخدم من كتابة المسافات او الرموز الخاصة او الأرقام او حروف غير اللغة الإنجليزية في حقل اسم المستخدم:

سوف نتبع القواعد التي ذكرناها فما سبق في كتابة الدالة واما logic فسيكون كالتالي:

جزء من ملف custom.validator.ts

```
1. static isEnglishLetters(control: AbstractControl): { [key: string]: boolean } | null {
2.     const EnglishLetters = /^[A-Za-z]+$/.test(control.value);
3.     if (control.hasError('pattern') && (control.value as string).length < 3) {
4.         return null;
5.     } else if (!EnglishLetters && (control.value as string).length >= 3) {
6.         return { 'isEnglishLetters': true };
7.     }
8.     return null;
9. }
10.
```

السطر 2 (يعمل اختبار لقيمة الأداة هل هي حروف فقط أو لا ويخزن القيمة true أو false في الثابت EnglishLetters بحيث يكون قيمة المتغير true إذا كانت القيمة في الأداة حروف إنجليزية فقط).

السطر 3 و4 (نتأكد من أن التحقق من الصحة pattern مفعل فإذا كان مفعل فلا نحتاج أن نتأكد من القيمة انها أحرف إنجليزية لذلك سوف نُرجع null لأن هنالك خطأ اصلاً وهو عدد الخانات اقل من ثلاث).

السطر 5 و6 (في حال أن pattern غير مفعل بمعنى أن عدد الخانات أكثر من ثلاث – مع ملاحظة أن يبدأ عد الخانات من الصفر – في هذه الحالة نبدأ نخضع قيمة الأداة لاختبار التحقق من الصحة الثانية وذلك من خلال معرفة قيمة المتغير في حال كان false من خلال علامة (!) قبل المتغير وبنفس الوقت عدد الخانات أكثر من ثلاث، في هذه الحال نُرجع object).

السطر 8 (في حال أن القيمة اجتازت جميع الاختبارات فمعنى هذا أن القيمة صحيحة فنرجع null.

3.3.4. دالة التأكد من وجود لواحق نطاقات البريد الإلكتروني:

دالة التحقق من الصحة المبنية ضمناً الخاصة بالتحقق من صحة صيغة البريد الإلكتروني Validators.email يوجد بها قصور، فلو قمنا بكتابة البريد الإلكتروني بهذه الصيغة test@test فسوف تعتبرها صحيحة، لذلك سوف نقوم بإعادة التحقق من صحة صياغة البريد الإلكتروني من خلال logic يقبل فقط صيغ البريد الإلكتروني التالية:

(yourname) @ (domain) . (extension) (.again)

أو

(yourname) @ (domain) . (extension)

مثال/ test@test.com – test@test.org.sa

حيث أن: (extension) يقبل فقط من خانتين إلى أربع خانات فقط.

و (again.) يقبل ايضاً من اربع إلى ثلاث خانات فقط

أما logic الدالة فسيكون كالتالي:

جزء من ملف custom.validator.ts

```
1. static emailValidation(control: AbstractControl): { [key: string]: boolean } | null {
2.     const regEx = /^([a-z\d\.-_]+)@([a-z\d-_]+)\.([a-z]{2,4})(\.[a-z]{2,4})?$/;
3.     const email = control.value;
4.     const emailValid = regEx.test(email);
5.     return (email === '' || emailValid) ? null : { 'emailValidation': true };
6. }
7.
```

السطر 1 (بداية الدالة واسمها emailValidation)

السطر 2 (متغير نُخزن فيه التعبير القياسي لتحقق من صحة البريد الإلكتروني وهو من النوع RegEx)

السطر 3 (نُخزن قيمة الأداة في متغير)

السطر 4 (نختبر قيمة الأداة من خلال التعبير القياسي ونُخزن نتيجة الاختبار في متغير والنتيجة تكون true في حال نجاح الاختبار أي أن قيمة الأداة مطابقة لشروط التحقق من الصحة وfalse في حال الفشل)

السطر 5 (نُرجع قيم الدالة من خلال شرطين إذا كانت قيمة المتغير فارغة أو قيمة المتغير emailValid تساوي true نُرجع null او نُرجع الكائن object).

4.3.4. دالة التأكد من أن كلمة السر مطابقة لإعادة أدخال كلمة السر:

حقيقة هنالك عدة طرق لتعامل مع هذا النوع من التحقق من الصحة، وهنا سوف استخدم طريقة وهي تجميع حقلي كلمة المرور وإعادة إدخال كلمة المرور في نموذج فرعي FormGroup، حيث أن التحقق من الصحة يكون على مستوى النموذج الفرعي لكي نستطيع الوصول لقيم كلا الأداتين التي يحتويها، والدالة تكون بالشكل التالي:

جزء من ملف custom.validator.ts

```
1. static passwordValidation(formGroup: AbstractControl):{[key: string]: boolean} | null {
2.     const password = formGroup.get('password');
3.     const confirmPassword = formGroup.get('confirmPassword');
4.     if (password && confirmPassword &&
5.         password.value !== confirmPassword.value &&
6.         (confirmPassword.dirty || confirmPassword.touched)) {
7.         return { 'passwordValidation': true };
8.     } else {
9.         return null;
10.    }
11.}
12.
```

السطر 1 (بداية الدالة ومررنا لها باراميتر اسميناه formGroup لكي يدل على ان هذا الباراميتر يشير إلى النموذج الفرعي)

السطر 2 و3 (الحصول على قيم الأداتين في النموذج الفرعي وتخزين هذه القيم في متغيرات)

السطر من 4 إلى 7 (وضع شروط التحقق أن هنالك قيم في الأداتين والقيم غير متساويتان وتم التعديل أو لمس أداة إعادة كتابة كلمة السر، فإذا تحققت هذه الشروط فسوف نُرجع الكائن object)

السطر 9 (نُرجع null في حال عدم تحقق الشروط السابقة والذي يعني أن القيمتين متساويتين)

الآن لنرى ملف custom.validators.ts بشكله النهائي:

ملف custom.validator.ts

```
1. import { AbstractControl, ValidatorFn } from '@angular/forms';
2.
3. export class CustomValidator {
4.
5.     static forbiddenNames(names: string[]): ValidatorFn {
6.         return (control: AbstractControl): { [key: string]: boolean } | null => {
7.             return names.includes(control.value) ? { 'forbiddenNames': true } : null;
8.         };
9.     }
```

```
10.
11.    static isEnglishLetters(control: AbstractControl): {[key: string]: boolean} | null {
12.        const EnglishLetters = /^[A-Za-z]+$/.test(control.value);
13.        if (control.hasError('pattern') && (control.value as string).length < 3) {
14.            return null;
15.        } else if (!EnglishLetters && (control.value as string).length >= 3) {
16.            return { 'isEnglishLetters': true };
17.        }
18.        return null;
19.     }
20.
21.    static emailValidation(control: AbstractControl): {[key: string]: boolean} | null {
22.        const regEx = /^([a-z\d\.-_]+)@([a-z\d-_]+)\.([a-z]{2,4})(\.[a-z]{2,4})?$/;
23.        const email = control.value;
24.        const emailValid = regEx.test(email);
25.        return (email === '' || emailValid) ? null : { 'emailValidation': true };
26.     }
27.
28.    static passwordValidation(formGroup: AbstractControl):{[key: string]: boolean}|null{
29.        const password = formGroup.get('password');
30.        const confirmPassword = formGroup.get('confirmPassword');
31.        if (password && confirmPassword &&
32.            password.value !== confirmPassword.value &&
33.            (confirmPassword.dirty || confirmPassword.touched)) {
34.            return { 'passwordValidation': true };
35.        } else {
36.            return null;
37.        }
38.     }
39.
40.}
41.
```

وأخيراً يجب أن ننوه على الذي ذكرناه في البداية أن هذا النوع من التحقق من الصحة ليس له قواعد ثابتة وأنما هو logic أو فكر برمجي يختلف من مطور إلى آخر على حسب احتياج هذا المطور وخبرته، وقد تكون أنت عزيزي القارئ تستطيع كتابة هذه الأكواد بطريقة مختصرة وبسيطة عما قمت بكتابته أنا، ولكن يجب عليك فقط معرفة الخطوط العريضة من ناحية أن هذه الدوال لا تقبل إلا باراميتر واحد وهو من النوع AbstractControl وهذا النوع يرث منه كلا الكلاسين FormGroup و FormControl أي بمعنى أن هذا الباراميتر يمتلك نفس الخصائص والدوال التي يمتلكها هذين الكلاسين وهذا ينفعنا بإن يُتاح لنا حرية التحقق من الصحة سواء على مستوى الأداة نفسها كما فعلنا في أغلب الدوال السابقة أو على مستوى النموذج الرئيسي أو النموذج الفرعي كما فعلنا في آخر دالة passwordValidation، وكيفية تمرير أكثر من باراميتر لدالة من خلال Closure وهو وجود دالة داخل دالة، أو كيفية كتابة هذه الدوال في كلاس منفصل في ملف خارجي...الخ، وغيره من الخطوط العريضة الأخرى.

5.3.4. تطبيق الدوال التحقق من الصحة على ملف app.component.ts:

بعد استكمال جميع الدوال السابقة بقي أن نقوم باستدعاء وتطبيق هذه الدوال مع العلم أنه يمكن استدعائها وتنفيذها في أي component نريده لأن هذا هو الهدف في وضعها في ملف منفصل، أما خطوات تطبيق وتنفيذ هذه الدوال في هذا الملف كالتالي:

أولاً: استدعاء الكلاس الموجود في الملف custom.validator.ts:

```
1.  import { CustomValidator } from 'src/app/shared/custom.validators';
```

ثانياً: إضافة دوال التحقق من الصحة لحقول النموذج الذي تم إنشائه برمجياً، مع ملاحظة تجميع أداتي كلمة السر وإعادة كلمة السر في نموذج فرعي باسم passwordGroup وإضافة دالة التحقق من الصحة passwordValidation لهذا النموذج الفرعي وليس للأدوات الموجودة داخله، كالتالي:

```
1.  ngOnInit() {
2.    this.form = this.fb.group({
3.      userName: [null,
4.        [
5.          Validators.required,
6.          Validators.pattern('.{3,}'),
7.          CustomValidator.forbiddenNames(this.names),
8.          CustomValidator.isEnglishLetters
9.        ]
10.     ],
11.     email: [null,
12.       [
13.         Validators.required,
14.         CustomValidator.emailValidation
15.       ]
16.     ],
17.     passwordGroup: this.fb.group({
18.       password: [null,
19.         [
20.           Validators.required,
21.           Validators.pattern('(?=.*[A-Za-z])(?=.*[A-Z])(?!.* )(?=.*[0-9])[A-Za-z0-9d$@].{5,}')
22.         ]
23.       ],
24.       confirmPassword: [null,
25.         [
26.           Validators.required
27.         ]
28.       ]
29.     }, { validator: CustomValidator.passwordValidation }),
30.     gender: [null, Validators.required],
31.     address: this.fb.group({
32.       city: [null, Validators.required],
33.       state: [null, Validators.required],
34.       zipCode: [null, [Validators.required, Validators.pattern('^[0-9]{5}$')]]
35.     })
36.   });
37. }
```

دوال التحقق من الصحة لأسم المستخدم مع ملاحظة أننا قمنا بتمرير مصفوفة تحتوي على الأسماء الممنوعة لدالة forbiddenNames كبارامتر

دالة التحقق من الصحة للبريد الإلكتروني مع ملاحظة أننا قمنا باستبدالها بدلاً من دالة التحقق من الصحة المبنية ضمناً email

النموذج الفرعي الجديد ويحتوي بداخله على الأداتين

دالة التحقق من الصحة وأضفناها على مستوى النموذج الفرعي وليس على مستوى الأداتين

ثالثاً: تعريف مصفوفة نصية تحتوي على الأسماء التي نريد منع المستخدم من التسجيل بها في حقل اسم المستخدم لكي نمررها لدالة forbiddenNames كما فعلنا في الخطوة رقم ثانياً، كالتالي:

```
1.  names: string[] = ['admin', 'administrator'];
```

رابعاً: كتابة رسائل الخطأ في الكائن messageValidation مع ملاحظة إضافة key جديد لهذا الكائن يشير إلى النموذج الفرعي passwordGroup بنفس اسم هذا النموذج ويحتوي هو ايضاً على key اسم هذا key مشابه لـkey الذي تعيده الدالة passwordValidation وقيمة هذا key هي رسالة الخطأ، كالتالي:

```
messageValidation = {
  'userName': {
    'required': 'اسم المستخدم مطلوب',
    'pattern': 'اسم المستخدم على الاقل ثلاث خانات',
    'forbiddenNames': 'لايُسمح بتسجيل اسم المستخدم المُدخل',
    'isEnglishLetters': 'لا يُسمح بوجود المسافات او الارقام او الرموز الخاصة او حروف غير الإنجليزية'
  },
  'email': {
    'required': ' البريد الإلكتروني مطلوب',
    'emailValidation': 'صيغة البريد الإلكتروني غير صحيحة'
  },
  'passwordGroup': {
    'passwordValidation': 'كلمة السر غير متطابقة'
  },
  'password': {
    'required': 'كلمة االسر مطلوبة',
    'pattern': ' كلمة السر ست خانات ارقام وحروف وعلى الاقل حرف واحد كبير'
  },
  'confirmPassword': {
    'required': ' الحقل مطلوب'
  },
  'gender': {
    'required': 'الحقل مطلوب'
  },
  'city': {
    'required': 'حقل اسم المدينة مطلوب'
  },
  'state': {
    'required': 'حقل المنطقة مطلوب'
  },
  'zipCode': {
    'required': 'حقل الرمز البريدي مطلوب',
    'pattern': 'الرمز البريدي لابد أن يكون قيمة رقمية من خمس خانات'
  }
};
```

خامساً: إضافة key للكائن currentMessageValidation يشير إلى النموذج الفرعي passwordGroup، كالتالي:

```
currentMessageValidation = {
  'userName': '',
  'email': '',
  'passwordGroup': '',
  'password': '',
  'confirmPassword': '',
  'gender': '',
  'city': '',
  'state': '',
  'zipCode': ''
};
```

سادساً: واخيراً سوف نجري بعض التعديلات على الدالة loopThroughControls الخاصة بإظهار الرسائل، وسبب لأننا قمنا بإضافة دالة التحقق من الصحة على مستوى النموذج الفرعي لذلك نريد منه أن يقوم بعمل loop ليس فقط على مستوى الأدوات وانما على مستوى النماذج الفرعية والتحقق ومن ثم إضافة رسالة الخطأ وإظهارها، ونستطيع القيام بذلك من خلال فصل الـif إلى أثنين منفصلة وليست متصلة if و else if ومن ثم جعل الشرط يقوم بإظهار الرسائل في البداية، كالتالي:

```
1.  loopThroughControls(formGroup: FormGroup = this.form) {
2.    Object.keys(formGroup.controls).forEach((key: string) => {
3.      const controlName = formGroup.get(key);
4.      this.currentMessageValidation[key] = '';
5.      if (controlName && controlName.invalid && controlName.touched) {
6.        const messages = this.messageValidation[key];
7.        for (const controlError in controlName.errors) {
8.          if (controlError) {
9.            this.currentMessageValidation[key] +=
10.             messages[controlError] + ' ';
11.         }
12.       }
13.     }
14.     if (controlName instanceof FormGroup) {
15.       this.loopThroughControls(controlName);
16.     }
17.   });
18. }
```

راجع الأسطر من 5 إلى 16

الآن لنرى ملف app.component.ts بعد أجراء جميع هذه التعديلات عليه:

ملف app.component.ts

```
1. import { Component, OnInit } from '@angular/core';
2. import { FormGroup, FormBuilder, Validators } from '@angular/forms';
3. import { CustomValidator } from 'src/app/shared/custom.validators';
4.
5. @Component({
6.     selector: 'app-singup',
7.     templateUrl: './singup.component.html',
8.     styleUrls: ['./singup.component.css']
9. })
10.
11.export class SingupComponent implements OnInit {
12.     states: string[] = ['ALRiyadh', 'Makkah', 'AlSharqiyah', 'AlQasim'];
13.     form: FormGroup;
14.     userNameLength: any = '0';
15.     names: string[] = ['admin', 'administrator'];
16.
17.     messageValidation = {
18.         'userName': {
19.             'required': 'اسم المستخدم مطلوب',
20.             'pattern': 'اسم المستخدم على الاقل ثلاث خانات',
21.             'forbiddenNames': 'لايُسمح بتسجيل اسم المستخدم المُدخل',
22.             'isEnglishLetters': 'لا يُسمح بوجود المسافات او الارقام او الرموز الخاصة او حروف غير الإنجليزية'
23.         },
24.         'email': {
25.             'required': ' البريد الإلكتروني مطلوب',
26.             'emailValidation': 'صيغة البريد الإلكتروني غير صحيحة'
27.         },
28.         'passwordGroup': {
29.             'passwordValidation': 'كلمة السر غير متطابقة'
30.         },
31.         'password': {
32.             'required': 'كلمة السر مطلوبة',
33.             'pattern': ' كلمة السر ست خانات ارقام وحروف وعلى الاقل حرف واحد كبير'
```

```
        },
        'confirmPassword': {
            'required': ' الحقل مطلوب'
        },
        'gender': {
            'required': 'الحقل مطلوب'
        },
        'city': {
            'required': 'حقل اسم المدينة مطلوب'
        },
        'state': {
            'required': 'حقل المنطقة مطلوب'
        },
        'zipCode': {
            'required': 'حقل الرمز البريدي مطلوب',
            'pattern': 'الرمز البريدي لابد أن يكون قيمة رقمية من خمس خانات'
        }
    };

    currentMessageValidation = {
        'userName': '',
        'email': '',
        'passwordGroup': '',
        'password': '',
        'confirmPassword': '',
        'gender': '',
        'city': '',
        'state': '',
        'zipCode': ''
    };
    constructor(private fb: FormBuilder) { }

    ngOnInit() {
        this.form = this.fb.group({
            userName: [null,
                [
                    Validators.required,
                    Validators.pattern('.{3,}'),
                    CustomValidator.forbiddenNames(this.names),
                    CustomValidator.isEnglishLetters
                ]
            ],
            email: [null,
                [
                    Validators.required,
                    CustomValidator.emailValidation
                ]
            ],
            passwordGroup: this.fb.group({
                password: [null,
                    [
                        Validators.required,
                        Validators.pattern(
```

```
                            '(?=.*[A-Za-z])(?=.*[A-Z])(?!.* )(?=.*[0-9])[A-Za-z0-
9d$@].{5,}'
                        )
                    ]
                ],
                confirmPassword: [null,
                    [
                        Validators.required
                    ]
                ]
            }, { validator: CustomValidator.passwordValidation }),
            gender: [null, Validators.required],
            address: this.fb.group({
                city: [null, Validators.required],
                state: [null, Validators.required],
                zipCode: [null,
                            [
                              Validators.required,
                              Validators.pattern('^[0-9]{5}$')
                            ]
                          ]
            })
        });
        this.userName.valueChanges.subscribe((value: string) => {
            this.userNameLength = value.length;
        });
        this.form.valueChanges.subscribe(data => {
            this.loopThroughControls(this.form);
        });
    }

    save() {
        console.log(this.form);
    }

    loopThroughControls(formGroup: FormGroup = this.form) {
        Object.keys(formGroup.controls).forEach((key: string) => {
            const controlName = formGroup.get(key);
            this.currentMessageValidation[key] = '';
            if (controlName && controlName.invalid && controlName.touched) {
                const messages = this.messageValidation[key];
                for (const controlError in controlName.errors) {
                    if (controlError) {
                        this.currentMessageValidation[key] +=
                            messages[controlError] + ' ';
                    }
                }
            }
            if (controlName instanceof FormGroup) {
                this.loopThroughControls(controlName);
            }
        });
    }
```

```
139.
140.       laodData() {
141.           this.form.patchValue({
142.               userName: 'DivFaisal',
143.               email: 'test@test.com',
144.               passwordGroup: {
145.                   password: 'Aa1111',
146.                   confirmPassword: 'Aa1111'
147.               },
148.               gender: 'male',
149.               address: {
150.                   city: 'Riyadh',
151.                   state: 'ALRiyadh',
152.                   zipCode: '87678'
153.               }
154.           });
155.       }
156.
157.       get userName() {
158.           return this.form.get('userName');
159.       }
160.       get email() {
161.           return this.form.get('email');
162.       }
163.       get password() {
164.           return this.form.get('password');
165.       }
166.       get confirmPassword() {
167.           return this.form.get('confirmPassword');
168.       }
169.       get gender() {
170.           return this.form.get('gender');
171.       }
172.       get address() {
173.           return this.form.get('address');
174.       }
175.       get city() {
176.           return this.form.get('address').get('city');
177.       }
178.       get state() {
179.           return this.form.get('address').get('state');
180.       }
181.       get zipCode() {
182.           return this.form.get('address').get('zipCode');
183.       }
184.   }
185.
```

6.3.4. التعديلات على ملف app.component.html او ما يسمى ملف template:

سوف نجري بعض التعديلات البسيطة على هذا الملف والسبب أن الأداة confirmPassword يوجد لها نوعين من التحقق من الصحة وكل نوع في key مختلف داخل الكائن currentMessageValidation وهما passwordGroup و confirmPassword لذلك لابد من إضافة الـkey الجديد passwordGroup لإظهار كلاسات البوتستراب في حال تحقق وهنالك خطأ، وبنفس الوقت لكي لا تتداخل رسائل الخطأ في حال أن كلا النوعين تحققا وارجعا القيمة true بمعنى ان القيمة في أداة أعادة إدخال كلمة المرور تم لمسها وخالية وبنفس الوقت لا تتساوى مع أداة كلمة السر لذلك لابد أن نضع شرط في حال أن الـkey confirmPassword –الخاص بتحقق أن القيمة ليست خالية – يُرجع القيمة true اظهر محتوى رسالته وفي حال false اظهر محتوى رسالة passwordGroup (القيم غير متساوية)، كالتالي:

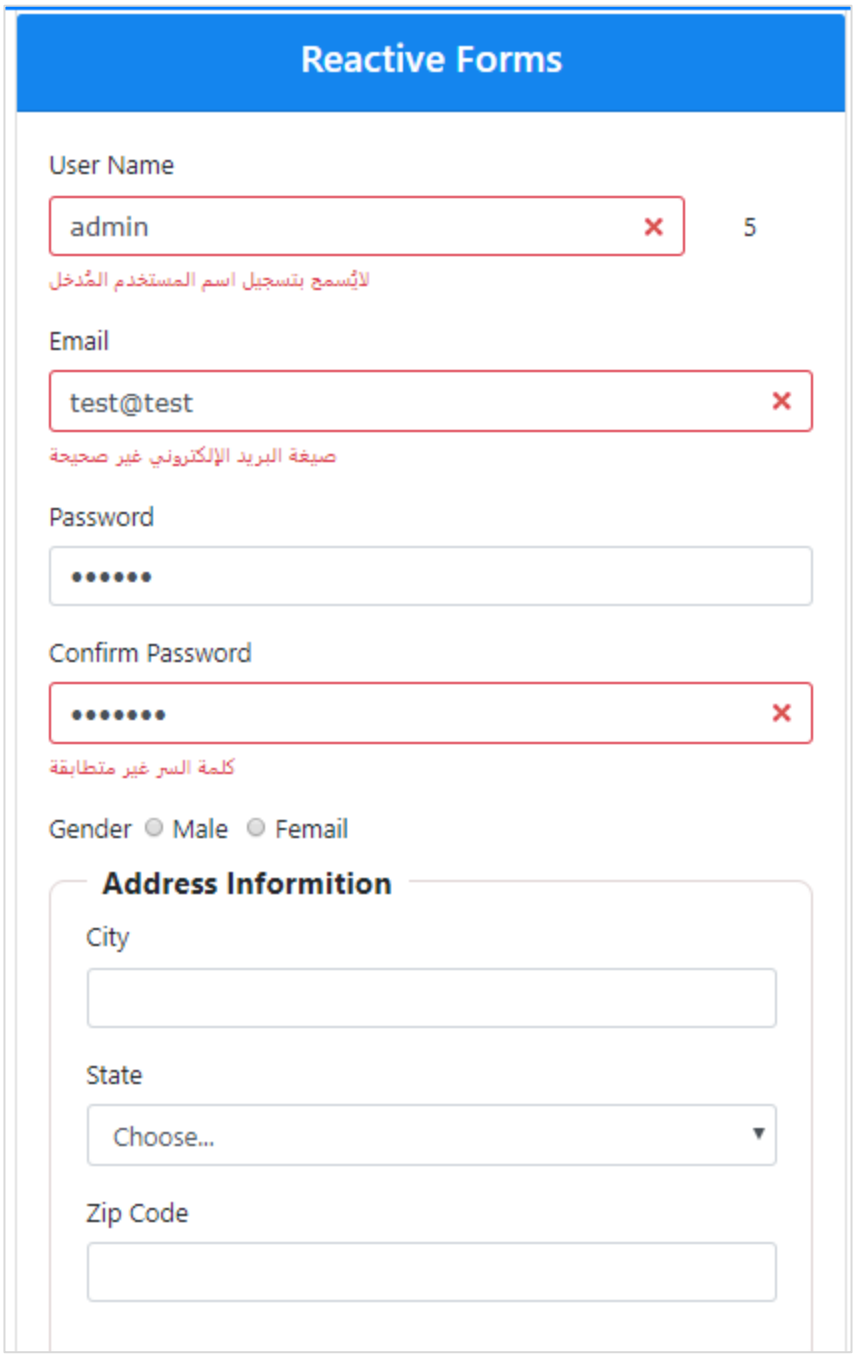
```
1.  <!-- أداة إعادة إدخال كلمة السر -->
2.  <div class="form-group">
3.    <label>Confirm Password</label>
4.    <input
5.      type="password"
6.      formControlName="confirmPassword"
7.      [ngClass]="{
8.        'form-control': true,
9.        'is-invalid': currentMessageValidation.confirmPassword || currentMessageValidation.passwordGroup}"
10.     (blur)="loopThroughControls()"/>
11.
12.   <!-- جزء التحقق من الصحة الخاص بأداة إعادة إدخال كلمة السر -->
13.   <small class="text-danger"
14.     *ngIf="currentMessageValidation.confirmPassword || currentMessageValidation.passwordGroup">
15.     {{currentMessageValidation.confirmPassword ?
16.       currentMessageValidation.confirmPassword : currentMessageValidation.passwordGroup}}
17.   </small>
18. </div>
```

الشرط الجديد لإظهار وإخفار كلاسات البوتسراب في حال الخطأ

الشرط الجديد لإظهار وإخفار رسائل الخطأ

التبديل في عرض محتوى رسائل الخطأ وفق الشروط

الآن لنرى التعديلات التي قمنا بها على النموذج الخاص بنا على المتصفح:



كما نرى النموذج يعمل بدون أي مشاكل.

وبذلك نكون أنهينا النوع الثاني من أنواع التحقق من الصحة custom validation وسوف نتكلم بإذن الله بعده على النوع الثالث وهو التحقق من الصحة المشروط او الشرطي Conditional Validation.

4.4. التحقق من الصحة المشروط Conditional Validation:

فكرة هذا التحقق مرتبط بشرط معين اثناء وقت التشغيل فإذا تحقق هذا الشرط يتحقق من الصحة وإذا لم يتحقق هذا الشرط فلن يقوم بالتحقق من الصحة، وله صور عديدة فمنها على سبيل المثال لا الحصر أن يحدد المستخدم وسيلة الاتصال الرئيسية هل هي البريد الإلكتروني أم الجوال عن طريق أداة radio فإذا حدد البريد يصبح التحقق من حقل البريد فقط وإذا حدد الجوال يتم التحقق من صحة حقل الجوال والبريد لا يتحقق منه، ومنها ايضاً وضع شرط للعنوان فإذا كان العنوان مؤقت تظهر له أداة تحديد تاريخ انتهاء السكن ويتم التحقق من الصحة منها في حال المستخدم لم يدخل التاريخ اما إذا حدد المستخدم بإن التاريخ دائم فيتم الغاء التحقق من الصحة واخفاء هذه الأداة، وكما قلنا صور هذا النوع من التحقق من الصحة متعددة وليس هنا المقام لحصرها.

أما الصورة التي سوف اعتمدها هنا هي الصورة الثانية لأنها تناسب النموذج الذي نعمل عليه، لذلك سوف نضيف في النموذج الفرعي الخاص بإدخال بيانات العنوان أداتين radio لتحديد نوع العنوان دائم او مؤقت وأداة لإدخال تاريخ انتهاء العنوان في حال كان مؤقت.

ملاحظة مهمة: هذا النوع هو ليس تحقق من الصحة بحد ذاته وانما هو في الحقيقة آلية توضح متى وكيف يتحقق من الصحة أما كود التحقق من الصحة فقد يكون Built-In Validation او Custom Validation او كلاهما معاً.

وللقيام بهذا الأمر سوف نكرر ما نقوم به دائماً، من حيث أضافة الأسماء البرمجية للأدوات السابقة في كود انشاء النموذج برمجياً في ملف app.component.ts:

app.component.ts جزء من ملف

```
1. ngOnInit() {
2.     this.form = this.fb.group({
3.         userName: [null,
4.             [
5.                 Validators.required,
6.                 Validators.pattern('.{3,}'),
7.                 CustomValidator.forbiddenNames(this.names),
8.                 CustomValidator.isEnglishLetters
9.             ]
10.         ],
11.         email: [null,
12.             [
13.                 Validators.required,
14.                 CustomValidator.emailValidation
15.             ]
16.         ],
17.         passwordGroup: this.fb.group({
18.             password: [null,
19.                 [
20.                     Validators.required,
21.                     Validators.pattern(
22.                         '(?=.* [A - Za - z])(?=.* [A - Z])(?!.* )(?=.* [0 - 9])[A - Za
   - z0 - 9d$@].{ 5,}'
23.                         )
```

```
                ]
            ],
            confirmPassword: [null,
                [
                    Validators.required
                ]
            ]
        }, { validator: CustomValidator.passwordValidation }),
        gender: [null, Validators.required],
        address: this.fb.group({
            addressType: ['permanent'],
            addressDate: [null],
            city: [null, Validators.required],
            state: [null, Validators.required],
            zipCode: [null, [Validators.required, Validators.pattern('^[0-9]{5}$')]]
        })
    });
}
```

راجع السطر 34 (الاسم البرمجي والذي يشير إلى أداتي radio واعطيناه قيمة افتراضية بمعنى أن radio ذات القيمة permanent هي المختارة بشكل افتراضي، وبذلك لا نحتاج أن نتحقق من الصحة هنا لأن هنالك قيمة مختارة بشكل افتراضي)

راجع السطر 35 (الاسم البرمجي لأداة التاريخ Date ولم نعطيه أي قيمة افتراضية null وبنفس الوقت لم نضع له أي نوع من التحقق من الصحة لأننا نريد أن نضع التحقق من الصحة ونزيله وفق شروط معينة)

وبنفس الوقت نقوم بإنشاء getter للأسماء البرمجية للأدوات المضافة عن طريق الدالة get:

```
get addressType() {
  return this.form.get('address').get('addressType');
}
get addressDate() {
  return this.form.get('address').get('addressDate');
}
```

أما نوع التحقق من الصحة للأداة addressDate (أداة التاريخ) فسوف يكون من أنواع التحقق من الصحة Built-In وهو required، لذلك لنضيف رسالة الخطأ إلى الكائن الخاص برسائل الخطأ massageValidation، كالتالي:

app.component.ts جزء من ملف

```
messageValidation = {
    'userName': {
        'required': 'اسم المستخدم مطلوب',
        'pattern': 'اسم المستخدم على الاقل ثلاث خانات',
        'forbiddenNames': 'لايُسمح بتسجيل اسم المستخدم المُدخل',
        'isEnglishLetters': 'لا يُسمح بوجود المسافات او الارقام او الرموز الخاصة او حروف غير الإنجليزية'
    },
    'email': {
        'required': ' البريد الإلكتروني مطلوب',
        'emailValidation': 'صيغة البريد الإلكتروني غير صحيحة'
    },
```

```
12.     'passwordGroup': {
13.         'passwordValidation': 'كلمة السر غير متطابقة'
14.     },
15.     'password': {
16.         'required': 'كلمة االسر مطلوبة',
17.         'pattern': ' كلمة السر ست خانات ارقام وحروف وعلى الاقل حرف واحد كبير'
18.     },
19.     'confirmPassword': {
20.         'required': ' الحقل مطلوب'
21.     },
22.     'gender': {
23.         'required': 'الحقل مطلوب'
24.     },
25.     'addressDate': {
26.         'required': 'حقل تاريخ انتهاء أقامه السكن مطلوب'
27.     },
28.     'city': {
29.         'required': 'حقل اسم المدينة مطلوب'
30.     },
31.     'state': {
32.         'required': 'حقل المنطقة مطلوب'
33.     },
34.     'zipCode': {
35.         'required': 'حقل الرمز البريدي مطلوب',
36.         'pattern': 'الرمز البريدي لابد أن يكون قيمة رقمية من خمس خانات'
37.     }
38. };
39.
```

راجع الاسطر (من 25 إلى 27)

وكما جرة العادة ايضاً نضيف key مشابهه للاسم البرمجي في الكائن currentMessageValidation:

جزء من ملف app.component.ts

```
1. currentMessageValidation = {
2.     'userName': '',
3.     'email': '',
4.     'passwordGroup': '',
5.     'password': '',
6.     'confirmPassword': '',
7.     'gender': '',
8.     'addressDate': '',
9.     'city': '',
10.    'state': '',
11.    'zipCode': ''
12. };
13.
```

اما في ملف app.component.html نضيف Markup او كود html التالي في الجزء الخاص بنموذج الفرعي:

app.component.html ملف

```
1.  <!-- بداية النموذج الفرعي -->
2.  <fieldset class="scheduler-border" formGroupName="address">
3.    <legend class="scheduler-border">Address Information</legend>
4.
5.    <!--date أداة التاريخ radio أداتي-->
6.    <label class="pr-2" style="margin-bottom: 0px">Address Type:</label>
7.    <br />
8.    <div class="form-check form-check-inline">
9.      <input
10.       class="form-check-input"
11.       type="radio"
12.       formControlName="addressType"
13.       id="temporary"
14.       value="temporary"
15.     />
16.     <label class="form-check-label" for="temporary">Temporary</label>
17.   </div>
18.   <div class="form-check form-check-inline">
19.     <input
20.       class="form-check-input"
21.       type="radio"
22.       formControlName="addressType"
23.       id="permanent"
24.       value="permanent"
25.     />
26.     <label class="form-check-label" for="permanent">Permanent</label>
27.   </div>
28.   <input type="date" formControlName="addressDate" />
29.
30.   <!--date جزء التحقق من الصحة الخاص بأداة التاريخ-->
31.   <div>
32.     <small class="text-danger"> </small>
33.   </div>
34.
35.   <!-- أداة إدخال اسم المدينة -->
36.   <div class="form-group pt-4">
37.     <label>City</label>
38.     <input
39.       formControlName="city"
40.       [ngClass]="{
41.                 'form-control': true,
42.                 'is-invalid': currentMessageValidation.city
43.               }"
44.       (blur)="loopThroughControls()"
45.     />
46.     <!-- جزء التحقق من الصحة لأداة إدخال اسم المدينة -->
47.     <small class="text-danger" *ngIf="currentMessageValidation.city">
48.       {{ currentMessageValidation.city }}
49.     </small>
```

```
50.   </div>
51.   <!-- أداة اختيار اسم المنطقة او الولاية -->
52.   <div class="form-group">
53.     <label>State</label>
54.     <select
55.       formControlName="state"
56.       [ngClass]="{
57.                 'form-control': true,
58.                 'is-invalid': currentMessageValidation.state
59.               }"
60.       (blur)="loopThroughControls()"
61.     >
62.       <option selected [ngValue]="null">Choose...</option>
63.       <option *ngFor="let item of states" [value]="item">{{ item }}</option>
64.     </select>
65.     <!-- جزء التحقق من الصحة لأداة أختيار اسم المنطقة -->
66.     <small class="text-danger" *ngIf="currentMessageValidation.state">
67.       {{ currentMessageValidation.state }}
68.     </small>
69.   </div>
70.   <!-- أداة إدخال الرمز البريدي -->
71.   <div class="form-group">
72.     <label>Zip Code</label>
73.     <input
74.       formControlName="zipCode"
75.       [ngClass]="{
76.                 'form-control': true,
77.                 'is-invalid': currentMessageValidation.zipCode
78.               }"
79.       (blur)="loopThroughControls()"
80.     />
81.     <!-- جزء التحقق من الصحة لأداة إدخال الرمز البريدي -->
82.     <small class="text-danger" *ngIf="currentMessageValidation.zipCode">
83.       {{ currentMessageValidation.zipCode }}
84.     </small>
85.   </div>
86. </fieldset>
87.
```

راجع الاسطر (من 5 إلى 33) حيث تمثل بداية ونهاية Markup الخاص بأداتي radio وأداة التاريخ

راجع الاسطر (12 – 22) حيث أجرينا ربط الاسم البرمجي بأداتي radio الأولى والثانية مع ملاحظة أن كلا الأداتين لهما نفس الاسم البرمجي

راجع الاسطر (14 – 24) حيث قمنا بإسناد القيم للخاصية value لأداتي Radio

راجع السطر 28 حيث اضفنا أداة التاريخ وربطنا الاسم البرمجي بها.

راجع الاسطر (31 – 33) الجزء الخاص بإظهار رسالة الخطأ ونلاحظ اننا لم نعمل به شيء إلى الآن

الآن نقوم ببناء الكود الذي يقوم بالتحقق من الشرط وتطبيق التحقق من الصحة على أداة addressDate أو ازالتها بحسب حالة الشرط في أداتي radio، وللقيام بهذا الأمر هنالك عدة طرق منها إنشاء دالة وليكن اسمها addressDateValidation وهذه الدالة تستقبل باراميتر نصي وهو عبارة عن قيمة value لأداتي radio اما temporary (مؤقت) او permanent (دائم) ومن ثم تتحقق من هذه القيمة إذا كانت temporary تضيف التحقق من الصحة required لأداة التاريخ addressDate وإذا كانت غير ذلك أي بمعنى قيمة الباراميتر permanent تزيل التحقق من الصحة، ولإضافة أو إزالة التحقق من الصحة هنالك ثلاث دوال نستخدمها، كالتالي:

setValidators: وتستقبل باراميتر أو مجموعة باراميترات على شكل مصفوفة تمثل أنواع التحقق من الصحة.

clearValidators: لإزالة أنواع التحقق من الصحة من الأداة.

updateValueAndValidity: تحديث الأداة مع كل عملية إضافة أو إزالة لأنواع التحقق من الصحة.

وأخيراً يتم تنفيذ هذه الدالة (addressDateValidation) في مكانين المكان الأول من الخلال الحدث (blur) في أداة التاريخ ذات الاسم البرمجي addressDate والثاني في الدالة valueChanges (الذي تكلمنا عنها سابقاً) لأداتي radio ذوات الاسم البرمجي addressType، والسبب في ذلك أننا نريد ان ينفذ الكود في الدالة (addressDateValidation) في حالة أن المستخدم لمس الأداة addressDate لذلك نستخدم الحدث (bluer) وفي حال أن المستخدم قام بتغيير بين خيارات اداتي radio لذلك نستخدم valueChanges لمراقبة التغييرات وتنفيذ الكود، الآن لنترجم جميع ماقلناه إلى كود برمجي، لذلك نذهب إلى ملف app.component.ts ونضيف الكود التالي:

ملف app.component.ts

```
1. import { Component, OnInit } from '@angular/core';
2.
3. import { FormGroup, FormBuilder, Validators } from '@angular/forms';
4. import { CustomValidator } from 'src/app/shared/custom.validators';
5.
6. @Component({
7.     selector: 'app-root',
8.     templateUrl: './app.component.html',
9.     styleUrls: ['./app.component.css']
10.})
11.
12.export class AppComponent implements OnInit {
13.
14.    states: string[] = ['ALRiyadh', 'Makkah', 'AlSharqiyah', 'AlQasim'];
15.    form: FormGroup;
16.    userNameLength: any = '0';
17.    names: string[] = ['admin', 'administrator'];
18.
19.    messageValidation = {
20.        'userName': {
21.            'required': 'اسم المستخدم مطلوب',
22.            'pattern': 'اسم المستخدم على الاقل ثلاث خانات',
23.            'forbiddenNames': 'لايُسمح بتسجيل اسم المستخدم المُدخل',
```

```
            'isEnglishLetters': 'لا يُسمح بوجود المسافات او الارقام او الرموز الخاصة او حروف غير الإنجليزية'
        },
        'email': {
            'required': ' البريد الإلكتروني مطلوب',
            'emailValidation': 'صيغة البريد الإلكتروني غير صحيحة'
        },
        'passwordGroup': {
            'passwordValidation': 'كلمة السر غير متطابقة'
        },
        'password': {
            'required': 'كلمة االسر مطلوبة',
            'pattern': ' كلمة السر ست خانات ارقام وحروف وعلى الاقل حرف واحد كبير'
        },
        'confirmPassword': {
            'required': ' الحقل مطلوب'
        },
        'gender': {
            'required': 'الحقل مطلوب'
        },
        'addressDate': {
            'required': 'حقل تاريخ إنتهاء أقامة السكن مطلوب'
        },
        'city': {
            'required': 'حقل اسم المدينة مطلوب'
        },
        'state': {
            'required': 'حقل المنطقة مطلوب'
        },
        'zipCode': {
            'required': 'حقل الرمز البريدي مطلوب',
            'pattern': 'الرمز البريدي لابد أن يكون قيمة رقمية من خمس خانات'
        }
    };

    currentMessageValidation = {
        'userName': '',
        'email': '',
        'passwordGroup': '',
        'password': '',
        'confirmPassword': '',
        'gender': '',
        'addressDate': '',
        'city': '',
        'state': '',
        'zipCode': ''
    };

    constructor(private fb: FormBuilder) { }

    ngOnInit() {
        this.form = this.fb.group({
            userName: [null,
                [
```

```
                    Validators.required,
                    Validators.pattern('.{3,}'),
                    CustomValidator.forbiddenNames(this.names),
                    CustomValidator.isEnglishLetters
                ]
            ],
            email: [null,
                [
                    Validators.required,
                    CustomValidator.emailValidation
                ]
            ],
            passwordGroup: this.fb.group({
                password: [null,
                    [
                        Validators.required,
                        Validators.pattern('(?=.*[A-Za-z])(?=.*[A-Z])(?!.* )(?=.*[0-9])[A-Za-z0-9d$@].{5,}')
                    ]
                ],
                confirmPassword: [null,
                    [
                        Validators.required
                    ]
                   ]
            }, { validator: CustomValidator.passwordValidation }),
            gender: [null, Validators.required],
            address: this.fb.group({
                addressType: ['permanent'],
                addressDate: [null],
                city: [null, Validators.required],
                state: [null, Validators.required],
                zipCode: [null, [Validators.required, Validators.pattern('^[0-9]{5}$')]]
            })
        });

        this.userName.valueChanges.subscribe((value: string) => {
            this.userNameLength = value.length;
        });

        this.form.valueChanges.subscribe(data => {
            this.loopThroughControls(this.form);
        });

        this.addressType.valueChanges.subscribe(data => {
            this.addressDateValidation(data);
        });

    }

    addressDateValidation(value: string) {
        if (value === 'temporary') {
```

```
128.                 this.addressDate.setValidators(Validators.required);
129.             } else {
130.                 this.addressDate.clearValidators();
131.                 this.addressDate.markAsUntouched();
132.                 this.addressDate.reset();
133.             }
134.             this.addressDate.updateValueAndValidity();
135.         }
136.
137.         save() {
138.             console.log(this.form);
139.         }
140.
141.         loopThroughControls(formGroup: FormGroup = this.form) {
142.             Object.keys(formGroup.controls).forEach((key: string) => {
143.                 const controlName = formGroup.get(key);
144.                 this.currentMessageValidation[key] = '';
145.                 if (controlName && controlName.invalid && controlName.touched) {
146.                     const messages = this.messageValidation[key];
147.                     for (const controlError in controlName.errors) {
148.                         if (controlError) {
149.                             this.currentMessageValidation[key] +=
150.                                 messages[controlError] + ' ';
151.                         }
152.                     }
153.                 }
154.                 if (controlName instanceof FormGroup) {
155.                     this.loopThroughControls(controlName);
156.                 }
157.             });
158.         }
159.
160.         laodData() {
161.             this.form.patchValue({
162.                 userName: 'DivFaisal',
163.                 email: 'test@test.com',
164.                 passwordGroup: {
165.                     password: 'Aa1111',
166.                     confirmPassword: 'Aa1111'
167.                 },
168.                 gender: 'male',
169.                 address: {
170.                     city: 'Riyadh',
171.                     state: 'ALRiyadh',
172.                     zipCode: '87678'
173.                 }
174.             });
175.         }
176.
177.         get userName() {
178.             return this.form.get('userName');
179.         }
180.         get email() {
```

```
181.          return this.form.get('email');
182.      }
183.      get password() {
184.          return this.form.get('password');
185.      }
186.      get confirmPassword() {
187.          return this.form.get('confirmPassword');
188.      }
189.      get gender() {
190.          return this.form.get('gender');
191.      }
192.      get address() {
193.          return this.form.get('address');
194.      }
195.      get city() {
196.          return this.form.get('address').get('city');
197.      }
198.      get state() {
199.          return this.form.get('address').get('state');
200.      }
201.      get zipCode() {
202.          return this.form.get('address').get('zipCode');
203.      }
204.      get addressType() {
205.          return this.form.get('address').get('addressType');
206.      }
207.      get addressDate() {
208.          return this.form.get('address').get('addressDate');
209.      }
210.  }
211.
```

راجع الاسطر (من 120 إلى 122) الدالة valueChanges لأداتي addressType وتقوم بمراقبة هذين الأداتين وفي حال قيام المستخدم بأي تعديل على قيمهما تعيد هذه القيمة من خلال الباراميتر الذي اسميناهdata وذلك يعني أن قيمة data اما ان تكون temporary او permanent ومن ثم نمرر هذه القيمة إلى الدالة addressDateValidation

راجع الاسطر (من 126 إلى 135) هذه هي دالة التحقق من الشروط وإضافة أو إزالة التحقق من الصحة، حيث بالبداية تستقبل باراميتر اسميته value يحتوي على قيمة الأداتين ومن ثم تقوم بتأكد من أن قيمة هذا الباراميتر إذا كان temporary تضيف التحقق من الصحة required إلى أداة addressDate وإن لم يكن تقوم بإزالة دالة التحقق required عن طريق clearValidators وجعلها untouched وإزالة القيمة منها إذا كان هنالك قيمة، واخيراً تحديث هذه القيمة عن طريق الدالة updateValueAndValidity

ومن ثم نذهب إلى ملف app.component.html ونضيف هذه الدالة إلى الحدث (blur) لأداة التاريخ، وبنفس الوقت نضيف شروط إضافة واخفاء كلاسات البوتستراب ورسائل الخطأ وغيرها من الأمور التي نعملها في كل مرة، ولا اريد إعادة شرحها هنا منعاً لتكرار:

ملف app.component.html

```
<!-- بداية النموذج الفرعي -->
<fieldset class="scheduler-border" formGroupName="address">
  <legend class="scheduler-border">Address Information</legend>

  <label class="pr-2" style="margin-bottom: 0px">Address Type:</label>
  <br />
  <div class="form-check form-check-inline">
    <input
      class="form-check-input"
      type="radio"
      formControlName="addressType"
      id="temporary"
      value="temporary"
    />
    <label class="form-check-label" for="temporary">Temporary</label>
  </div>
  <div class="form-check form-check-inline">
    <input
      class="form-check-input"
      type="radio"
      formControlName="addressType"
      id="permanent"
      value="permanent"
    />
    <label class="form-check-label" for="permanent">Permanent</label>
  </div>

  <input
    type="date"
    formControlName="addressDate"
    [class.has-error]="currentMessageValidation.addressDate"
    *ngIf="addressType.value === 'temporary'"
    (blur)="addressDateValidation('temporary')"
  />
  <div>
    <small class="text-danger" *ngIf="currentMessageValidation.addressDate">
      {{ currentMessageValidation.addressDate }}
    </small>
  </div>

  <!-- أداة إدخال اسم المدينة -->
  <div class="form-group pt-4">
    <label>City</label>
    <input
      formControlName="city"
      [ngClass]="{
                'form-control': true,
                'is-invalid': currentMessageValidation.city
              }"
      (blur)="loopThroughControls()"
    />
```

```
52.     <!-- جزء التحقق من الصحة لأداة إدخال اسم المدينة -->
53.     <small class="text-danger" *ngIf="currentMessageValidation.city">
54.       {{ currentMessageValidation.city }}
55.     </small>
56.   </div>
57.   <!-- أداة اختيار اسم المنطقة او الولاية -->
58.   <div class="form-group">
59.     <label>State</label>
60.     <select
61.       formControlName="state"
62.       [ngClass]="{
63.                 'form-control': true,
64.                 'is-invalid': currentMessageValidation.state
65.               }"
66.       (blur)="loopThroughControls()"
67.     >
68.       <option selected [ngValue]="null">Choose...</option>
69.       <option *ngFor="let item of states" [value]="item">{{ item }}</option>
70.     </select>
71.     <!-- جزء التحقق من الصحة لأداة أختيار اسم المنطقة -->
72.     <small class="text-danger" *ngIf="currentMessageValidation.state">
73.       {{ currentMessageValidation.state }}
74.     </small>
75.   </div>
76.   <!-- أداة إدخال الرمز البريدي -->
77.   <div class="form-group">
78.     <label>Zip Code</label>
79.     <input
80.       formControlName="zipCode"
81.       [ngClass]="{
82.                 'form-control': true,
83.                 'is-invalid': currentMessageValidation.zipCode
84.               }"
85.       (blur)="loopThroughControls()"
86.     />
87.     <!-- جزء التحقق من الصحة لأداة إدخال الرمز البريدي -->
88.     <small class="text-danger" *ngIf="currentMessageValidation.zipCode">
89.       {{ currentMessageValidation.zipCode }}
90.     </small>
91.   </div>
92. </fieldset>
93.
```

راجع السطر 31 (شرط إخفاء وإظهار كلاس البوتستراب الذي يعطي إطار احمر حول الأداة في حال الخطأ)

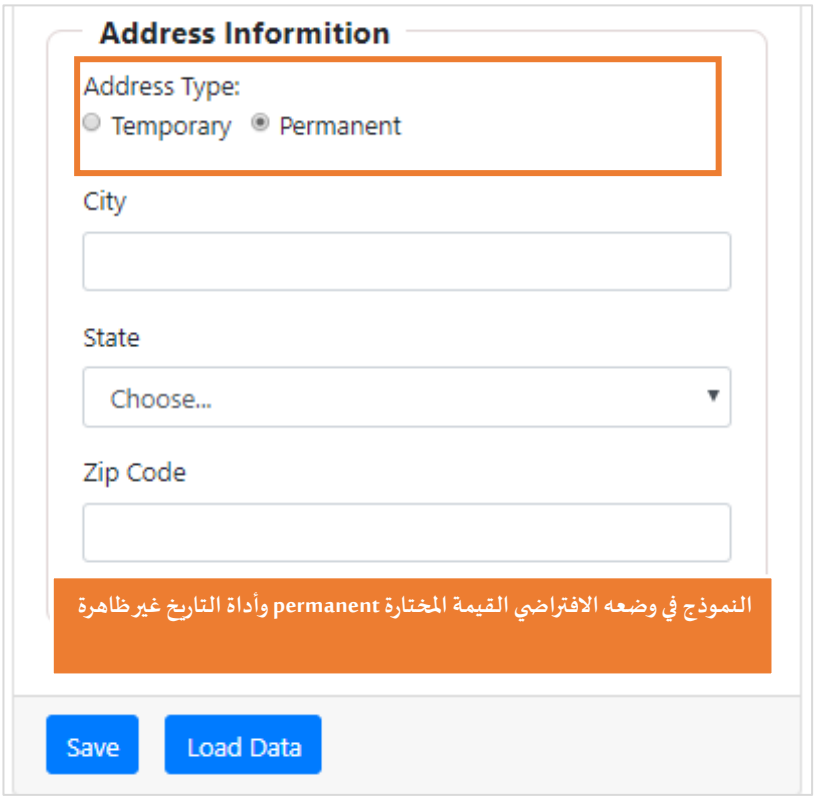
راجع السطر 32 (شرط إخفاء أداة التاريخ في حال اختيار المستخدم لخيار permanent)

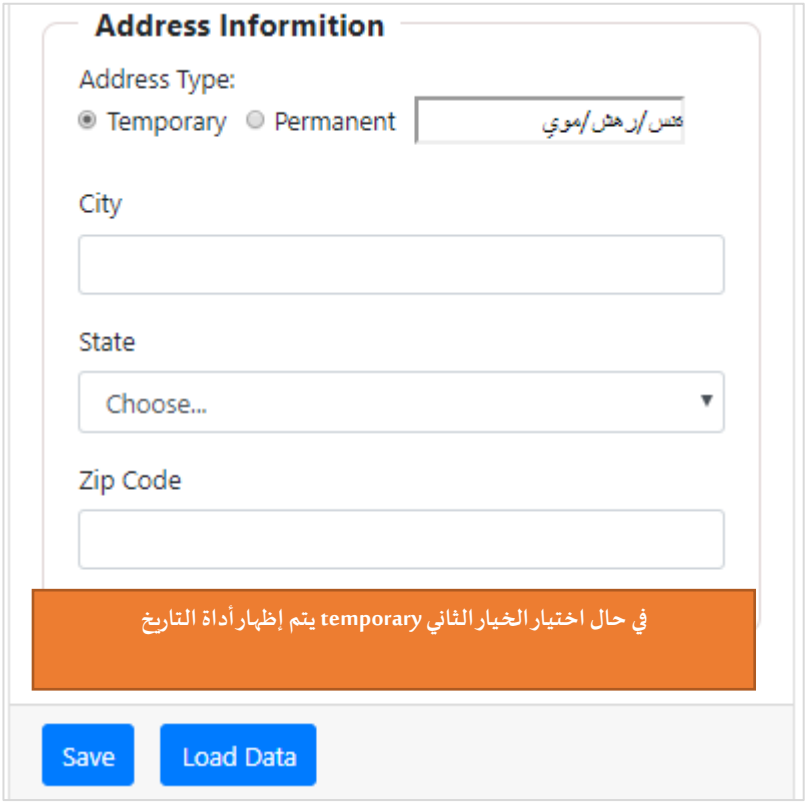
راجع السطر 33 (الدالة يتم تنفيذها في الحدث (blur) لأداة التاريخ ومررنا لها القيمة temporary بشكل افتراضي)

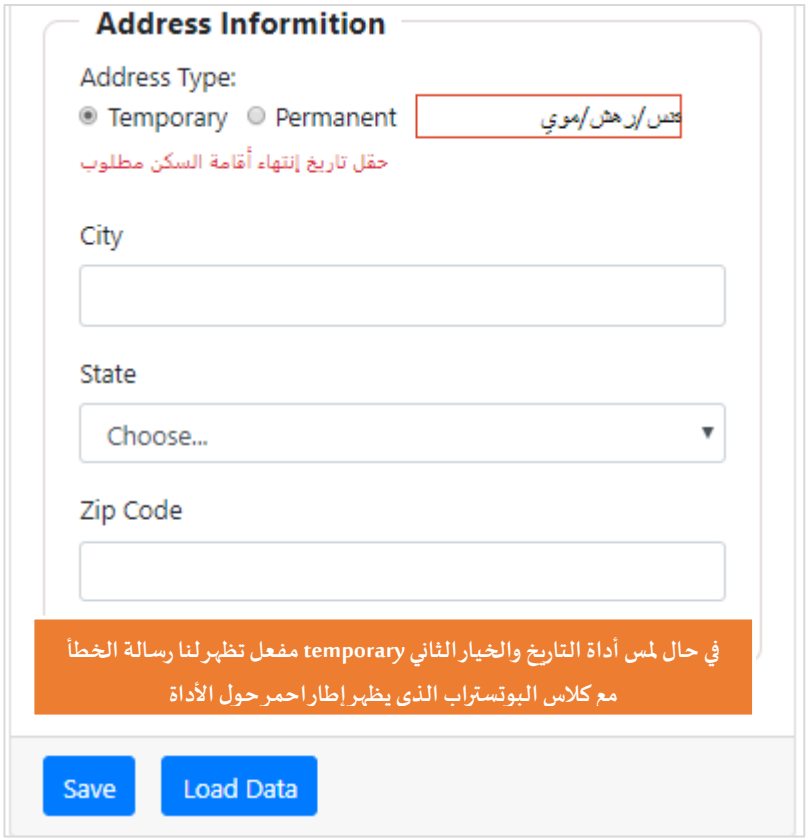
راجع السطر 36 (شرط إخفاء وإظهار رسالة الخطأ)

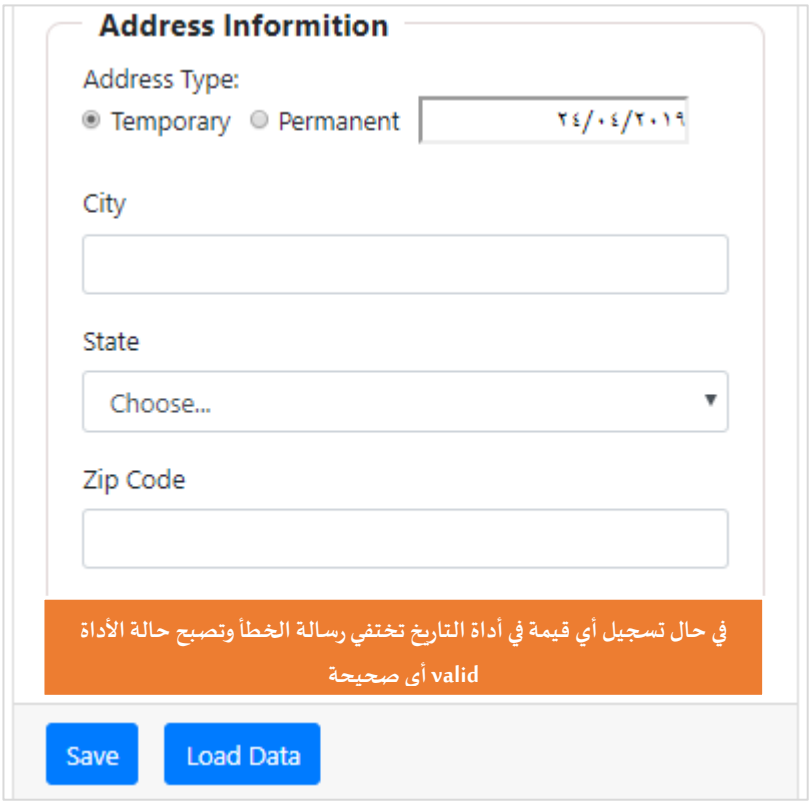
راجع السطر 37 (محتوى رسالة الخطأ)

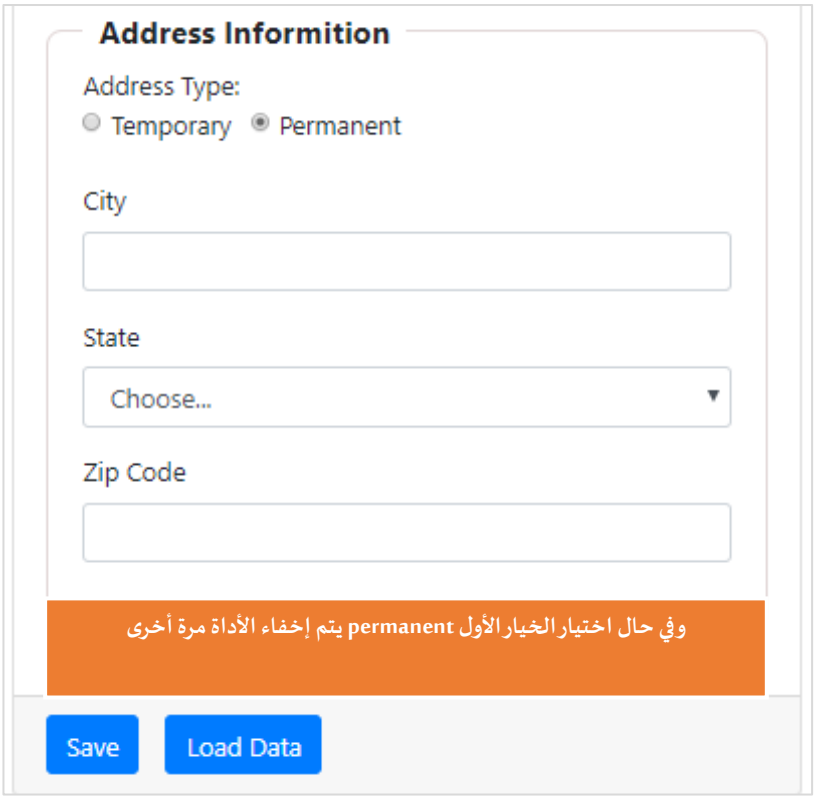
الآن لنشاهد ما قمنا به من تعديلات على النموذج:

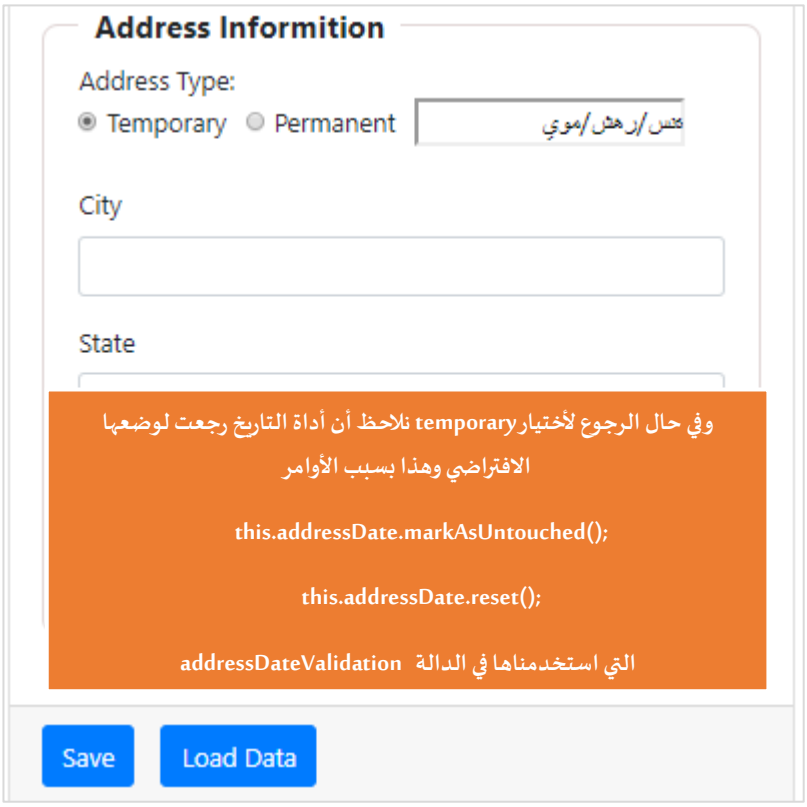





Address Information
Address Type:
Temporary
Permanent
حقل تاريخ إنتهاء أقامة السكن مطلوب
City
State
Choose...
Zip Code
في حال لمس أداة التاريخ والخيار الثاني temporary مفعل تظهر لنا رسالة الخطأ
مع كلاس البوتستراب الذى يظهر إطار احمر حول الأداة
Save
Load Data


Address Information
Address Type:
Temporary
Permanent
٢٤/٠٤/٢٠١٩
City
State
Choose...
Zip Code
في حال تسجيل أي قيمة في أداة التاريخ تختفي رسالة الخطأ وتصبح حالة الأداة
valid أى صحيحة
Save
Load Data





وبذلك نكون أنهينا النوع الثالث من أنواع التحقق من الصحة وسوف نتكلم الآن عن النوع الرابع وهو Asynchronous Validation.

5.4. التحقق من الصحة غير التزامني Asynchronous Validation:

هذا النوع من التحقق من الصحة في الغالب يتعامل مع البيانات القادمة من قاعدة البيانات والتي هي الأخرى في الغالب تكون موجودة على سيرفر في شبكة حاسوبية، ولو رجعنا إلى بداية كلامنا في الجزء الخاص بأنشاء النموذج برمجياً قلنا ان الاسم البرمجي والذي يُشير إلى الأداة يأخذ قيمة عبارة عن مصفوفة تتكون من ثلاث أجزاء الجزء الأول هي القيمة الافتراضية للأداة والجزء الثاني هو دوال التحقق من الصحة سواء built-in او custom اما الجزء الثالث فهو Asynchronous Validation او ما يسمى التحقق من الصحة بشكل غير تزامني، وسبب هذه التسمية ان هذه النوع يتعامل مع بيانات قادمة من السيرفر وهذا يأخذ وقت من حيث اتصال الموقع بالسيرفر وانتظار الرد إلى أن يحصل استجابة وهذا قد يستغرق ثوانٍ او اكثر على حسب عوامل متعددة من سرعة الاتصال بالشبكة وغيره من العوامل الأخرى، لذلك لا نريد من الموقع أن يتوقف عن العمل إلى أن يأتيه الرد من السيرفر وانما يستطيع المستخدم أن يكمل عمله بشكل عادي وعند حصول استجابة من السيرفر يتم التحقق وإظهار النتيجة وهذه الطريقة تسمى الغير تزامنية Asynchronous، وله صور عديدة من أهمها التحقق من أن اسم المستخدم او البريد الإلكتروني الذي يريد المستخدم التسجيل فيه متاح او لا، فبهذه الطريقة يستطيع المستخدم كتابة اسم المستخدم ولا يتوقف النظام بانتظار الاستجابة من السيرفر وانما يُتيح للمستخدم استكمال بقية البيانات على النموذج وعند حدوث استجابة تظهر مثلاً رسالة تبين للمستخدم ان هذا الاسم متاح او لا.

ملاحظة: في حقيقة الأمر هذا النوع من التحقق من الصحة هو custom validation ويتم فيه جميع القواعد التي ذكرناها فيه، ولكن تم فصله هنا كنوع لوحده لأن طريقة تعامله مع البيانات تختلف بالإضافة إلى تعامله مع تقنيات البرمجة الغير تزامنية.

عندما نتكلم عن تقنيات البرمجة الغير تزامنية فهذا يقودنا إلى ما يسمى Promise و Async Awaite و Observable، وهي من التقنيات المهمة جداً التي يجب على كل مطور ويب Front-End Developer أن يُتقنها بشكل جيد، حيث أن جميع هذه التقنيات تتعامل مع البيانات الغير تزامنية القادمة من السيرفر لعرضها على المتصفح او لأرسالها إلى السيرفر للحفظ او التعديل او الحذف، لأننا لا نريد من البرنامج أن يتوقف عن العمل بانتظار الرد من السيرفر، كأننا نقول للبرنامج في حال طلب المستخدم عرض بيانات معينة من قاعدة البيانات على السيرفر قم بأرسال هذا الطلب وبنفس الوقت لا تتوقف تنتظر الرد وإنما اسمح للمستخدم بالقيام بأي عمل يريده داخل البرنامج وفي حال وصول الرد من السيرفر اعرض هذه البيانات للمستخدم، وجميع التقنيات السابقة الذكر تتيح لنا القيام بهذا الأمر، وبنفس الوقت ليس هنا المقام لشرح هذه التقنيات بشكل مفصل لأنه يُفترض بالقارئ انه ملم ولو بأساسيات هذه التقنيات، وانما ما يهمنا هنا هو كيف نستفيد من هذه التقنيات الثلاث في Asynchronous Validation، وسوف اورد مثال على Promise وعلى Observable.

1.5.4. Asynchronous Validation with Promises:

ليكن المثال اننا نريد أن يتحقق من اسم المستخدم username هل هو متاح او ولا وذلك عن طريق ارسال طلب لسيرفر لتأكد من أن اسم المستخدم متاح من عدمه، وهذه العملية قد تستغرق بعض الوقت تبعاً لعوامل متعددة منها جودة الاتصال بالشبكة والضغط على السيرفر وحجم قاعدة البيانات المخزنة على السيرفر ...الخ، وبما انه ليس لدينا هنا قاعدة بيانات ولا سيرفر سوف أُحاكي هذا الأمر من خلال استخدام الدالة setTimeout() والتي تُتيح بتنفيذ الكود بداخلها بعد

فترة زمنية معينة ولتكن ثانيتين (2000 milliseconds)، وبما أن هذه النوع من التحقق من الصحة هو Custom Validation فلذلك سوف ننشأ دالة static في ملف custom.validators.ts الذي انشأناه سابقاً، وليكن اسم الدالة isUserNameTaken() وتأخذ مصفوفة نصية كبارامتر وليكن اسمها userNames حيث انها تحتوي على أسماء المستخدم التي نُحاكي انها مخزنة في قاعدة البيانات، كالتالي:

```
1.  static isUserNameTaken(userNames: string[]): ValidatorFn  {
2.
3.
4.
5.  }
```

وكما كنا نعمل سابقاً عن طريق استخدام ميزة Closure نعيد الدالة الفرعية التي تحتوي على بارامتر واحد هو الأداة أو النموذج الرئيسي أو الفرعي (على حسب احتياجنا منه) من النوع AbstractControl ولكن الفرق هنا أن الدالة هنا لا تُعيد كائن نصي وقيمته any او Boolean او تُعيد null كما كنا نفعل سابقاً، وانما تُعيد promise نوعه كائن وقيمته نوعها any او null أو في تُعيد observable نوعه كائن وقيمته نوعها any او null، كالتالي:

```
1.  static isUserNameTaken(userNames: string[]): ValidatorFn {
2.   return (control: AbstractControl): Promise<ValidationErrors | null> | Observable<ValidationErrors | null> =>{
3.
4.
5.
6.
7.
8.   };
9.  }
```

النوع الأول الذي ممكن أن تُعيده الدالة وهو promise ونوعه أما كائن ValiditionErrors أو null

أو

النوع الثاني الذي ممكن أن تُعيده الدالة وهو observable ونوعه أما كائن ValidationErrors او null

ملاحظات مهمة:

- ✓ الكائن ValidationErrors هو مشابهه لما كنا نفعله سابقاً {[key: string]: boolean} ولكن هنا بصياغة أخرى قدمتها لنا angular جاهزة وقيمتها any وليس boolean، ولكن عند استخدام هذه الطريقة يجب استدعائها من angular forms، كالتالي:

```
1.  import { AbstractControl, ValidatorFn, ValidationErrors } from '@angular/forms';
```

مع العلم أنه يمكن استخدام الصيغة التي كنا نستخدمها سابقا بدلاً من ValidationErrors بدون أي مشاكل.

- ✓ الذي يُحدد ما تُعيده الدالة هو Promise أو Observable هو الLogic داخل الدالة مع العلم انه يمكن الاكتفاء بواحد منها إذا كان logic داخل الدالة يُعيد Promise يمكن الاكتفاء فقط بكتابة | Promise<ValidationErrors null> وبنفس الطريقة بالنسبة لObservable، وإذا كان هنالك احتمال انها تُعيد كلا النوعين فيتم كتابتها معاً.
- ✓ نُعيد ما ذكرناه سابقاً أن الدالة isUserNameTaken تُعيد ValidatorFn وهو نوع جاهز من angular معناه أن هذه الدالة تُعيد دالة أخرى تحتوي على بارامتر من النوع Abstract Control.

الخطوة الثالثة التي نعملها نُعيد Promise جديد لأنه كما قلنا سابقاً أن الدالة السابقة لابد أن تُعيد أما Promise أو Observable، كالتالي:

```
static isUserNameTaken(userNames: string[]): ValidatorFn {
  return(control:AbstractControl): Promise<ValidationErrors | null> | Observable<ValidationErrors | null> =>{
      return new Promise((resolve, reject) => {

      });
  };
}
```

وداخل هذه Promise والذي هو في الحقيقة عبارة عن دالة، نقوم بتعريف ثابتين الأول نُخزن فيه قيمة الأداة userName أو بصياغة أخرى القيمة التي يُدخلها المستخدم في حقل اسم المستخدم بعد تحويلها إلى حروف إنجليزية صغيرة Lower Case، أما الثابت الثاني فهو عبارة عن مصفوفة ونُخزن فيه نسخة من مصفوفة الأسماء userNames عن طريق الدالة map بعد تحويلها إلى حروف إنجليزية صغيرة Lower Case، وليكن اسم الثابت الأول userNameValue والثابت الثاني userNamesLowerCase، كالتالي:

```
static isUserNameTaken(userNames: string[]): ValidatorFn {
  return (control: AbstractControl): Promise<ValidationErrors | null> | Observable<ValidationErrors | null>=>{
     return new Promise((resolve, reject) => {
         const userNameValue = control.value.toLowerCase();
         const userNamesLowerCase = userNames.map(names => names.toLowerCase());
     });
  };
}
```

الخطوة الرابعة نقوم بكتابة الدالة setTimeout ومهمتها فقط تنفيذ الكود بعد فترة زمنية محددة ولتكن بعد ثانيتين لمحاكاة الاتصال بالسيرفر وانتظار الرد بعد ثانيتين، كالتالي:

```
static isUserNameTaken(userNames: string[]): ValidatorFn {
  return (control: AbstractControl): Promise<ValidationErrors | null> | Observable<ValidationErrors | null>=>{
     return new Promise((resolve, reject) => {
         const userNameValue = control.value.toLowerCase();
         const userNamesLowerCase = userNames.map(names => names.toLowerCase());
         setTimeout(() => {

         }, 2000);
     });
  };
}
```

الخطوة الخامسة والأخيرة نقوم بكتابة شرط التحقق من أن المصفوفة userNamesLowerCase تحتوي على القيمة الموجودة في userNameValue فإذا كانت موجودة فذلك يعني أن القيمة التي ادخلها المستخدم في حقل اسم المستخدم موجودة في مصفوفة أسماء المستخدم وذلك يعني انه غير متاح وفي هذه الحالة نُرجع الكائن ValidationErrors والقيمة true اما إذا كانت لا تحتوي القيمة نُرجع null، مع ملاحظة أنه في Promise لا نرجع القيمة الناجحة عن طريق الكلمة return وإنما عن طريق الكلمة resolve، بحيث تكون الدالة بشكلها النهائي كالتالي:

```
static isUserNameTaken(userNames: string[]): ValidatorFn {
```

```
    return (control: AbstractControl): Promise<ValidationErrors | null> | Observable<ValidationErrors | null>=>{
       return new Promise((resolve, reject) => {
           const userNameValue = control.value.toLowerCase();
           const userNamesLowerCase = userNames.map(names => names.toLowerCase());
           setTimeout(() => {
             userNamesLowerCase.includes(userNameValue) ? resolve({ 'isUserNameTaken': true }) : resolve(null);
           }, 2000);
       });
   };
}}
```

بعد اكتمال دالة التحقق من الصحة، ننتقل إلى ملف app.component.ts، ونقوم بعمل الخطوات التالية:

أولاً: تعريف مصفوفة نخزن فيها مجموعة من الأسماء وليكن اسمها userNames، كالتالي:

```
userNames: string[] = ['faisal', 'DivFaisal'];
```

ثانياً: نضيف رسالة الخطأ في الكائن messeageValidation وkey نفس اسم الkeyl للكائن الذي عملنا له resolve في دالة التحقق من الصحة التي أنشأناها سابقاً، كالتالي:

app.component.ts جزء من ملف

```
messageValidation = {
    'userName': {
        'required': 'اسم المستخدم مطلوب',
        'pattern': 'اسم المستخدم على الاقل ثلاث خانات',
        'forbiddenNames': 'لايُسمح بتسجيل اسم المستخدم المُدخل',
        'isEnglishLetters': 'لا يُسمح بوجود المسافات او الارقام او الرموز الخاصة او حروف غير الإنجليزية',
        'isUserNameTaken': 'اسم المستخدم غير متاح'
    },
    'email': {
        'required': ' البريد الإلكتروني مطلوب',
        'emailValidation': 'صيغة البريد الإلكتروني غير صحيحة'
    },
    'passwordGroup': {
        'passwordValidation': 'كلمة السر غير متطابقة'
    },
    'password': {
        'required': 'كلمة االسر مطلوبة',
        'pattern': ' كلمة السر ست خانات ارقام وحروف وعلى الاقل حرف واحد كبير'
    },
    'confirmPassword': {
        'required': ' الحقل مطلوب'
    },
    'gender': {
        'required': 'الحقل مطلوب'
    },
    'addressDate': {
        'required': 'حقل تاريخ إنتهاء أقامة السكن مطلوب'
    },
    'city': {
        'required': 'حقل اسم المدينة مطلوب'
    },
    'state': {
        'required': 'حقل المنطقة مطلوب'
```

```
    },
    'zipCode': {
        'required': 'حقل الرمز البريدي مطلوب',
        'pattern': 'الرمز البريدي لابد أن يكون قيمة رقمية من خمس خانات'
    }
};
```

ثالثاً: نضيف دالة التحقق من الصحة لقيم الاسم البرمجي username الذي يُشير إلى حقل اسم المستخدم في النموذج، مع ملاحظة أن هذه الدالة لا تُضاف في الجزء الخاص بـCustom Validation و Built-In Validation وإنما في الجزء الذي يليه عن طريق إضافة الفاصلة ثم كتابة اسم الكلاس وبعده اسم الدالة وفي حال كانت لدينا أكثر من دالة نضيف اقواس المصفوفة ثم نكتب ما نشاء من الدوال، مع تمرير مصفوفة الأسماء userNames لهذه الدالة، كالتالي:

app.component.ts جزء من ملف

```
ngOnInit() {
    this.form = this.fb.group({
        userName: [null,
            [
                Validators.required,
                Validators.pattern('.{3,}'),
                CustomValidator.forbiddenNames(this.names),
                CustomValidator.isEnglishLetters
            ], [CustomValidator.isUserNameTaken(this.userNames)]
        ],
        email: [null,
            [
                Validators.required,
                CustomValidator.emailValidation
            ]
        ],
        passwordGroup: this.fb.group({
            password: [null,
                [
                    Validators.required,
                    Validators.pattern('(?=.*[A-Za-z])(?=.*[A-Z])(?!.* )(?=.*[0-9])[A-Za-z0-9d$@].{5,}')
                ]
            ],
            confirmPassword: [null,
                [
                    Validators.required
                ]
            ]
        }, { validator: CustomValidator.passwordValidation }),
        gender: [null, Validators.required],
        address: this.fb.group({
            addressType: ['permanent'],
            addressDate: [null],
            city: [null, Validators.required],
            state: [null, Validators.required],
```

نلاحظ أننا لم نقم بكتابة دالة التحقق من الصحة مع دوال التحقق التي أنشأناها سابقاً لأن أي حقل يأخذ قيمة من ثلاث أجزاء الجزء الأول وهو القيمة الافتراضية وقيمته هنا null والجزء الثاني وهو دوال التحقق من الصحة Built-In والجزء الثالث هو دوال التحقق من الصحة Custom وتتم Asynchronies ويتم الفصل بين هذه الأجزاء الثلاثة عن طريق الفاصلة (,)

```
            zipCode: [null, [Validators.required, Validators.pattern('^[0-9]{5}$')]]
        })
    });
}

```

بذلك نكون أنهينا التعديلات على هذا الملف وبقي أن نجري التعديلات على الملف app.component.html، وسوف أركز على هذه الملف بشكل عام على النقاط التالية:

- ✓ إظهار رسالة للمستخدم لتبين له أن البرنامج يتصل بالسيرفر لتحقق من إتاحة اسم المستخدم الذي قام بإدخاله.
- ✓ إظهار تغذية راجعة للمستخدم عن طريقة وضع إطار اخضر حول الأداة مع علامة الصح لنبين له أن اسم المستخدم متاح.
- ✓ إظهار رسالة للمستخدم في حال أن الاسم الذي قام بإدخال غير متاح.

أولاً: يمكن إظهار رسالة للمستخدم تبين له ان البرنامج يتصل بالسيرفر وينتظر الرد منه عن طريق كلاس css تضيفه angular بشكل تلقائي للأداة التي تحتوي على دالة التحقق من الصحة الغير تزامنية Asynchronous عند وجود أي اتصال بالسيرفر ويقوم بحذفه عند تلقي الرد، واسم هذا الكلاس pending ولذلك يمكن الاستفادة منه عن طريق مثلاً إضافة div ونضع شرط له ان يظهر فقط في حالة أن الكلاس pending مضاف له أي بمعنى قيمته تساوي true مع إضافة بعض كلاسات البوتستراب لتعطي شكل جمالي للرسالة، ويمكن الوصول لهذا الكلاس عن طريق كتابة الاسم البرمجي للأداة المستهدفة ثم نقطة ثم اسم الكلاس السابق الذكر، كالتالي:

```
<div class="alert alert-info col-10" *ngIf="userName.pending">
    جاري التحقق من اتاحة اسم المستخدم
</div>
```

ثانياً: يمكن تقديم تغذية راجعة للمستخدم لنبين له أن الاسم الذي اختاره متاح ويمكن استخدامه وهنالك طرق متعددة تختلف بحسب الاحتياج ونوع إطار عمل css الذي يستخدمه المطور في مشروعه، وانا هنا بما أنني استخدم إطار عمل البوتستراب فسوف استفيد من كلاس جاهز يوفره لنا واسم هذا الكلاس is-valid مع وضع بعض الشروط له وهي أن يظهر هذا الكلاس فقط في حال أن أداة اسم المستخدم لا تحتوي على خطأ isUserTaken بمعنى قيمته تساوي false وكلاس pending ايضاً قيمته تساوي false وقيمة حقل إدخال اسم المستخدم لا تساوي null واخيراً عدد الخانات في حقل إدخال اسم المستخدم أكبر من أو تساوي ثلاثة لأن إذا كانت أقل من ثلاث فهنالك يظهر خطأ آخر وهو عدد الخانات لابد أن تكون ثلاث خانات كحد أدنى، وبنفس الوقت نضيف شرط آخر لكلاس is-invalid وهو أن يظهر إذا كان هنالك خطأ ما في username عن طريق الكائن currentMessageValidation (أو) هنالك خطأ في isUserTaken أي بمعنى قيمته تساوي true، كالتالي:

جزء من ملف app.component.html

```
<input
  formControlName="userName"
  [ngClass]="{
    'col-10': true,
```

```
5.      'form-control': true,
6.      'is-
   invalid': currentMessageValidation.userName || userName.hasError('isUserNameTaken'),
7.      'is-valid':
8.        !userName.hasError('isUserNameTaken') &&
9.        !userName.pending &&
10.       userName.value !== null &&
11.       userNameLength >= 3
12.   }"
13.   (blur)="loopThroughControls()"
14. />
```

الشرط الجديد للكلاس is-invalid

الكلاس is-valid مع الشروط الخاصة به

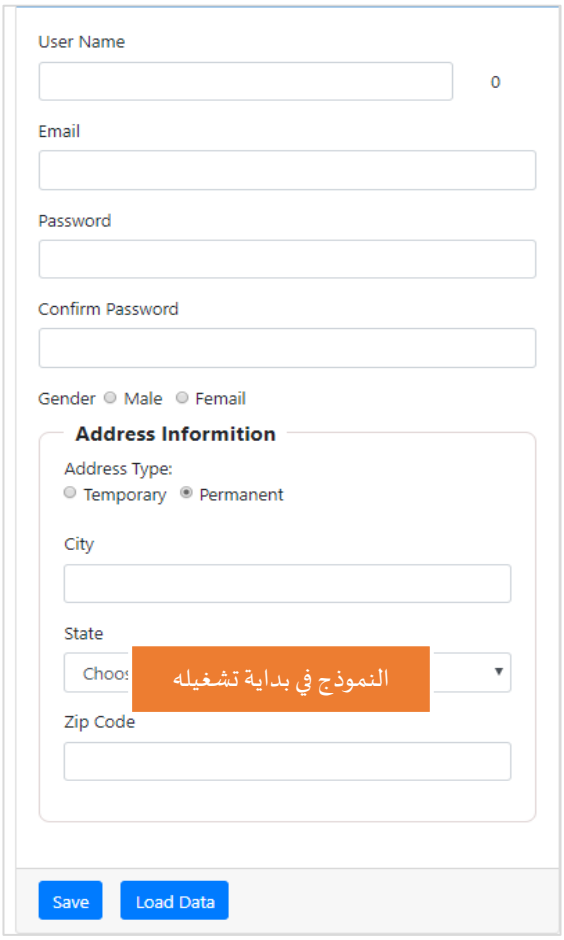
ثالثاً: إظهار رسالة للمستخدم في حال أن أسم المستخدم غير متاح، وهذه الرسالة تظهر بنفس طريقة إظهار رسائل الخطأ عند إضافتنا لها في كائن رسائل الخطأ messageValidation.

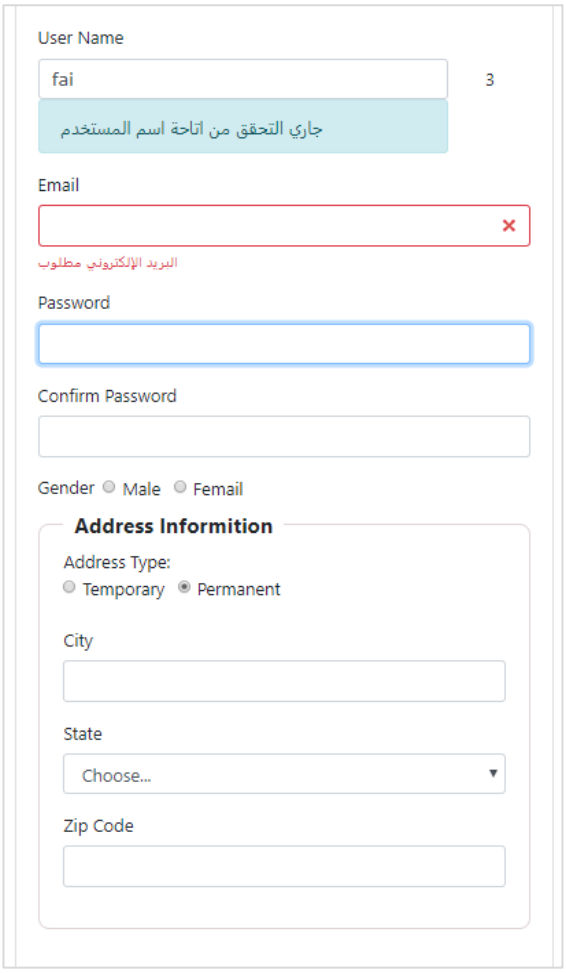
الآن لنضيف الأكواد السابقة إلى كود html الخاص بأداة اسم المستخدم، كالتالي:

app.component.html جزء من ملف

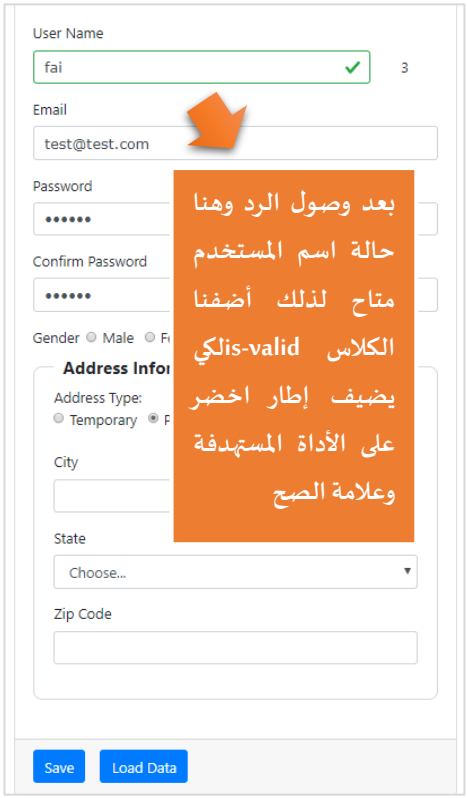
```
1. <div class="form-group">
2.   <label>User Name</label>
3.   <div class="form-inline">
4.     <input
5.       formControlName="userName"
6.       [ngClass]="{
7.         'col-10': true,
8.         'form-control': true,
9.         'is-
   invalid': currentMessageValidation.userName || userName.hasError('isUserNameTaken'),
10.        'is-valid':
11.          !userName.hasError('isUserNameTaken') &&
12.          !userName.pending &&
13.          userName.value !== null &&
14.          userNameLength >= 3
15.      }"
16.      (blur)="loopThroughControls()"
17.    />
18.    <label class="col-2">{{ userNameLength }}</label>
19.  </div>
20.  <!-- جزء التحقق من الصحة الخاص باسم المستخدم -->
21.  <small class="text-danger" *ngIf="currentMessageValidation.userName">
22.    {{ currentMessageValidation.userName }}
23.  </small>
24.  <div class="alert alert-info col-10" *ngIf="userName.pending">
25.    جاري التحقق من اتاحة اسم المستخدم
26.  </div>
27. </div>
```

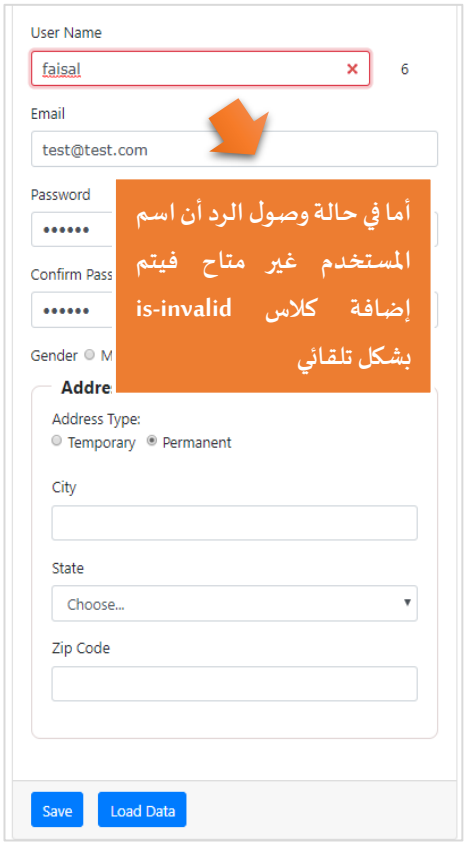
الآن لنشاهد ما قمنا به من تعديلات على النموذج في المتصفح:

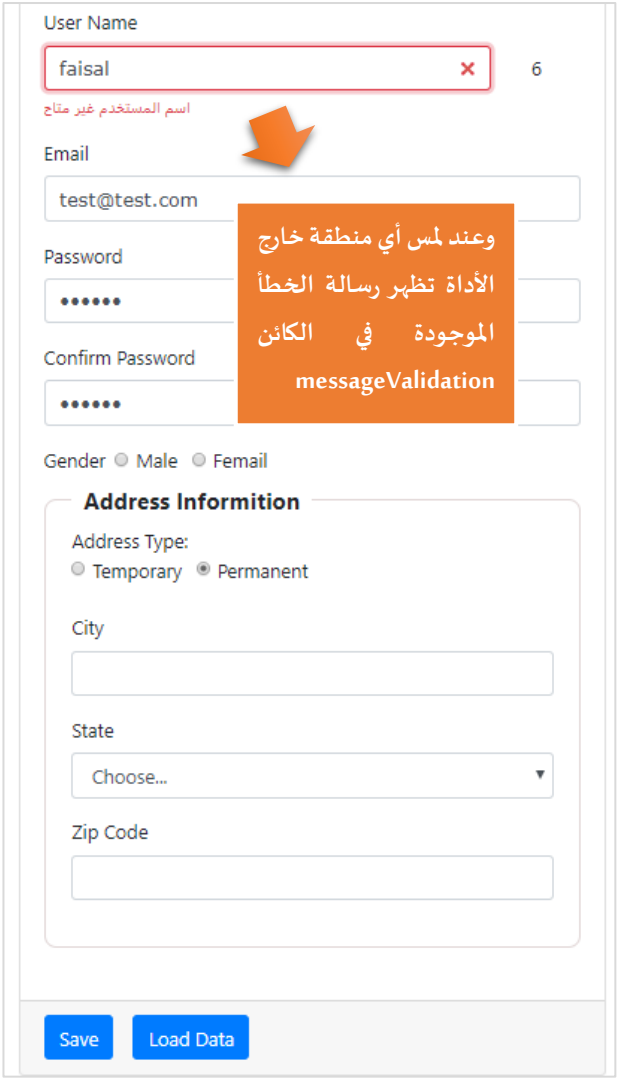

User Name
0
Email
Password
Confirm Password
Gender Male Femail
Address Information
Address Type:
Temporary Permanent
City
State
Choos
النموذج في بداية تشغيله
Zip Code
Save
Load Data


User Name
fai
3
جاري التحقق من اتاحة اسم المستخدم
Email
البريد الإلكتروني مطلوب
Password
Confirm Password
Gender Male Femail
Address Information
Address Type:
Temporary Permanent
City
State
Choose...
Zip Code

نلاحظ عندما نقوم بكتابة ثلاث حروف تظهر له رسالة بسبب إضافة كلاس pending كما تكلمت سابقاً، وبنفس الوقت نلاحظ أن النظام يتيح لك تكملة باقي بيانات النموذج والتحقق من صحتها ولا يتوقف منتظراً الرد من السيرفر، ومن هنا تظهر قوة واهمية تقنيات البرمجة للبيانات الغير تزامنية.







2.5.4. Asynchronous Validation with Observable:

في الحقيقة وفي البرامج الواقعية وخصوصاً في إطار عمل angular يكون التعامل مع البيانات القادمة من السيرفر من خلال تقنيات Observable ومكتبتها RxJs لما تقدمه من عمليات ودوال تسهل لنا التعامل مع هذا النوع من البيانات، فمثلاً المصفوفات وخصوصاً في الجافا سكربت ES6 والإصدارات الأعلى نلاحظ أن هنالك دوال كثيرة تسهل لنا التعامل مع المصفوفات كmaps التي تأخذ نسخة من المصفوفة الأصلية مع إمكانية التعديل عليها وحفظها في النسخة الجديدة (والتي تعاملنا معها سابقاً) او filter وهي نفس map ولكن تنشأ عناصر المصفوفة الجديدة وفق شرط معين كأن تقوم بترشيح او (فلترة) هذه البيانات، وغيرها الكثير من الدوال التي تسهل التعامل مع المصفوفات، وبنفس الطريقة هنا فلدينا كم كبير من الدوال والعمليات operators التي تسهل وتساعد في التعامل مع البيانات الغير تزامنية مثل map وfilter وtake وdelay وfrom وof و....الخ وغيره الكثير ليس هنا المقام لشرحها.

بعد هذه المقدمة البسيطة نبدأ في إعطاء مثال يوضح طريقة التعامل Asynchronous Validation مع Observable، في المثال السابق Promise كنا نتعامل مع حقل اسم المستخدم أما هنا سوف نتعامل مع حقل البريد الإلكتروني لتأكد هل هو متاح أم لا، وبما أنه ليس لدينا Back-End أي قاعدة بيانات وسيرفر، لذلك سوف نحاكي ذلك من خلال مصفوفة نخزن فيها بعض الإيميلات.

وبما أننا نتعامل مع تقنية Observable فسوف نحول هذه المصفوفة إلى Observable.

وللقيام بهذا العمل، نقوم أولاً بأنشاء دالة وليكن اسمها isEmailTaken في ملف custom.validators.ts، ونمرر لها مصفوفة نصية وبنفس الوقت نعيد الدالة الفرعية التي تحتوي على باراميتر الأداة المستهدفة، مع تعريف ثابتين الأول لقيمة الأداة والثاني لأخذ نسخة من المصفوفة، كما كنا نفعل في المثال السابق الخاص بPromise، كالتالي:

```
1.  static isEmailTeken(emails: string[]): ValidatorFn {
2.      return (control: AbstractControl): Promise<ValidationErrors | null> | Observable<ValidationErrors | null> =>{
3.          const emailValue = control.value.toLowerCase();
4.          const emailLowerCase = emails.map(emailsName => emailsName.toLowerCase());
5.
6.      };
7.  }
```

إلى الآن لم نقم بعمل أي شيء جديد فهذا المثال مشابه للمثال السابق والفرق فقط باسم الدالة.

لذلك الخطوة التالية هي تحويل هذه المصفوفة النصية التي تمرر لهذه الدالة تحت اسم emails إلى Observable وللقيام بذلك لدينا دالتين من مكتبة rxjs هما الدالة of والدالة from والفرق أن of تحول المصفوفة إلى مصفوفة من observable اما from فتحول المصفوفة إلى سلسلة نصية من observable، لذلك الأنسب لنا هنا هي الدالة of لذلك سوف نعيد return المصفوفة emails باستخدام الدالة of، كالتالي:

```
1.  static isEmailTeken(emails: string[]): ValidatorFn {
2.      return (control: AbstractControl): Promise<ValidationErrors | null> | Observable<ValidationErrors | null> => {
3.          const emailValue = control.value.toLowerCase();
4.          const emailLowerCase = emails.map(emailsName => emailsName.toLowerCase());
5.
6.          return of(emailLowerCase).pipe(
7.
8.
9.          );
10.
11.     };
12. }
```

وفي الخطوة الأخير نستخدم الدالة delay لمحاكاة وقت الاتصال بالسيرفر ونعطيها ثانيتين (2000 millisecond) ومن ثم نستخدم الدالة map الموجودة في مكتبة rxjs لنأخذ نسخة من المصفوفة ومن ثم نطبق عليها شرط لتأكد هل القيمة في emailValue موجودة في هذه المصفوفة او لا، مع ارجاع القيم إذا كانت موجودة يُرجع كائن بقيمة true وإذا لا يُرجع null، كالتالي:

```
1.  static isEmailTeken(emails: string[]): ValidatorFn {
2.      return (control: AbstractControl): Promise<ValidationErrors | null> | Observable<ValidationErrors | null> => {
3.          const emailValue = control.value.toLowerCase();
4.          const emailLowerCase = emails.map(names => names.toLowerCase());
5.          return of(emailLowerCase).pipe(
6.              delay(1000),
7.              map((newEmail) => newEmail.includes(emailValue) ? { 'isEmailTeken': true } : null)
8.          );
9.      };
10. }
```

الآن لنرى شكل ملف custom.validators.ts بشكله النهائي:

custom.validator.ts ملف

```
1. import { AbstractControl, ValidatorFn, ValidationErrors } from '@angular/forms';
2. import { Observable, of } from 'rxjs';
3. import { map, delay } from 'rxjs/operators';
4.
5. export class CustomValidator {
6.
7.     static forbiddenNames(names: string[]): ValidatorFn {
8.         return (control: AbstractControl): { [key: string]: boolean } | null => {
9.             return names.includes(control.value) ? { 'forbiddenNames': true } : null;
10.         };
11.     }
12.
13.     static isEnglishLetters(control: AbstractControl): { [key: string]: boolean } | nul
    l {
14.         const EnglishLetters = /^[A-Za-z]+$/.test(control.value);
15.         if (control.hasError('pattern') && (control.value as string).length < 3) {
16.             return null;
17.         } else if (!EnglishLetters && (control.value as string).length >= 3) {
18.             return { 'isEnglishLetters': true };
19.         }
20.         return null;
21.     }
22.
23.     static emailValidation(control: AbstractControl): { [key: string]: boolean } | null
    {
24.         const regEx = /^([a-z\d\.-_]+)@([a-z\d-_]+)\.([a-z]{2,4})(\.[a-z]{2,4})?$/;
25.         const email = control.value;
26.         const emailValid = regEx.test(email);
27.         if (control.dirty) {
28.             return (email === '' || emailValid) ? null : { 'emailValidation': true };
29.         }
30.     }
31.
32.     static passwordValidation(formGroup: AbstractControl): { [key: string]: boolean } |
    null {
33.         const password = formGroup.get('password');
34.         const confirmPassword = formGroup.get('confirmPassword');
35.         if (password && confirmPassword &&
36.             password.value !== confirmPassword.value &&
37.             (confirmPassword.dirty || confirmPassword.touched)) {
38.             return { 'passwordValidation': true };
39.         } else {
40.             return null;
41.         }
42.     }
43.
44.     static isUserNameTaken(userNames: string[]): ValidatorFn {
45.         return (control: AbstractControl): Promise<ValidationErrors | null> | Observabl
    e<ValidationErrors | null> => {
46.             return new Promise((resolve, reject) => {
47.                 const userNameValue = control.value.toLowerCase();
```

```
                const userNamesLowerCase = userNames.map(names => names.toLowerCase());
                setTimeout(() => {
                    userNamesLowerCase.includes(userNameValue) ? resolve({ 'isUserNameTaken': true }) : resolve(null);
                }, 5000);
            });
        };
    }

    static isEmailTaken(emails: string[]): ValidatorFn {
        return (control: AbstractControl): Promise<ValidationErrors | null> | Observable<ValidationErrors | null> => {
            const emailValue = control.value.toLowerCase();
            const emailLowerCase = emails.map(names => names.toLowerCase());
            return of(emailLowerCase).pipe(
                delay(1000),
                map((newEmail) => newEmail.includes(emailValue) ? { 'isEmailTaken': true } : null)
            );
        };
    }
}
```

أما التعديلات والإضافات في ملفي app.component.ts وملف app.component.html فهما مشابهين لما قمنا به في المثال على Promise في حقل userName والفرق هنا أننا نطبق هذا المثال على حقل email، لذلك ومنعاً لتكرار سوف اجري هذه التعديلات والإضافات بدون شرح، كالتالي:

ملف app.component.ts

```
import { Component, OnInit } from '@angular/core';

import { FormGroup, FormBuilder, Validators } from '@angular/forms';
import { CustomValidator } from 'src/app/shared/custom.validators';

@Component({
    selector: 'app-root',
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css']
})

export class AppComponent implements OnInit {

    states: string[] = ['ALRiyadh', 'Makkah', 'AlSharqiyah', 'AlQasim'];
    userNames: string[] = ['faisal', 'DivFaisal'];
    form: FormGroup;
    userNameLength: any = '0';
    names: string[] = ['admin', 'administrator'];
    emails: string[] = ['faisal@gmail.com', 'google@gmail.com'];

    messageValidation = {
        'userName': {
```

```
            'required': 'اسم المستخدم مطلوب',
            'pattern': 'اسم المستخدم على الاقل ثلاث خانات',
            'forbiddenNames': 'لايُسمح بتسجيل اسم المستخدم المُدخل',
            'isEnglishLetters': 'لا يُسمح بوجود المسافات او الارقام او الرموز الخاصة او حروف غير الإنجليزية',
            'isUserNameTaken': 'اسم المستخدم غير متاح'
        },
        'email': {
            'required': ' البريد الإلكتروني مطلوب',
            'emailValidation': 'صيغة البريد الإلكتروني غير صحيحة',
            'isEmailTaken': 'البريد الإلكتروني غير متاح'
        },
        'passwordGroup': {
            'passwordValidation': 'كلمة السر غير متطابقة'
        },
        'password': {
            'required': 'كلمة السر مطلوبة',
            'pattern': ' كلمة السر ست خانات ارقام وحروف وعلى الاقل حرف واحد كبير'
        },
        'confirmPassword': {
            'required': ' الحقل مطلوب'
        },
        'gender': {
            'required': 'الحقل مطلوب'
        },
        'addressDate': {
            'required': 'حقل تاريخ إنتهاء أقامة السكن مطلوب'
        },
        'city': {
            'required': 'حقل اسم المدينة مطلوب'
        },
        'state': {
            'required': 'حقل المنطقة مطلوب'
        },
        'zipCode': {
            'required': 'حقل الرمز البريدي مطلوب',
            'pattern': 'الرمز البريدي لابد أن يكون قيمة رقمية من خمس خانات'
        }
    };

    currentMessageValidation = {
        'userName': '',
        'email': '',
        'passwordGroup': '',
        'password': '',
        'confirmPassword': '',
        'gender': '',
        'addressDate': '',
        'city': '',
        'state': '',
        'zipCode': ''
    };

    constructor(private fb: FormBuilder) { }
```

```
76.
77.    ngOnInit() {
78.        this.form = this.fb.group({
79.            userName: [null,
80.                [
81.                    Validators.required,
82.                    Validators.pattern('.{3,}'),
83.                    CustomValidator.forbiddenNames(this.names),
84.                    CustomValidator.isEnglishLetters
85.                ], [CustomValidator.isUserNameTaken(this.userNames)
86.                ]
87.            ],
88.            email: [null,
89.                [
90.                    Validators.required,
91.                    CustomValidator.emailValidation
92.                ], [CustomValidator.isEmailTaken(this.emails)]
93.            ],
94.            passwordGroup: this.fb.group({
95.                password: [null,
96.                    [
97.                        Validators.required,
98.                        Validators.pattern('(?=.*[A-Za-z])(?=.*[A-Z])(?!.* )(?=.*[0-
   9])[A-Za-z0-9d$@].{5,}')
99.                    ]
100.                ],
101.                confirmPassword: [null,
102.                    [
103.                        Validators.required
104.                    ]
105.                ]
106.            }, { validator: CustomValidator.passwordValidation }),
107.            gender: [null, Validators.required],
108.            address: this.fb.group({
109.                addressType: ['permanent'],
110.                addressDate: [null],
111.                city: [null, Validators.required],
112.                state: [null, Validators.required],
113.                zipCode: [null, [Validators.required, Validators.pattern('^[0-
   9]{5}$')]]
114.            })
115.        });
116.
117.        this.userName.valueChanges.subscribe((value: string) => {
118.            this.userNameLength = value.length;
119.        });
120.
121.        this.form.valueChanges.subscribe(data => {
122.            this.loopThroughControls(this.form);
123.        });
124.
125.        this.addressType.valueChanges.subscribe(data => {
126.            this.addressDateValidation(data);
```

```
127.         });
128.     }
129.
130.     addressDateValidation(data: string) {
131.         if (data === 'temporary') {
132.             this.addressDate.setValidators(Validators.required);
133.         } else {
134.             this.addressDate.clearValidators();
135.             this.addressDate.markAsUntouched();
136.             this.addressDate.reset();
137.         }
138.         this.addressDate.updateValueAndValidity();
139.     }
140.
141.     save() {
142.         console.log(this.form);
143.     }
144.
145.     loopThroughControls(formGroup: FormGroup = this.form) {
146.         Object.keys(formGroup.controls).forEach((key: string) => {
147.             const controlName = formGroup.get(key);
148.             this.currentMessageValidation[key] = '';
149.             if (controlName && controlName.invalid && controlName.touched) {
150.                 const messages = this.messageValidation[key];
151.                 for (const controlError in controlName.errors) {
152.                     if (controlError) {
153.                         this.currentMessageValidation[key] +=
154.                             messages[controlError] + ' ';
155.                     }
156.                 }
157.             }
158.             if (controlName instanceof FormGroup) {
159.                 this.loopThroughControls(controlName);
160.             }
161.         });
162.     }
163.
164.     laodData() {
165.         this.form.patchValue({
166.             userName: 'DivFaisal',
167.             email: 'test@test.com',
168.             passwordGroup: {
169.                 password: 'Aa1111',
170.                 confirmPassword: 'Aa1111'
171.             },
172.             gender: 'male',
173.             address: {
174.                 city: 'Riyadh',
175.                 state: 'ALRiyadh',
176.                 zipCode: '87678'
177.             }
178.         });
179.     }
```

```
180.
181.       get userName() {
182.           return this.form.get('userName');
183.       }
184.       get email() {
185.           return this.form.get('email');
186.       }
187.       get password() {
188.           return this.form.get('password');
189.       }
190.       get confirmPassword() {
191.           return this.form.get('confirmPassword');
192.       }
193.       get gender() {
194.           return this.form.get('gender');
195.       }
196.       get address() {
197.           return this.form.get('address');
198.       }
199.       get city() {
200.           return this.form.get('address').get('city');
201.       }
202.       get state() {
203.           return this.form.get('address').get('state');
204.       }
205.       get zipCode() {
206.           return this.form.get('address').get('zipCode');
207.       }
208.       get addressType() {
209.           return this.form.get('address').get('addressType');
210.       }
211.       get addressDate() {
212.           return this.form.get('address').get('addressDate');
213.       }
214.   }
```

راجع الاسطر (19 – 27 – 85)

ملف app.component.html

```
1. <div class="container-fluid">
2.   <div class="card mx-auto col-sm-12 col-md-6 col-lg-6 col-xl-6 pt-sm-4">
3.     <form [formGroup]="form">
4.       <div class="card-header">
5.         <h4 class="text-center">Reactive Forms</h4>
6.       </div>
7.       <div class="card-body">
8.         <!-- أداة إدخال اسم المستخدم -->
9.         <div class="form-group">
10.          <label>User Name</label>
11.          <div class="form-inline">
12.            <input
13.              formControlName="userName"
14.              [ngClass]="{
```

```
                'col-10': true,
                'form-control': true,
                  'is-
  invalid': currentMessageValidation.userName || userName.hasError('isUserNameTaken'),
                'is-valid':
                  !userName.hasError('isUserNameTaken') &&
                  !userName.pending &&
                  userName.value !== null &&
                  userNameLength >= 3
              }"
              (blur)="loopThroughControls()"
              (input)="loopThroughControls()"
            />
            <label class="col-2">{{ userNameLength }}</label>
          </div>
          <!-- جزء التحقق من الصحة الخاص باسم المستخدم -->
          <small class="text-danger" *ngIf="currentMessageValidation.userName">
            {{ currentMessageValidation.userName }}
          </small>
          <div class="alert alert-info col-10" *ngIf="userName.pending">
            جاري التحقق من اتاحة اسم المستخدم
          </div>
        </div>
        <!-- أداة إدخال البريد الإلكتروني -->
        <div class="form-group">
          <label>Email</label>
          <input
            type="email"
            formControlName="email"
            [ngClass]="{
              'form-control': true,
              'is-
  invalid': currentMessageValidation.email || email.hasError('isEmailTaken'),
              'is-valid':
                !email.hasError('isEmailTaken') &&
                !email.pending &&
                email.value !== null &&
                email.value !== '' &&
                !email.hasError('emailValidation')
            }"
            (blur)="loopThroughControls()"
          />
          <!-- جزء التحقق من الصحة الخاص بالبريد الإلكتروني -->
          <small class="text-danger" *ngIf="currentMessageValidation.email">
            {{ currentMessageValidation.email }}
          </small>
          <div class="alert alert-info" *ngIf="email.pending">
            جاري التحقق من إتاحة البريد الإلكتروني
          </div>
        </div>

        <!-- بداية النموذج الفرعي لكلمة السر وإعادة إدخال كلمة السر -->
        <div formGroupName="passwordGroup">
```

```
66.           <!-- أداة إدخال كلمة السر -->
67.           <div class="form-group">
68.             <label>Password</label>
69.             <input
70.               type="password"
71.               formControlName="password"
72.               [ngClass]="{
73.                 'form-control': true,
74.                 'is-invalid': currentMessageValidation.password
75.               }"
76.               (blur)="loopThroughControls()"
77.             />
78.             <!-- جزء التحقق من الصحة الخاص بكلمة السر -->
79.             <small
80.               class="text-danger"
81.               *ngIf="currentMessageValidation.password"
82.             >
83.               {{ currentMessageValidation.password }}
84.             </small>
85.           </div>
86.
87.           <!-- أداة إعادة إدخال كلمة السر -->
88.           <div class="form-group">
89.             <label>Confirm Password</label>
90.             <input
91.               type="password"
92.               formControlName="confirmPassword"
93.               [ngClass]="{
94.                 'form-control': true,
95.                 'is-
   invalid': currentMessageValidation.confirmPassword || currentMessageValidation.password
   Group
96.               }"
97.               (blur)="loopThroughControls()"
98.             />
99.
100.              <!-- جزء التحقق من الصحة الخاص بأداة إعادة إدخال كلمة السر -->
101.              <small
102.                class="text-danger"
103.                *ngIf="currentMessageValidation.confirmPassword || currentMessageValid
   ation.passwordGroup"
104.              >
105.                {{ currentMessageValidation.confirmPassword ?
106.                currentMessageValidation.confirmPassword :
107.                currentMessageValidation.passwordGroup }}
108.              </small>
109.            </div>
110.          </div>
111.          <!-- أداة تحديد نوع الجنس ذكر أو أنثى -->
112.          <label class="pr-2">Gender</label>
113.          <div class="form-check form-check-inline">
114.            <input
115.              type="radio"
```

```
116.              formControlName="gender"
117.              id="maleGender"
118.              value="male"
119.              [ngClass]="{
120.                'form-check-input': true,
121.                'is-invalid': currentMessageValidation.gender
122.              }"
123.              (blur)="loopThroughControls()"
124.            />
125.            <label class="form-check-label" for="gender">Male</label>
126.          </div>
127.          <div class="form-check form-check-inline">
128.            <input
129.              type="radio"
130.              formControlName="gender"
131.              id="femaleGender"
132.              value="femail"
133.              [ngClass]="{
134.                'form-check-input': true,
135.                'is-invalid': currentMessageValidation.gender
136.              }"
137.              (blur)="loopThroughControls()"
138.            />
139.            <label class="form-check-label" for="femaleGender">Femail</label>
140.          </div>
141.          <!--  الجزء الخاص بتحقق من الصحة لأداة تحديد نوع الجنس  -->
142.          <small class="text-danger" *ngIf="currentMessageValidation.gender">
143.            {{ currentMessageValidation.gender }}
144.          </small>
145.
146.          <!--  بداية النموذج الفرعي  -->
147.          <fieldset class="scheduler-border" formGroupName="address">
148.            <legend class="scheduler-border">Address Information</legend>
149.
150.            <label class="pr-2" style="margin-bottom: 0px">Address Type:</label>
151.            <br />
152.            <div class="form-check form-check-inline">
153.              <input
154.                class="form-check-input"
155.                type="radio"
156.                formControlName="addressType"
157.                id="temporary"
158.                value="temporary"
159.              />
160.              <label class="form-check-label" for="temporary">Temporary</label>
161.            </div>
162.            <div class="form-check form-check-inline">
163.              <input
164.                class="form-check-input"
165.                type="radio"
166.                formControlName="addressType"
167.                id="permanent"
168.                value="permanent"
```

```
169.             />
170.             <label class="form-check-label" for="permanent">Permanent</label>
171.           </div>
172.           <input
173.             type="date"
174.             formControlName="addressDate"
175.             [class.has-error]="currentMessageValidation.addressDate"
176.             *ngIf="addressType.value === 'temporary'"
177.             (blur)="addressDateValidation('temporary')"
178.           />
179.           <div>
180.             <small
181.               class="text-danger"
182.               *ngIf="currentMessageValidation.addressDate"
183.             >
184.               {{ currentMessageValidation.addressDate }}
185.             </small>
186.           </div>
187.
188.           <!-- أداة إدخال اسم المدينة -->
189.           <div class="form-group pt-4">
190.             <label>City</label>
191.             <input
192.               formControlName="city"
193.               [ngClass]="{
194.                 'form-control': true,
195.                 'is-invalid': currentMessageValidation.city
196.               }"
197.               (blur)="loopThroughControls()"
198.             />
199.             <!-- جزء التحقق من الصحة لأداة إدخال اسم المدينة -->
200.             <small class="text-danger" *ngIf="currentMessageValidation.city">
201.               {{ currentMessageValidation.city }}
202.             </small>
203.           </div>
204.           <!-- أداة اختيار اسم المنطقة او الولاية -->
205.           <div class="form-group">
206.             <label>State</label>
207.             <select
208.               formControlName="state"
209.               [ngClass]="{
210.                 'form-control': true,
211.                 'is-invalid': currentMessageValidation.state
212.               }"
213.               (blur)="loopThroughControls()"
214.             >
215.               <option selected [ngValue]="null">Choose...</option>
216.               <option *ngFor="let item of states" [value]="item">
217.                 {{ item }}
218.               </option>
219.             </select>
220.             <!-- جزء التحقق من الصحة لأداة أختيار اسم المنطقة -->
221.             <small class="text-danger" *ngIf="currentMessageValidation.state">
```

```
222.                 {{ currentMessageValidation.state }}
223.               </small>
224.             </div>
225.             <!-- أداة إدخال الرمز البريدي -->
226.             <div class="form-group">
227.               <label>Zip Code</label>
228.               <input
229.                 formControlName="zipCode"
230.                 [ngClass]="{
231.                   'form-control': true,
232.                   'is-invalid': currentMessageValidation.zipCode
233.                 }"
234.                 (blur)="loopThroughControls()"
235.               />
236.               <!-- جزء التحقق من الصحة لأداة إدخال الرمز البريدي -->
237.               <small class="text-danger" *ngIf="currentMessageValidation.zipCode">
238.                 {{ currentMessageValidation.zipCode }}
239.               </small>
240.             </div>
241.           </fieldset>
242.         </div>
243.
244.         <!-- بداية أدوات الأزرار -->
245.         <div class="card-footer">
246.           <button class="btn btn-primary" (click)="save()">Save</button>
247.           <button class="btn btn-primary ml-3" (click)="laodData()">
248.             Load Data
249.           </button>
250.         </div>
251.       </form>
252.     </div>
253.   </div>
```

راجع الاسطر (من 37 إلى 62)

والآن لنشاهد التعديلات على النموذج من خلال المتصفح:

User Name

faisal 6

جاري التحقق من اتاحة اسم المستخدم

Email

faisal@gmail.com

جاري التحقق من إتاحة البريد الإلكتروني

Password

Confirm Password

Gender Male Femail

Address Information

Address Type:

Temporary Permanent

City

State

Choose...

Zip Code

User Name

faisal ×

6

اسم المستخدم غير متاح

Email

faisal@gmail.com ×

البريد الإلكتروني غير متاح

Password

Confirm Password

Gender ○ Male ○ Femail

Address Information

Address Type:

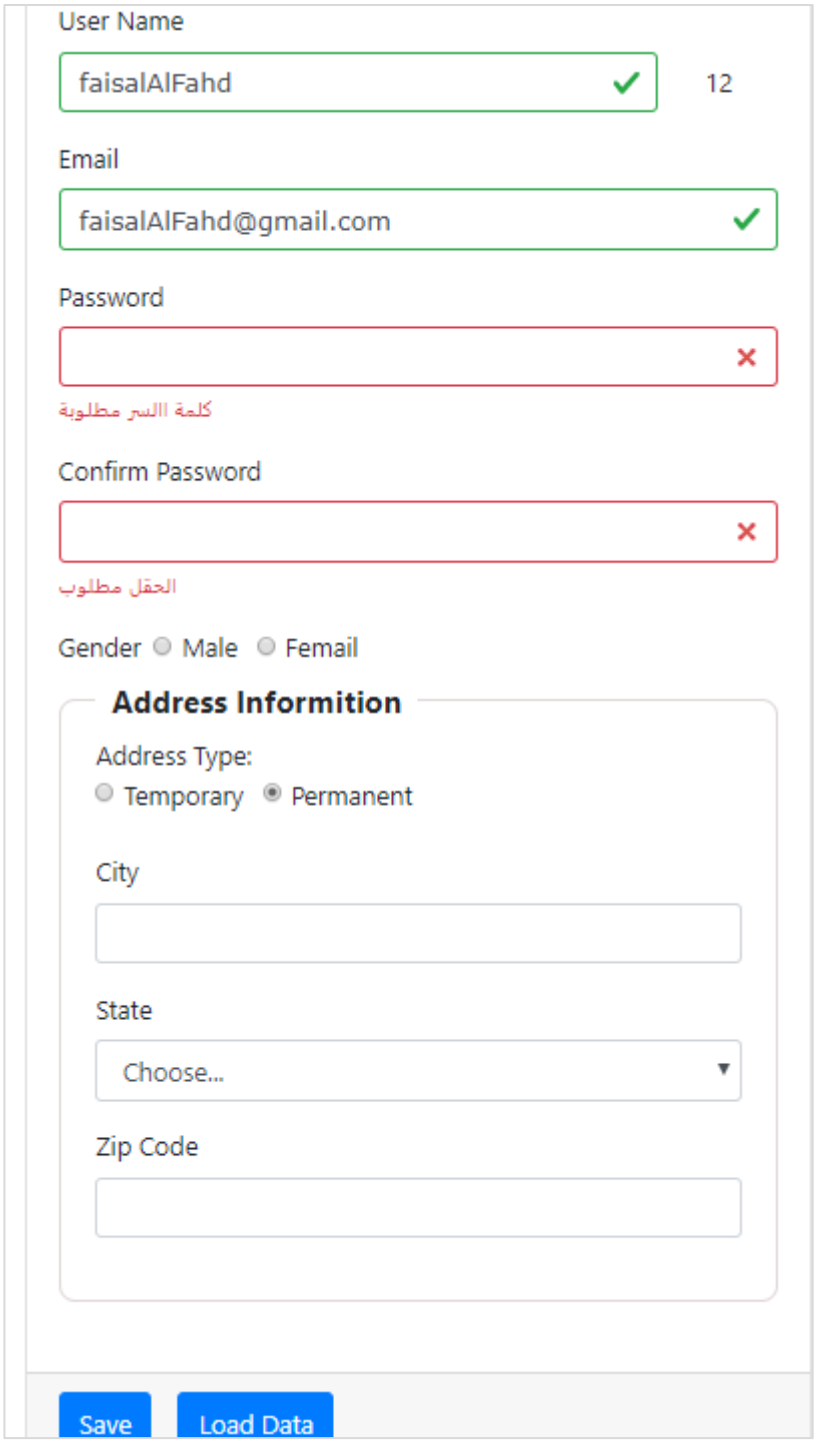
○ Temporary ◉ Permanent

City

State

Choose...

Zip Code


User Name
faisalAlFahd
12
Email
faisalAlFahd@gmail.com
Password
كلمة السر مطلوبة
Confirm Password
الحقل مطلوب
Gender Male Femail
Address Information
Address Type:
Temporary Permanent
City
State
Choose...
Zip Code
Save
Load Data

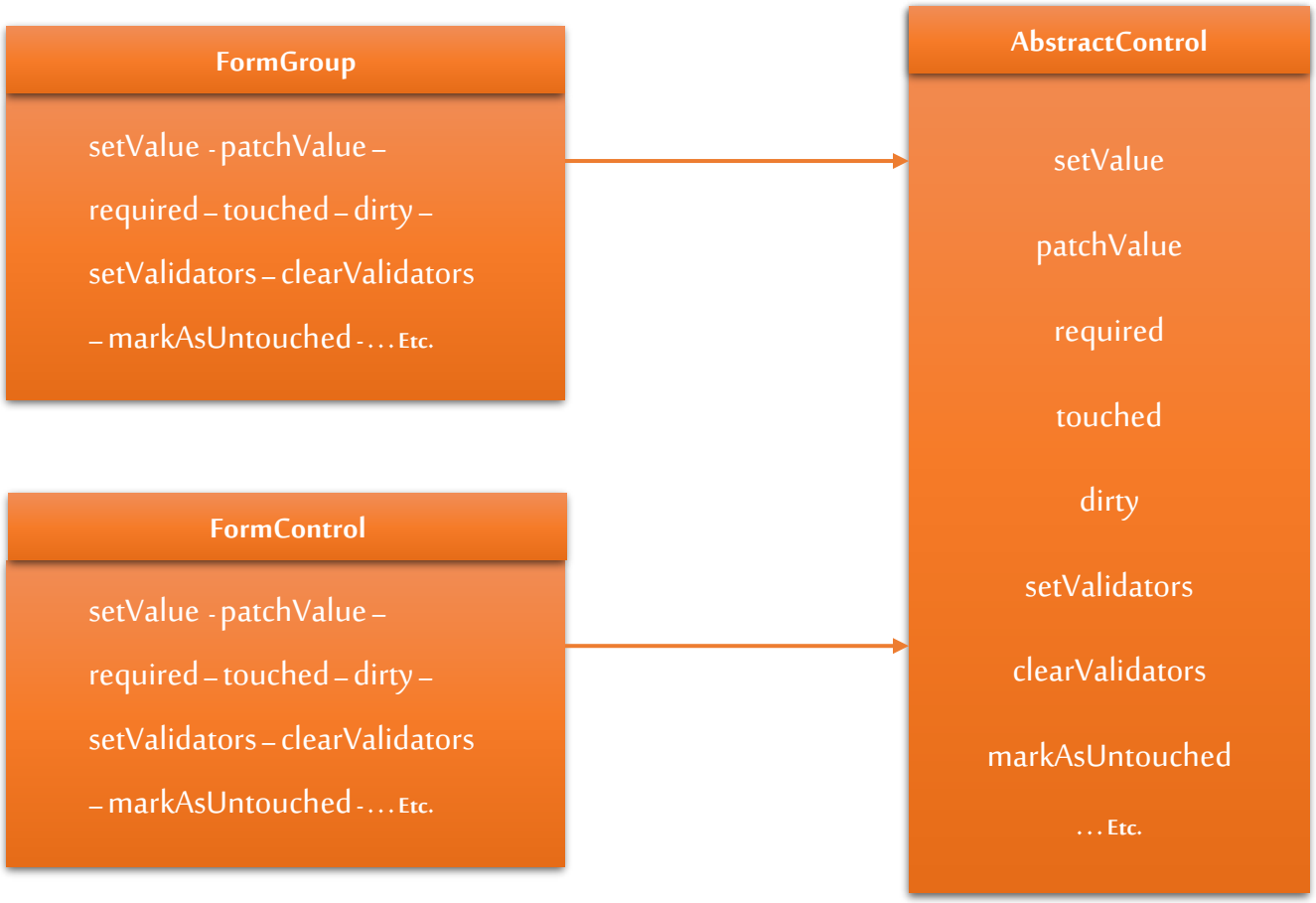
الفصل الخامس

النماذج الديناميكية في Reactive Forms

1.5. المقدمة:

كما تطرقنا سابقاً ان angular forms يُقدم لنا كلاسين هما FormGroup و FormControl وكلا هذين الكلاسين يرثان من الكلاس الأب AbstractControl وهو من النوع abstract – لا نستطيع أن نعمل منه instance لكائن object وانما نستطيع أن نجعل كلاسات ترث منه فقط – حيث أن هذا الكلاس الأب يمتلك مجموعة من الدوال methods والخصائص properties التي تسهل لنا التعامل مع النماذج من النوع angular reactive forms وبما أن هذين الكلاسين يرثان من هذا الكلاس فمن الطبيعي أنهما ايضاً اصبحا يمتلكان حق الوصول إلى جميع هذه الدوال والخصائص، مع العلم ان ما قيل سابقاً ليس له علاقة بالخصوص بي Angular وانما هو من أسس OOP.

ومن أمثلة هذه الدوال والخصائص والتي تعاملنا معها سابقاً: setValue – patchValue – required – touched – dirty – markAsUntouched – clearValidators – setValidators – reset – وغيرها الكثير، ويمكن أن نمثل العلاقة بين هذه الكلاسات، كما في الشكل التالي:



وهذين الكلاسين نستعملها كما تعاملنا معهم سابقاً في تعريف النموذج الرئيسي عن طريق الكلاس FormGroup أو أي نموذج فرعي باستخدام الinterface ذات الاسم FormBuilder حيث عندما نكتب fb.group({ }) فنحن بذلك نشير إلى الكلاس FormGroup (fb هي المتغير الذي عرفنا عن طريقه FormBuilder كما شرحنا سابقاً) اما الكلاس FormControl فتعرف بشكل تلقائي عندما نعرف الأدوات داخل هذا النموذج ونربطها بأدوات النموذج في ملف HTML.

والنموذج الفرعي هو جزء من النموذج الرئيسي ويمكن هو ايضاً ان يحتوي بداخله على نموذج فرعي آخر وهكذا.

هذي كانت مراجعة سريعة لهذين الكلاسين وعلاقتهما مع بعضهما البعض، ولكن هذين الكلاسين لا يتعاملان مع النماذج الديناميكية التي تُرسم أدواتها أثناء وقت تشغيل النموذج كأن يكون هنالك زر وكلما ضغط المستخدم على هذا الزر تضاف له أداة معينة ليقوم بإدخال البيانات التي تلبي احتياجاته، لذلك استدعى الأمر من فريق عمل angular إلى إنشاء كلاس ثالث تحت اسم FormArray وهذا الكلاس هو ايضاً يرث من الكلاس AbstractControl وبذلك هو ايضاً يمتلك حق الوصول لجميع الدوال والخصائص سابقة الذكر بالإضافة إلى مجموعة من الدوال الخاصة به التي تساعد وتسهل في تعامله مع النماذج الديناميكية، ويمكن حصر أشهر هذه الدوال في الجدول التالي:

الدالة	الوصف
push	إضافة أداة في آخر مصفوفة الأدوات
insert	إضافة أداة في مكان محدد في مصفوفة الأدوات
removeAt	حذف أداة من خلال مكان محدد في مصفوفة الأدوات
setControl	استبدال أداة بأداة أخرى في مكان محدد في مصفوفة الأدوات
at	ارجاع أداة معينة في مكان محدد في مصفوفة الأدوات

وعموماً الأكثر استخداماً في هذه الدوال هو push و removeAt، ونلاحظ أن FormArray يعامل الأدوات على انها مصفوفة نصية تحتوي على أسماء هذه الأدوات التي نريد اضافتها إلى النموذج بعكس FormGroup الذي يتعامل معها على انها كائن object.

وآخر شيء نريد التنبيه له أن FormArray يمكن أن تكون جزء FormGroup للنموذج الرئيسي أو جزء من FormGroup الذي يشير إلى النموذج الفرعي او حتى يمكن ان يكون جزء من نموذج فرعي لنموذج فرعي آخر وهكذا، وايضاً ممكن ان يكون FormArray يحتوي بداخله FormArray أخرى او حتى يمكن أن يحتوي بداخله FormGroup فرعي تحتوي بداخله على مجموعة أدوات نُريد أن تنشأ هذه الأدوات بشكل ديناميكي اثناء وقت التشغيل لذلك نجمعها في FormGroup واحد ونتعامل معه بشكل اسهل من التعامل مع كل أداة على حدا، والسبب في السماح في هذا التداخل والتشابك أننا في حقيقة الأمر كما اشرنا قبل قليل نتعامل مع مصفوفات FormArray وكائنات FormGroup، فالكائن يمكن أن يحتوي بداخله على مجموعة من المصفوفات والكائنات وكل مصفوفة أو كائن فرعي ممكن أن تحتوي بداخلها ايضاً على مصفوفات وكائنات فرعية وهكذا. اما الأمر الآخر لهذا التشابك والتداخل هو لتلبية جميع احتياجات المبرمجين وخصوصاً في النماذج المعقدة والمتداخلة، وأخيراً أحببت ان أقول عزيزي القارئ لا تقلق فسوف أحاول تبسيطها وتوضيحها من خلال الأمثلة التوضيحية التي سوف تسهل لك بإذن الله طريقة التعامل مع النماذج الديناميكية.

أما الآن وبعد هذه المقدمة النظرية وتوضيح المفاهيم سوف ننتقل إلى الأمثلة التي سوف تساعدنا بإن الله على فهم واستيعاب هذا النوع من النماذج.

2.5. المثال الأول:

في هذا المثال نُريد أن نُضيف نموذج فرعي يحتوي على أداتين لإدخال رقم الهاتف الجوال ورقم الهاتف الثابت مع وجود زر يسمح للمستخدم إضافة نماذج فرعية أخرى تحتوي على مربعات إدخال بشكل ديناميكي على النموذج مع كل ضغطة على الزر، كما انه يُتاح للمستخدم حذف أي من هذه النماذج الفرعية بشكل كامل مع أدواتها، ويختفي زر الحذف في حال كان هنالك نموذج فرعي واحد فقط، ولجعل المثال أكثر احترافية نريد من التحقق من الصحة أن يكون ديناميكي ومشروط بنفس الوقت بمعنى أن يكون لكل نموذج فرعي أداتين radio يحدد المستخدم أي من أداتين الإدخال هي وسيلة الاتصال الرئيسية رقم الهاتف او رقم الجوال وفي حال اختيار رقم الهاتف يتم تطبيق التحقق من الصحة على أداة إدخال الهاتف ولا يتم تطبيقها على أداة أدخال رقم الجوال وبنفس الطريقة في حال اختيار رقم الجوال، مع العلم أن لكل نموذج فرعي أداتي radio الخاصة به ويتم إنشائها ديناميكياً مع النموذج الفرعي، وبنفس الوقت لكل نموذج فرعي تم إنشاءه ديناميكياً التحقق من الصحة الخاص به فمثلاً النموذج الفرعي الأول يتم التحقق من رقم الهاتف والثاني من رقم الجوال والثالث من الجوال ايضاً وهكذا بحيث يعطي مرونة للمستخدم أكثر في تحديد اختياراته وتلبية احتياجاته، مع العلم إن أداتي radio مختاره بشكل افتراضي على ان وسيلة الاتصال الرئيسية هي رقم الجوال ويتم التحقق من الصحة بشكل افتراضي على أداة إدخال رقم الجوال في حال لم يقم المستخدم بتعديلها إلى رقم الهاتف.

الحل: -

هذا المثال شامل وجيد في حال فهمه ويمكن أخذ الفكرة وتطبيقها في افكار أخرى، أما من ناحية حل هذا المثال فبما انه نموذج ديناميكي فنحتاج إلى FormArray وداخله FormGroup وداخله ثلاث أدوات FormControl الأداة الأولى لإدخال رقم الجوال والثانية لإدخال رقم الهاتف والثالثة أداتين radio، لذلك أولاً نقوم بإنشاء Markup أو كود HTML لهذه النموذج الفرعي في ملف app.component.html، وسوف اضيف هذا Markup تحت النموذج الفرعي السابق الخاص بإدخال بيانات العنوان، كالتالي:

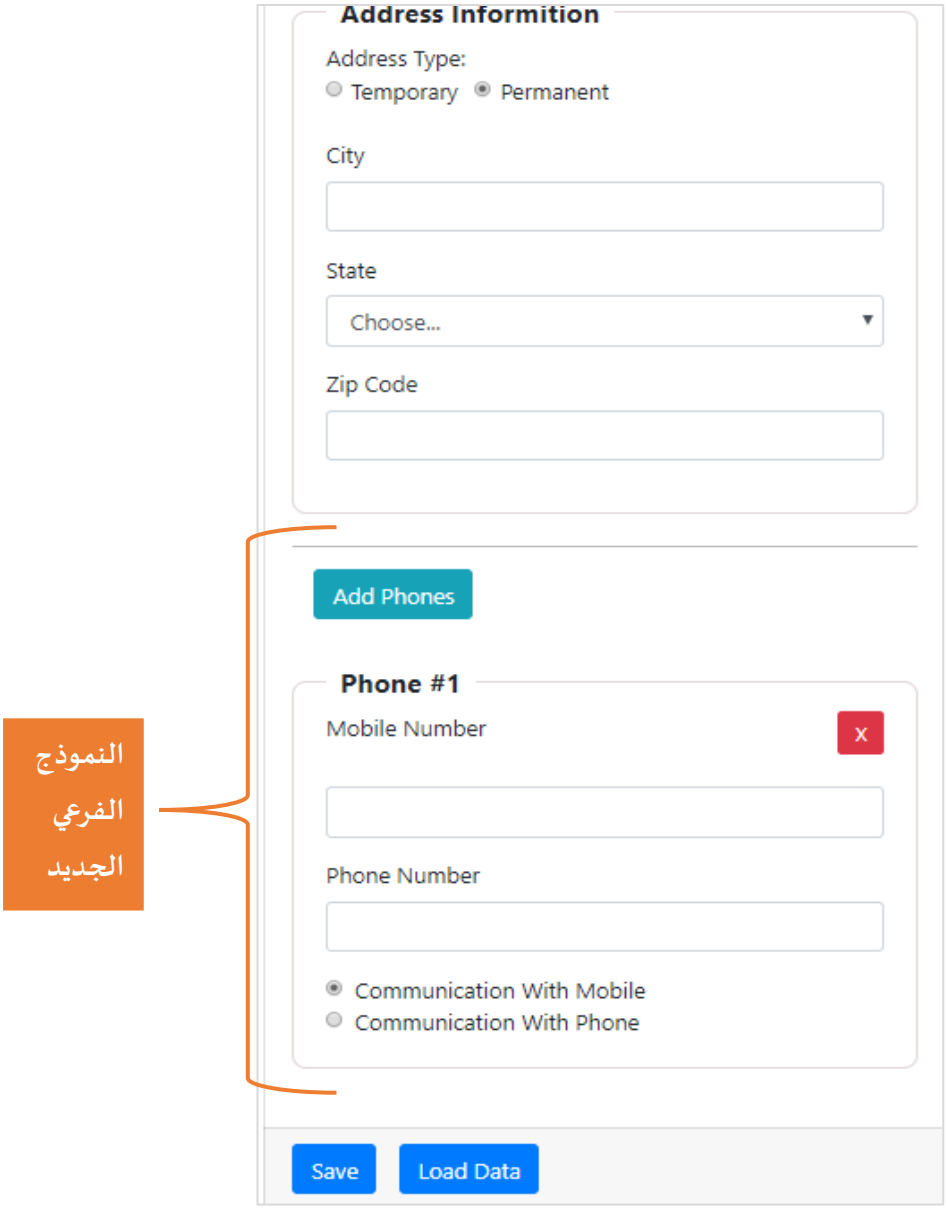
جزء من ملف app.component.html

```
1. <hr style="border: 1px solid silver" />
2. <div class="form-group">
3.   <div class="col-md-offset-2 col-md-4">
4.     <button type="button" class="btn btn-info mb-3">Add Phones</button>
5.   </div>
6. </div>
7. <fieldset class="scheduler-border">
8.   <legend class="scheduler-border">Phone #1</legend>
9.   <div>
10.    <div>
11.      <button class="btn btn-danger btn-xs float-right mb-4 w-100">X</button>
12.    </div>
13.    <div class="form-group">
14.      <label for="MobileNumber">Mobile Number</label>
15.      <input type="tel" id="MobileNumber" [ngClass]="{'form-control': true}" />
16.    </div>
17.    <div class="form-group">
```

```
      <label for="PhoneNumber">Phone Number</label>
      <input type="tel" id="PhoneNumber" [ngClass]="{'form-control': true}" />
    </div>
    <div class="form-check">
      <input
        class="form-check-input"
        type="radio"
        id="mobileCommunication"
        value="mobileCommunication"
      />
      <label class="form-check-label" for="mobileCommunication">
        Communication With Mobile
      </label>
    </div>
    <div class="form-check">
      <input
        class="form-check-input"
        type="radio"
        id="phoneCommunication"
        value="phoneCommunication"
      />
      <label class="form-check-label" for="phoneCommunication">
        Communication With Phone
      </label>
    </div>
  </div>
</fieldset>
```

قيمة أداة radio الخاصة بتحديد إذا الاتصال الرئيسي هو الجوال، وسوف نستخدمها عند إنشاء النموذج برمجياً لتكون مختارة بشكل افتراضي

ونتيجة إضافة هذا الكود، هو التالي:



بعد كتابة الأكواد في ملف HTML نقوم الآن ببنائه برمجياً في ملف app.component.ts، كما كنا نفعل مع الأدوات والنماذج السابقة، ولكن هنا نستخدم FormArray، كالتالي:

app.component.ts جزء من ملف

```
1. ngOnInit() {
2.     this.form = this.fb.group({
3.         userName: [
4.             null,
5.             [
6.                 Validators.required,
7.                 Validators.pattern('.{3,}'),
8.                 CustomValidator.forbiddenNames(this.names),
9.                 CustomValidator.isEnglishLetters
10.            ],
11.            [
12.                CustomValidator.isUserNameTaken(this.userNames)
13.            ]
14.        ],
15.        email: [
```

```
            null,
            [
                Validators.required,
                CustomValidator.emailValidation
            ],
            [
                CustomValidator.isEmailTaken(this.emails)
            ]
        ],
        passwordGroup: this.fb.group({
            password: [null,
                [
                    Validators.required,
                    Validators.pattern(
                        '(?=.*[A-Za-z])(?=.*[A-Z])(?!.* )(?=.*[0-9])[A-Za-z0-
9d$@].{5,}'
                    )
                ]
            ],
            confirmPassword: [null,
                [
                    Validators.required
                ]
            ]
        },
            {
                validator: CustomValidator.passwordValidation
            }
        ),
        gender: [null, Validators.required],
        address: this.fb.group({
            addressType: ['permanent'],
            addressDate: [null],
            city: [null, Validators.required],
            state: [null, Validators.required],
            zipCode: [
                null,
                [
                    Validators.required, Validators.pattern('^[0-9]{5}$')
                ]
            ]
        }),
        phones: this.fb.array([

        ])
    });
}

```

هنا نقوم بكتابة جميع النماذج الفرعية والأدوات التي نريد إنشائها بشكل ديناميكي

أنشئنا نموذج فرعي باسم phones وهو عبارة عن مصفوفة ونوعه FormArray عن طريق FormBuilder التي عملنا لها حقن في المتغير fb.

وداخل هذه المصفوفة phones نُريد أن ننشأ نموذج فرعي نوعه FormGroup ويحتوي على ثلاث أدوات بشكل ديناميكي، ونقوم بهذا الأمر كالتالي:

app.component.ts جزء من ملف

```
1. ngOnInit() {
2.     this.form = this.fb.group({
3.         userName:
4.             [
5.                 null,
6.             [
7.                 Validators.required,
8.                 Validators.pattern('.{3,}'),
9.                 CustomValidator.forbiddenNames(this.names),
10.                CustomValidator.isEnglishLetters
11.            ],
12.            [
13.                CustomValidator.isUserNameTaken(this.userNames)
14.            ]
15.        ],
16.        email:
17.            [
18.                null,
19.            [
20.                Validators.required,
21.                CustomValidator.emailValidation
22.            ],
23.            [
24.                CustomValidator.isEmailTaken(this.emails)
25.            ]
26.        ],
27.        passwordGroup: this.fb.group({
28.            password:
29.                [
30.                    null,
31.                [
32.                    Validators.required,
33.                    Validators.pattern(
34.                        '(?=.*[A-Za-z])(?=.*[A-Z])(?!.* )(?=.*[0-9])[A-Za-z0-
   9d$@].{5,}'
35.                    )
36.                ]
37.            ],
38.            confirmPassword:
39.                [
40.                    null,
41.                [
42.                    Validators.required
43.                ]
```

```
            ]
        },
            {
                validator: CustomValidator.passwordValidation
            }
        ),
        gender: [null, Validators.required],
        address: this.fb.group({
            addressType: ['permanent'],
            addressDate: [null],
            city: [null, Validators.required],
            state: [null, Validators.required],
            zipCode: [null, [Validators.required, Validators.pattern('^[0-9]{5}$')]]
        }),
        phones: this.fb.array([
            this.fb.group({
                mobile: [null],
                phone: [null],
                mainCommunication: ['mobileCommunication']
            })
        ])
    });
}

```

نلاحظ اضفنا النموذج الفرعي من النوع FormGroup والذي نُريد أن يتم أنشائه ديناميكياً ولم نعطه اسم لأننا نريد أن يكون الأسم ديناميكي، كما سوف نشرحه بعد قليل، ويحتوي بداخله على ثلاث أدوات الأداة الأولى تحت اسم mobile وسوف نربطها بأداة الإدخال الخاصة برقم الجوال، والأداة الثانية phone وسوف نربطها بأداة الإدخال الخاصة بإدخال رقم الهاتف الثابت وكلا هاتين الأداتين لم نعطيهما قيمة افتراضية وانما قيمهما null، أما الأداة الأخيرة باسم mainCommunication وسوف نربطها بأداتي radio وأعطيناها قيمة افتراضية mobileCommunication وهي قيمة أداة radio الأولى التي تحدد إذا وسيلة الاتصال الرئيسية هي رقم الجوال لأننا نريد أن تكون مختارة بشكل افتراضي عند إنشاء النماذج الفرعية بشكل ديناميكي، ولكن هنا ملاحظة بسيطة وهي أننا نريد كتابة كود النموذج الفرعي التي اضفناه قبل قليل في دالة منفصلة لأننا سوف نستخدم هذه الكود في أكثر من مكان في component وليكن اسمه هذه الدالة groupPhones() وتُرجع FormGroup، ونكتب هذه الدالة بعد الدالة ngOnInit التي تحتوي على اكواد إنشاء النموذج برمجياً، كالتالي:

جزء من ملف app.component.ts

```
ngOnInit() {
    this.form = this.fb.group({
        userName:[null,
                [
                    Validators.required,
                    Validators.pattern('.{3,}'),
                    CustomValidator.forbiddenNames(this.names),
                    CustomValidator.isEnglishLetters
                ],
                [
                    CustomValidator.isUserNameTaken(this.userNames)
```

```
                ]
            ],
        email:[null,
                [
                    Validators.required,
                    CustomValidator.emailValidation
                ],
                [
                    CustomValidator.isEmailTaken(this.emails)
                ]
            ],
        passwordGroup: this.fb.group({
            password:[null,
                    [
                        Validators.required,
                        Validators.pattern(
                            '(?=.*[A-Za-z])(?=.*[A-Z])(?!.* )(?=.*[0-9])[A-Za-z0-
    9d$@].{5,}'
                        )
                    ]
                ],
            confirmPassword:[null,
                    [
                        Validators.required
                    ]
                ]
        },
            {
                validator: CustomValidator.passwordValidation
            }
        ),
        gender: [null, Validators.required],
        address: this.fb.group({
            addressType: ['permanent'],
            addressDate: [null],
            city: [null, Validators.required],
            state: [null, Validators.required],
            zipCode: [null, [Validators.required, Validators.pattern('^[0-9]{5}$')]]
        }),
        phones: this.fb.array([this.groupPhones()])
    });
}

groupPhones(): FormGroup {
    return this.fb.group({
        mobile: [null],
        phone: [null],
        mainCommunication: ['mobileCommunication']
    });
}
```

الدالة الجديدة

وبما أن الاختيار الافتراضي لتحقق من الصحة هو رقم الجوال إذن لنضيف لرقم الجوال نوعين من التحقق من الصحة الأولى required والثاني pattern لتأكد أن القيم المدخلة فقط أرقام، كالتالي:

app.component.ts جزء من ملف

```
1. groupPhones(): FormGroup {
2.   return this.fb.group({
3.       mobile:
4.           [
5.               null,
6.               [
7.                   Validators.required,
8.                   Validators.pattern(
9.                     '(?:NaN|-?(?:(?:\\d+|\\d*\\.\\d+)(?:[E|e][+|-]?\\d+)?|Infinity))'
10.                  )
11.              ]
12.          ],
13.      phone:
14.          [
15.              null
16.          ],
17.      mainCommunication:
18.          [
19.              'mobileCommunication'
20.          ]
21.  });
22.}
```

الدالة الجديدة بعد التعديل

وبنفس الوقت نقوم بإنشاء getter للاسم البرمجي phones عن طريق الدالة get كما كنا نفعل سابقاً، كالتالي:

app.component.ts جزء من ملف

```
1. get phones() {
2.     return this.form.get('phones') as FormArray;
3. }
```

الفرق الوحيد هنا أننا استخدمنا as FormArray كأننا نخبر angular بأن هذه الدالة ذات الاسم phones هي من النوع FormArray لكي نستطيع الوصول إلى الدوال الخاصة بهذا النوع مثل push و removeAt ..الخ التي ذكرناها في مقدمة كلامنا عن النماذج الديناميكية.

الآن لننشأ دالتين الدالة الأولى تقوم بإضافة النموذج الفرعي بشكل ديناميكي إلى المصفوفة phones الاسم البرمجي للنموذج الفرعي الذي انشأناه سابقاً وليكن اسمها addPhones( )، والدالة الثانية مهمتها حذف النموذج الفرعي وليكن اسمها removePhones( ) وتستقبل باراميتر واحد وهو index للنموذج الفرعي داخل المصفوفة phones، كالتالي:

app.component.ts جزء من ملف

```
62.ngOnInit() {
63.    this.form = this.fb.group({
64.        userName:[null,
65.                [
66.                    Validators.required,
```

```
                    Validators.pattern('.{3,}'),
                    CustomValidator.forbiddenNames(this.names),
                    CustomValidator.isEnglishLetters
                ],
                [
                    CustomValidator.isUserNameTaken(this.userNames)
                ]
            ],
        email:[null,
                [
                    Validators.required,
                    CustomValidator.emailValidation
                ],
                [
                    CustomValidator.isEmailTaken(this.emails)
                ]
            ],
        passwordGroup: this.fb.group({
            password:[null,
                    [
                        Validators.required,
                        Validators.pattern(
                            '(?=.*[A-Za-z])(?=.*[A-Z])(?!.* )(?=.*[0-9])[A-Za-z0-
9d$@].{5,}'
                        )
                    ]
                ],
            confirmPassword:[null,
                    [
                        Validators.required
                    ]
                ]
        },
            {
                    validator: CustomValidator.passwordValidation
                }
            ),
            gender: [null, Validators.required],
            address: this.fb.group({
                addressType: ['permanent'],
                addressDate: [null],
                city: [null, Validators.required],
                state: [null, Validators.required],
                zipCode: [null, [Validators.required, Validators.pattern('^[0-9]{5}$')]]
            }),
            phones: this.fb.array([this.groupPhones()])
        });
  }

  groupPhones(): FormGroup {
    return this.fb.group({
        mobile:
            [
```

```
                null,
                [
                    Validators.required,
                    Validators.pattern(
                      '(?:NaN|-?(?:(?:\\d+|\\d*\\.\\d+)(?:[E|e][+|-
   ]?\\d+)?|Infinity))'
                    )
                ]
            ],
        phone:
            [
                null
            ],
        mainCommunication:
            [
                'mobileCommunication'
            ]
    });
  }

  addPhones() {
      this.phones.push(this.groupPhones());
  }

  removePhones(index: number) {
      this.phones.removeAt(index);
  }

```

راجع الاسطر (من 138 إلى 144)

بذلك نكون أنهينا اغلب الأكواد في ملف app.component.ts، وسوف ننتقل إلى ملف app.component.html لإجراء بعض التعديلات، وسوف اكتب الكود وأعلق على كل تعديل مهم، كالتالي:

جزء من ملف app.component.html

```
<!--بداية النموذج الفرعي الديناميكي-->
<hr style="border: 1px solid silver" />
<div class="form-group">
  <!--زر إضافة نموذج فرعي-->
  <div class="col-md-offset-2 col-md-4">
    <button type="button" class="btn btn-info mb-3" (click)="addPhones()">
      Add Phones
    </button>
  </div>
</div>
<!--phones المتمثلة في المصفوفة FormArray بداية النموذج الفرعي-->
<fieldset
  class="scheduler-border"
  formArrayName="phones"
  *ngFor="let phoneArray of phones.controls; let i = index"
>
  <legend class="scheduler-border">Phone #{{ i + 1 }}</legend>
```

```
18.  <!--FormGroup بداية النموذج الفرعي التي يتم إنشائه ديناميكياً-->
19.  <div [formGroupName]="i">
20.    <!--بداية زر حذف لكل نموذج فرعي يتم إنشائه ديناميكياً-->
21.    <div class="form-group">
22.      <button
23.        type="button"
24.        class="btn btn-danger btn-xs float-right mb-4 w-100"
25.        (click)="removePhones(i)"
26.        *ngIf="phones.length > 1"
27.      >
28.        X
29.      </button>
30.    </div>
31.    <!--بداية أداة أدخال رقم الجوال-->
32.    <div class="form-group">
33.      <label [attr.for]="'MobileNumber' + i">Mobile Number</label>
34.      <input
35.        type="tel"
36.        [id]="'MobileNumber' + i"
37.        formControlName="mobile"
38.        [ngClass]="{'form-control': true}"
39.      />
40.    </div>
41.    <!--بداية أداة إدخال رقم الهاتف الثابت-->
42.    <div class="form-group">
43.      <label [attr.for]="'PhoneNumber' + i">Phone Number</label>
44.      <input
45.        type="tel"
46.        [id]="'PhoneNumber' + i"
47.        formControlName="phone"
48.        [ngClass]="{'form-control': true}"
49.      />
50.    </div>
51.    <!--radio بداية أداتي-->
52.    <!--أداة الـ radio الأولى التي إذا كانت مختارة يتم التحقق من الصحة لأداة إدخال رقم الجوال-->
53.    <div class="form-check">
54.      <input
55.        class="form-check-input"
56.        type="radio"
57.        formControlName="mainCommunication"
58.        [id]="'mobileCommunication' + i"
59.        value="mobileCommunication"
60.      />
61.      <label class="form-check-label" [attr.for]="'mobileCommunication' + i">
62.        Communication With Mobile
63.      </label>
64.    </div>
65.    <!--أداة الـ radio الثانية التي إذا كانت مختارة يتم التحقق من الصحة لأداة إدخال رقم الهاتف الثابت-->
66.    <div class="form-check">
67.      <input
68.        class="form-check-input"
69.        type="radio"
70.        formControlName="mainCommunication"
```

```
71.         [id]="'phoneCommunication' + i"
72.         value="phoneCommunication"
73.       />
74.       <label class="form-check-label" [attr.for]="'phoneCommunication' + i">
75.         Communication With Phone
76.       </label>
77.     </div>
78.   </div>
79. </fieldset>
```

راجع السطر 6 (إضافة الدالة addPhones لحدث click بحيث يتم تنفيذها كلما ضغط المستخدم على إضافة هواتف جديدة او بطريقة أخرى نماذج فرعية ديناميكية جديدة، ولفهم أعمق عن Event Binding الرجاء مراجعة الكتاب الثاني من هذه السلسلة).

راجع السطر 14 (ربط الاسم البرمجي الذي أنشأناه phones بالحاوية التي تحوي بداخلها على النموذج الفرعي والأدوات التي يتم إنشائها ديناميكياً)

راجع السطر 15 (بعد الربط نقوم بعمل loop على FormArray والمتمثلة في phones حيث نبحث علن controls التي بداخلها وفي مثالنا هنا هي عبارة عن النموذج الفرعي FormGroup ويخزن كل نموذج فرعي داخل المصفوفة في المتغير phoneArray وبنفس الوقت نخزن index لكل نموذج فرعي في المصفوفة phones داخل متغير وليكن اسمه i، ولفهم أعمق عن هذا Directive الرجاء مراجعة الكتاب الثالث من هذه السلسلة Angular Pipes and Directives)

راجع السطر 19 (نضع اسم لنموذج الفرعي FormGroup الديناميكي بحيث يكون متغير بحسب قيمة المتغير i الذي تم انشاءه في السطر 15، بحيث يكون الأول صفر والثاني واحد والثالث اثنان وهكذا)

راجع السطر 25 (نضيف دالة حذف النموذج الفرعي بناءً على index الخاص به والمخزن في المتغير i حيث أن الـ i يتم تخزين فيها موقع (index) النموذج الفرعي FormGroup داخل المصفوفة FormArray ذات الاسم phones بحيث لو تم تخليق ثلاث نماذج بشكل ديناميكي يكون النموذج الأول قيمة i الخاصة به ٠ والثاني ١ والثالث ٢ وبذلك يستطيع angular معرفة أي نموذج حدده المستخدم لحذفه بناءً على قيمة i الخاصة به)

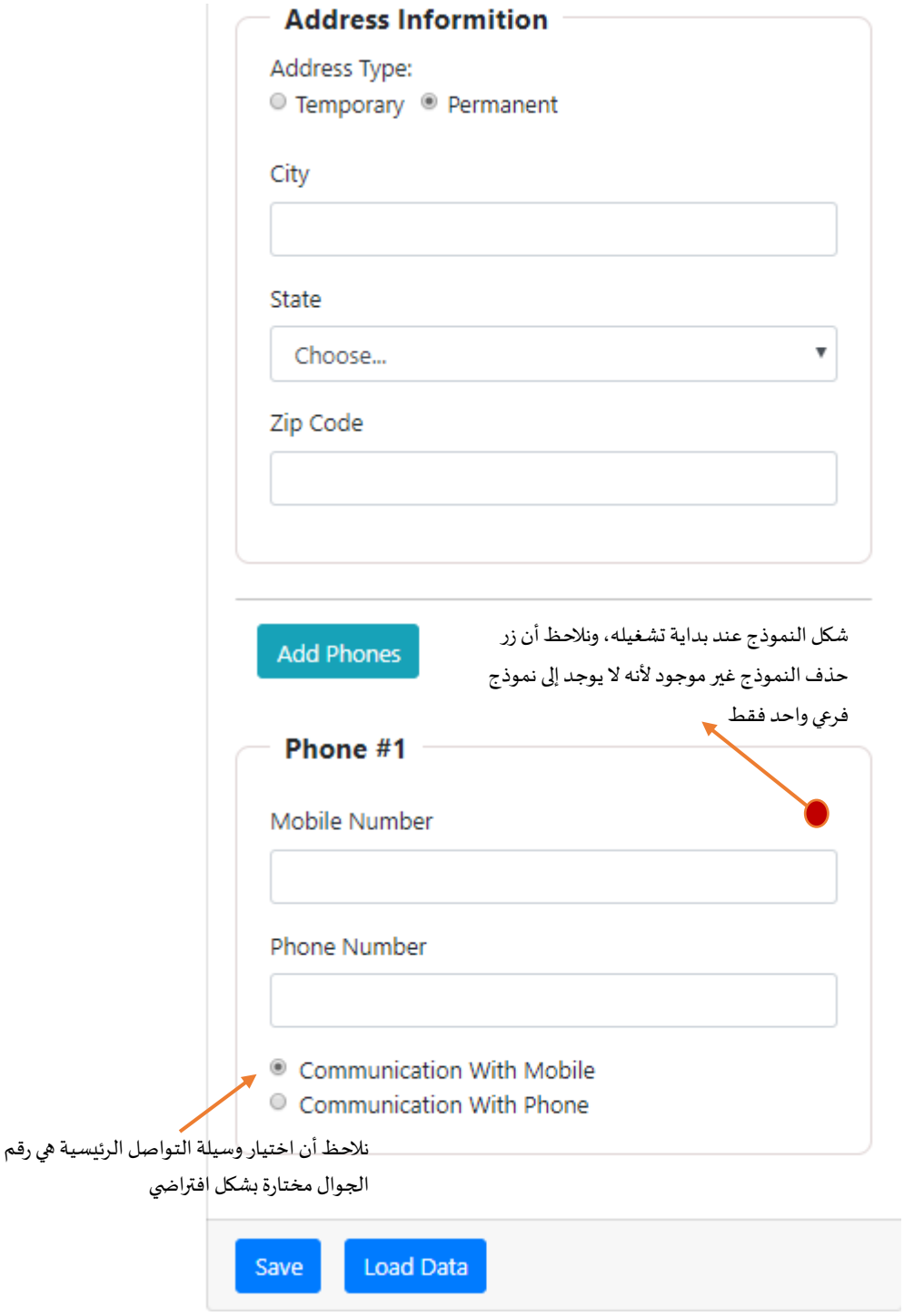
راجع السطر 26 (وضع شرط بحيث لا نريد زر الحذف يظهر إلا إذا كانت النماذج الفرعية داخل المصفوفة أكثر من واحد)

راجع الاسطر (33 – 36 – 43 – 46) (نربط label مع أداة الإدخال من خلال اسم متغير لكل أداة يتم تخليقها داخل النموذج الفرعي)

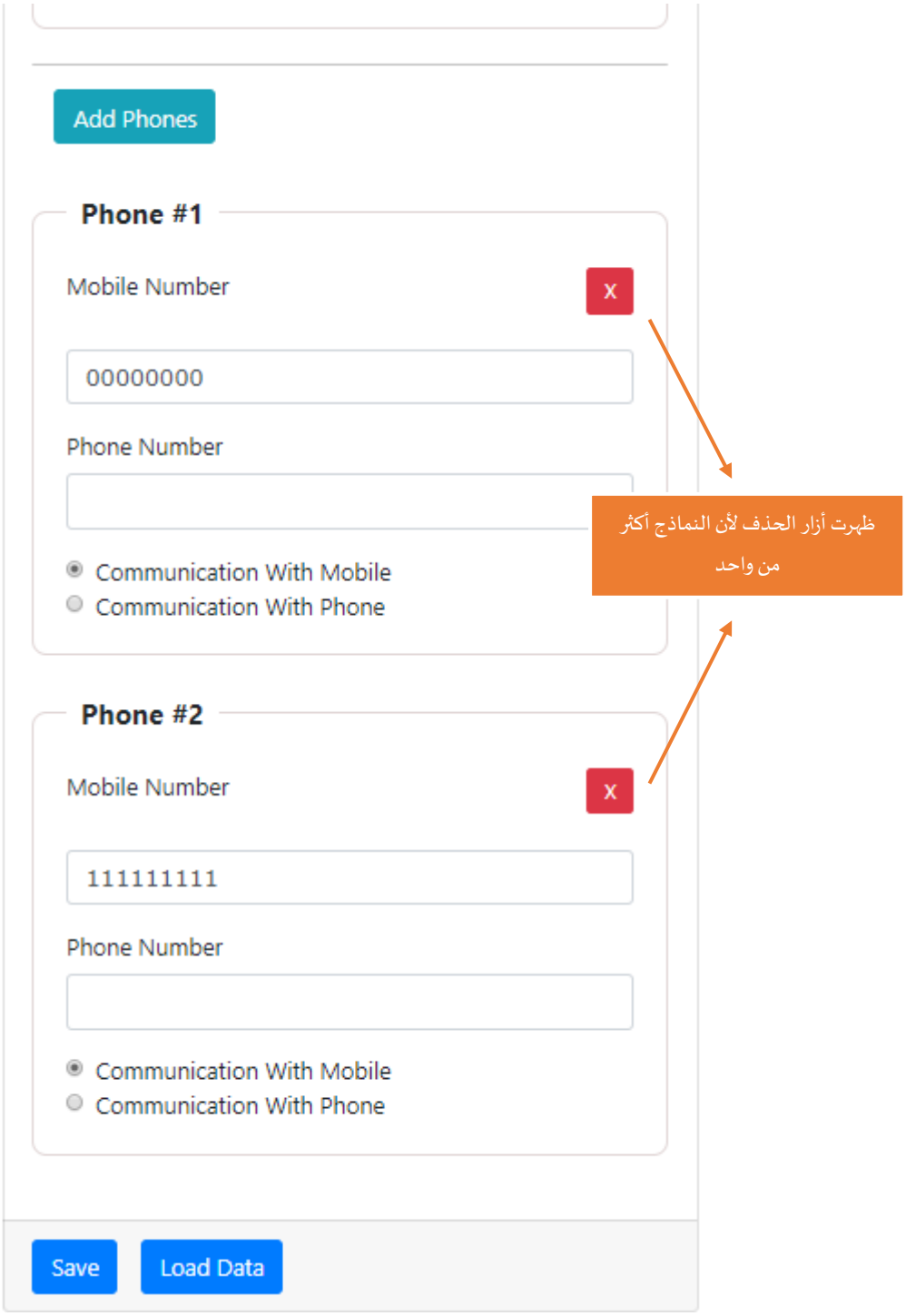
راجع الاسطر (37 – 47) (ربط الاسم البرمجي مع أداتي إدخال رقم الجوال ورقم الهاتف)

راجع (57 – 70) (ربط الاسم البرمجي مع أداتي radio)

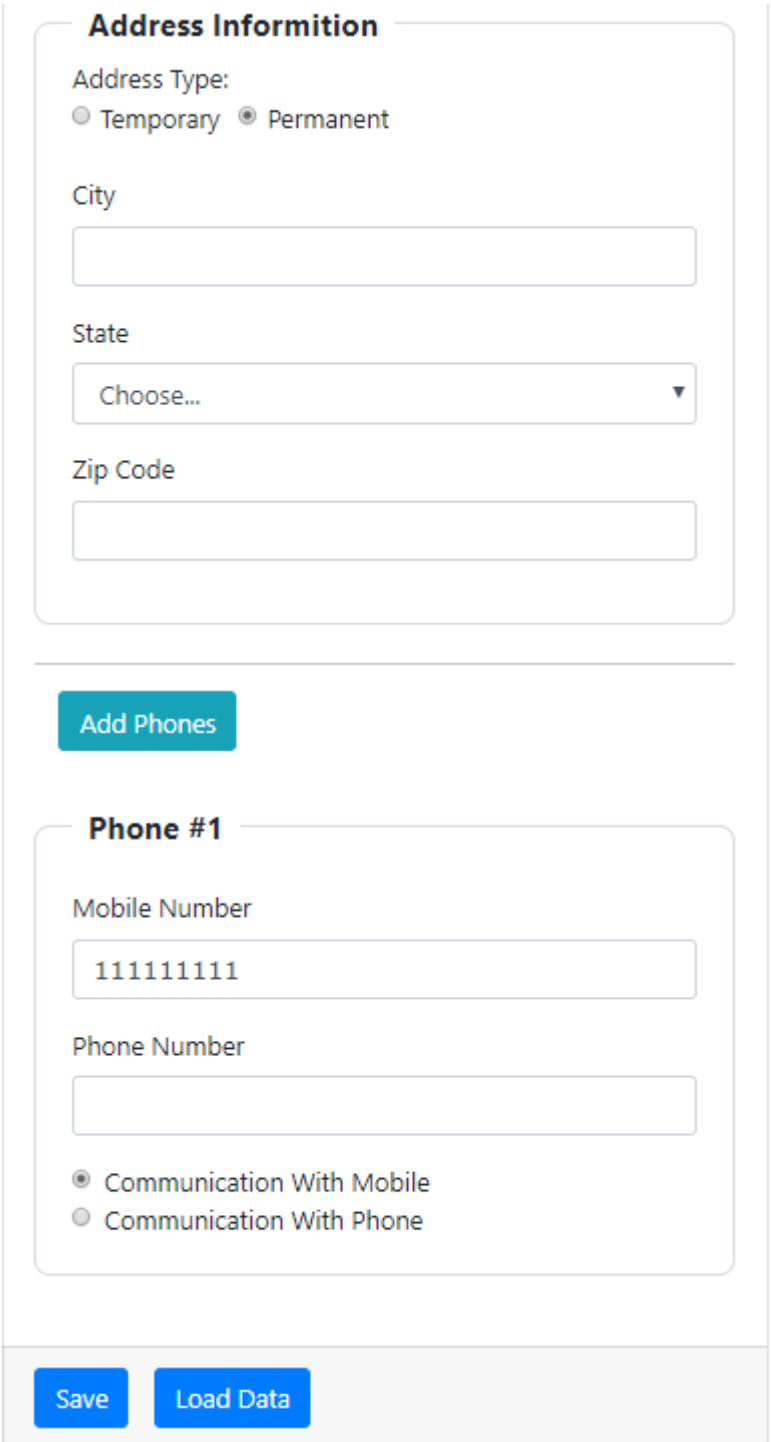
بعد شرح أهم الأجزاء في الكود الخاص بالنموذج الديناميكي في ملف app.component.html، الآن لنرى شكل النموذج على المتصفح:



الآن لنضغط على زر Add Phones ونقوم بتعبئة بعض البيانات ولنرى ماذا سيحدث:



نلاحظ أنه تم إنشاء وتخليق نموذج فرعي ثاني بدون أي مشاكل، الآن لنقم بحذف أحد هذين النموذجين الفرعيين ولنرى ماذا سيحدث:



نلاحظ أنه رجع النموذج إلى وضعه السابق، إلى الآن قطعنا شوط جيد في هذا المثال فقد قمنا ببناء نموذج فرعي ديناميكي يستطيع المستخدم إضافة نماذج بحسب احتياجه وبنفس الوقت يستطيع حذف ما يريد منها وجميع هذه الأمور تتم بشكل ديناميكي، بقي أخيراً التعامل مع التحقق من الصحة وكما قلنا في بداية هذا المثال نريد أن يكون مشروط وديناميكي بنفس الوقت، ونستطيع عمل هذا الأمر بكل بساطة من خلال إنشاء دالة وليكن أسمها phonesValidation() يتم تنفيذها في الحدث change لأداتي radio بحيث يتم تنفيذها كلما قام المستخدم باختيار أحد اختيارات أداتي radio، وهذه الأداة تستقبل باراميتر واحد وهو عبارة عن index للنموذج الفرعي داخل FormArray المتمثلة في المصفوفة التي اسميناها phones (نفس دالة الحذف) حيث تقوم بكل بساطة بتحديد الأدوات الثلاث (رقم الجوال ورقم الهاتف وأداة radio) داخل

كل نموذج فرعي منشأ ديناميكياً عن طريق index الخاص به ونخزنها في ثوابت، ومن ثم نقارن قيمة أداة radio هل هي تساوي القيمة mobileCommunication فإذا كانت تساويها فعناه أن المستخدم قد أختار خيار رقم الجوال، لذلك نضيف دوال التحقق من الصحة لأداة إدخال رقم الجوال ونحذفه من أداة إدخال رقم الهاتف الثابت، مع كتابة بعض الأكواد الإضافية، كالتالي: (ملف app.component.ts)

ملف app.component.ts

```
1. import { Component, OnInit, Renderer2 } from '@angular/core';
2. import { FormGroup, FormBuilder, Validators, FormArray } from '@angular/forms';
3. import { CustomValidator } from 'src/app/shared/custom.validators';
4.
5. @Component({
6.     selector: 'app-root',
7.     templateUrl: './app.component.html',
8.     styleUrls: ['./app.component.css'],
9. })
10.export class AppComponent implements OnInit {
11.     // tslint:disable: object-literal-key-quotes
12.     states: string[] = ['ALRiyadh', 'Makkah', 'AlSharqiyah', 'AlQasim'];
13.     userNames: string[] = ['faisal', 'DivFaisal'];
14.     form: FormGroup;
15.     userNameLength: any = '0';
16.     names: string[] = ['admin', 'administrator'];
17.     emails: string[] = ['faisal@gmail.com', 'google@gmail.com'];
18.
19.     messageValidation = {
20.         userName: {
21.             required: 'اسم المستخدم مطلوب',
22.             pattern: 'اسم المستخدم على الاقل ثلاث خانات',
23.             forbiddenNames: 'لايُسمح بتسجيل اسم المستخدم المُدخل',
24.             isEnglishLetters:
25.                 'لا يُسمح بوجود المسافات او الارقام او الرموز الخاصة او حروف غير الإنجليزية',
26.             isUserNameTaken: 'اسم المستخدم غير متاح',
27.         },
28.         email: {
29.             required: ' البريد الإلكتروني مطلوب',
30.             emailValidation: 'صيغة البريد الإلكتروني غير صحيحة',
31.             isEmailTaken: 'البريد الإلكتروني غير متاح',
32.         },
33.         passwordGroup: {
34.             passwordValidation: 'كلمة السر غير متطابقة',
35.         },
36.         password: {
37.             required: 'كلمة االسر مطلوبة',
38.             pattern: ' كلمة السر ست خانات ارقام وحروف وعلى الاقل حرف واحد كبير',
39.         },
40.         confirmPassword: {
41.             required: ' الحقل مطلوب',
42.         },
43.         gender: {
44.             required: 'الحقل مطلوب',
```

```
45.         },
46.         addressDate: {
47.             required: 'حقل تاريخ إنتهاء أقامة السكن مطلوب',
48.         },
49.         city: {
50.             required: 'حقل اسم المدينة مطلوب',
51.         },
52.         state: {
53.             required: 'حقل المنطقة مطلوب',
54.         },
55.         zipCode: {
56.             required: 'حقل الرمز البريدي مطلوب',
57.             pattern: 'الرمز البريدي لابد أن يكون قيمة رقمية من خمس خانات',
58.         },
59.     };
60.
61.     currentMessageValidation = {
62.         userName: '',
63.         email: '',
64.         passwordGroup: '',
65.         password: '',
66.         confirmPassword: '',
67.         gender: '',
68.         addressDate: '',
69.         city: '',
70.         state: '',
71.         zipCode: '',
72.     };
73.
74.     constructor(private fb: FormBuilder, private renderer: Renderer2) { }
75.
76.     ngOnInit() {
77.         this.form = this.fb.group({
78.             userName: [
79.                 null,
80.                 [
81.                     Validators.required,
82.                     Validators.pattern('.{3,}'),
83.                     CustomValidator.forbiddenNames(this.names),
84.                     CustomValidator.isEnglishLetters,
85.                 ],
86.                 [CustomValidator.isUserNameTaken(this.userNames)],
87.             ],
88.             email: [
89.                 null,
90.                 [Validators.required, CustomValidator.emailValidation],
91.                 [CustomValidator.isEmailTaken(this.emails)],
92.             ],
93.             passwordGroup: this.fb.group(
94.                 {
95.                     password: [
96.                         null,
97.                         [
```

```
                            Validators.required,
                            Validators.pattern(
                                    '(?=.*[A-Za-z])(?=.*[A-Z])(?!.* )(?=.*[0-9])[A-Za-z0-9d$@].{5,}'
                                ),
                            ],
                        ],
                        confirmPassword: [null, [Validators.required]],
                    },
                    { validator: CustomValidator.passwordValidation }
                ),
                gender: [null, Validators.required],
                address: this.fb.group({
                    addressType: ['permanent'],
                    addressDate: [null],
                    city: [null, Validators.required],
                    state: [null, Validators.required],
                    zipCode: [
                        null,
                        [Validators.required, Validators.pattern('^[0-9]{5}$')],
                    ],
                }),
                phones: this.fb.array([this.groupPhones()]),
            });

            this.userName.valueChanges.subscribe((value: string) => {
                this.userNameLength = value.length;
            });

            this.form.valueChanges.subscribe((data) => {
                this.loopThroughControls(this.form);
            });

            this.addressType.valueChanges.subscribe((data) => {
                this.addressDateValidation(data);
            });
        }

        groupPhones(): FormGroup {
            return this.fb.group({
                mobile: [
                    null,
                    [
                        Validators.required,
                        Validators.pattern(
                            '(?:NaN|-?(?:(?:\\d+|\\d*\\.\\d+)(?:[E|e][+|-]?\\d+)?|Infinity))'
                        ),
                    ],
                ],
                phone: [{ value: null, disabled: true }],
                mainCommunication: ['mobileCommunication'],
            });
```

```
149.      }
150.
151.      addPhones() {
152.          this.phones.push(this.groupPhones());
153.      }
154.
155.      removePhones(index: number) {
156.          this.phones.removeAt(index);
157.      }
158.
159.      phonesValidation(index: number) {
160.          const mainCommunication = this.phones.controls[index].get(
161.              'mainCommunication'
162.          );
163.          const mobile = this.phones.controls[index].get('mobile');
164.          const phone = this.phones.controls[index].get('phone');
165.          const Reg =
166.              '(?:NaN|-?(?:(?:\\d+|\\d*\\.\\d+)(?:[E|e][+|-]?\\d+)?|Infinity))';
167.          mobile.reset();
168.          phone.reset();
169.          if (mainCommunication.value === 'mobileCommunication') {
170.              mobile.setValidators([Validators.required, Validators.pattern(Reg)]);
171.              mobile.enable();
172.              this.renderer.selectRootElement('#MobileNumber' + index).focus();
173.              phone.clearValidators();
174.              phone.disable();
175.          } else {
176.              phone.setValidators([Validators.required, Validators.pattern(Reg)]);
177.              phone.enable();
178.              this.renderer.selectRootElement('#PhoneNumber' + index).focus();
179.              mobile.clearValidators();
180.          }
181.          mobile.updateValueAndValidity();
182.          phone.updateValueAndValidity();
183.      }
184.
185.      addressDateValidation(data: string) {
186.          if (data === 'temporary') {
187.              this.addressDate.setValidators(Validators.required);
188.          } else {
189.              this.addressDate.clearValidators();
190.              this.addressDate.markAsUntouched();
191.              this.addressDate.reset();
192.          }
193.          this.addressDate.updateValueAndValidity();
194.      }
195.
196.      save() {
197.          console.log(this.email.errors);
198.      }
199.
200.      loopThroughControls(formGroup: FormGroup = this.form) {
201.          Object.keys(formGroup.controls).forEach((key: string) => {
```

```
202.             const controlName = formGroup.get(key);
203.             this.currentMessageValidation[key] = '';
204.
205.             if (
206.                 controlName &&
207.                 controlName.invalid &&
208.                 (controlName.touched || controlName.dirty)
209.             ) {
210.                 const messages = this.messageValidation[key];
211.                 for (const controlError in controlName.errors) {
212.                     if (controlError) {
213.                         this.currentMessageValidation[key] += messages[controlError]
    + ' ';
214.                     }
215.                 }
216.             }
217.
218.             if (controlName instanceof FormGroup) {
219.                 this.loopThroughControls(controlName);
220.             }
221.         });
222.     }
223.
224.     laodData() {
225.         this.form.patchValue({
226.             userName: 'DivFaisal',
227.             email: 'test@test.com',
228.             passwordGroup: {
229.                 password: 'Aa1111',
230.                 confirmPassword: 'Aa1111',
231.             },
232.             gender: 'male',
233.             address: {
234.                 city: 'Riyadh',
235.                 state: 'ALRiyadh',
236.                 zipCode: '87678',
237.             },
238.         });
239.     }
240.
241.     get userName() {
242.         return this.form.get('userName');
243.     }
244.     get email() {
245.         return this.form.get('email');
246.     }
247.     get password() {
248.         return this.form.get('password');
249.     }
250.     get confirmPassword() {
251.         return this.form.get('confirmPassword');
252.     }
253.     get gender() {
```

```
254.            return this.form.get('gender');
255.        }
256.        get address() {
257.            return this.form.get('address');
258.        }
259.        get city() {
260.            return this.form.get('address').get('city');
261.        }
262.        get state() {
263.            return this.form.get('address').get('state');
264.        }
265.        get zipCode() {
266.            return this.form.get('address').get('zipCode');
267.        }
268.        get addressType() {
269.            return this.form.get('address').get('addressType');
270.        }
271.        get addressDate() {
272.            return this.form.get('address').get('addressDate');
273.        }
274.        get phones() {
275.            return this.form.get('phones') as FormArray;
276.        }
277.    }
278.
```

راجع السطر ١٤٦ (نجعل أداة إدخال رقم الهاتف الثابت غير مفعلة بشكل افتراضي لأن أداة radio القيمة الافتراضية لها هي mobileCommunication والذي يعني أن المختار هو وسيلة الاتصال الرئيسية هي رقم الجوال لذلك نعطل أداه إدخال رقم الهاتف الثابت عند بداية إنشاء او تخليق أي نموذج فرعي بشكل ديناميكي.)

راجع الاسطر (من ١٥٩ إلى ١٨٣) (دالة التحقق من الصحة، دالة التحقق من الصحة وتستقبل باراميتر واحد وهو index او موقع النموذج الفرعي المنشأ ديناميكياً داخل المصفوفة FormArray التي اسميناها phones)

راجع الاسطر (من ١٦٠ إلى ١٦٨) (الوصول إلى الأدوات داخل النموذج الفرعي المنشأ ديناميكي عن طريق index الخاص به، ومن ثم تخزينها في الثوابت بنفس أسماء الأدوات (لك حرية اختيار الأسماء التي تريدها لكن انا هنا فضلت ان اسمي الثوابت بنفس أسماء الأدوات)، ومن ثم نقوم بإرجاع الأدوات إداة إدخال رقم الجوال وأداة إدخال رقم الهاتف التي خزناها في الثوابت ذات الاسم mobile وphone إلى الوضع الافتراضي عن طريق الدالة reset)

راجع الاسطر ١٦٩) إلى ١٧٤) (نقوم بوضع شرط لتأكد هل قيمة أداة radio هي mobileCommunication أي المختار هو وسيلة الاتصال رقم الجوال)، فإذا كانت كذلك نقوم بإضافة دوال التحقق من الصحة إلى أداة mobile مع تفعيلها enable وحذف دوال التحقق من الصحة من الأداة phone مع تعطيلها.

راجع الاسطر (من ١٧٥ إلى ١٨٠) (في حال تم اختيار أداة radio الثانية الخاصة برقم الهاتف الثابت، نُعيد تكرار نفس الأكواد في الاسطر من ١٦٩ إلى ١٧٤ مع مراعاة تبديل الأدوات)

بعد تهيئتنا لدالة التحقق من الصحة في ملف app.component.ts، بقي أن نضع الشروط لإظهار وإخفاء رسائل الخطأ لكل أداة، والفرق هنا أننا لن نضعها في الكائن messageValidation كما كنا نفعل سابقاً وانما سوف اكتب رسائل الخطأ بشكل مباشر في ملف app.component.html، مع العلم انها تكرار لما أخذناه لذلك سوف استعرض محتويات هذا الملف كاملاً مع وضع علامة عند الأكواد المضافة، كالتالي:

app.component.html ملف

```
1. <div class="container-fluid">
2.   <div class="card mx-auto col-sm-12 col-md-6 col-lg-6 col-xl-6 pt-sm-4">
3.     <form [formGroup]="form">
4.       <div class="card-header">
5.         <h4 class="text-center">Reactive Forms</h4>
6.       </div>
7.
8.       <div class="card-body">
9.         <!-- أداة إدخال اسم المستخدم -->
10.        <div class="form-group">
11.          <label>User Name</label>
12.          <div class="form-inline">
13.            <input
14.              formControlName="userName"
15.              [ngClass]="{
16.                'col-10': true,
17.                'form-control': true,
18.                'is-
   invalid': currentMessageValidation.userName || userName.hasError('isUserNameTaken'),
19.                'is-valid':
20.                  !userName.hasError('isUserNameTaken') &&
21.                  !userName.pending &&
22.                  userName.value !== null &&
23.                  userNameLength >= 3
24.              }"
25.              (blur)="loopThroughControls()"
26.              (input)="loopThroughControls()"
27.            />
28.            <label class="col-2">{{ userNameLength }}</label>
29.          </div>
30.          <!-- جزء التحقق من الصحة الخاص باسم المستخدم -->
31.          <small class="text-danger" *ngIf="currentMessageValidation.userName">
32.            {{ currentMessageValidation.userName }}
33.          </small>
34.          <!-- <small
35.            class="text-danger"
36.            *ngIf="userName.hasError('isUserNameTaken') && userName.dirty && !currentMe
   ssageValidation.userName"
37.          >
38.            اسم المستخدم غير متاح
39.          </small> -->
40.          <div class="alert alert-info col-10" *ngIf="userName.pending">
41.            جاري التحقق من اتاحة اسم المستخدم
42.          </div>
```

```
43.         </div>
44.
45.         <!-- أداة إدخال البريد الإلكتروني -->
46.         <div class="form-group">
47.           <label>Email</label>
48.           <input
49.             type="email"
50.             formControlName="email"
51.             [ngClass]="{
52.               'form-control': true,
53.               'is-
   invalid': currentMessageValidation.email || email.hasError('isEmailTaken'),
54.               'is-valid':
55.                 !email.hasError('isEmailTaken') &&
56.                 !email.pending &&
57.                 email.value !== null &&
58.                 email.value !== '' &&
59.                 !email.hasError('emailValidation')
60.             }"
61.             (blur)="loopThroughControls()"
62.           />
63.
64.           <!-- جزء التحقق من الصحة الخاص بالبريد الإلكتروني -->
65.           <small class="text-danger" *ngIf="currentMessageValidation.email">
66.             {{ currentMessageValidation.email }}
67.           </small>
68.           <div class="alert alert-info" *ngIf="email.pending">
69.             جاري التحقق من إتاحة البريد الإلكتروني
70.           </div>
71.         </div>
72.
73.         <!-- بداية النموذج الفرعي لكلمة السر وإعادة إدخال كلمة السر -->
74.         <div formGroupName="passwordGroup">
75.           <!-- أداة إدخال كلمة السر -->
76.           <div class="form-group">
77.             <label>Password</label>
78.             <input
79.               type="password"
80.               autocomplete="of"
81.               formControlName="password"
82.               [ngClass]="{
83.                 'form-control': true,
84.                 'is-invalid': currentMessageValidation.password
85.               }"
86.               (blur)="loopThroughControls()"
87.             />
88.             <!-- جزء التحقق من الصحة الخاص بكلمة السر -->
89.             <small
90.               class="text-danger"
91.               *ngIf="currentMessageValidation.password"
92.             >
93.               {{ currentMessageValidation.password }}
94.             </small>
```

```
95.          </div>
96.
97.          <!-- أداة إعادة إدخال كلمة السر -->
98.          <div class="form-group">
99.            <label>Confirm Password</label>
100.             <input
101.               type="password"
102.               autocomplete="of"
103.               formControlName="confirmPassword"
104.               [ngClass]="{
105.                 'form-control': true,
106.                 'is-
   invalid': currentMessageValidation.confirmPassword || currentMessageValidation.password
   Group
107.               }"
108.               (blur)="loopThroughControls()"
109.             />
110.
111.             <!-- جزء التحقق من الصحة الخاص بأداة إعادة إدخال كلمة السر -->
112.             <small
113.               class="text-danger"
114.               *ngIf="currentMessageValidation.confirmPassword || currentMessageValid
   ation.passwordGroup"
115.             >
116.               {{ currentMessageValidation.confirmPassword ?
117.               currentMessageValidation.confirmPassword :
118.               currentMessageValidation.passwordGroup }}
119.             </small>
120.           </div>
121.         </div>
122.         <!-- أداة تحديد نوع الجنس ذكر أو أنثى -->
123.         <label class="pr-2">Gender</label>
124.         <div class="form-check form-check-inline">
125.           <input
126.             type="radio"
127.             formControlName="gender"
128.             id="maleGender"
129.             value="male"
130.             [ngClass]="{
131.               'form-check-input': true,
132.               'is-invalid': currentMessageValidation.gender
133.             }"
134.             (blur)="loopThroughControls()"
135.           />
136.           <label class="form-check-label" for="gender">Male</label>
137.         </div>
138.         <div class="form-check form-check-inline">
139.           <input
140.             type="radio"
141.             formControlName="gender"
142.             id="femaleGender"
143.             value="femail"
144.             [ngClass]="{
```

```
145.                 'form-check-input': true,
146.                 'is-invalid': currentMessageValidation.gender
147.               }"
148.               (blur)="loopThroughControls()"
149.             />
150.             <label class="form-check-label" for="femaleGender">Femail</label>
151.           </div>
152.           <!-- الجزء الخاص بتحقق من الصحة لأداة تحديد نوع الجنس -->
153.           <small class="text-danger" *ngIf="currentMessageValidation.gender">
154.             {{ currentMessageValidation.gender }}
155.           </small>
156.
157.           <!-- بداية النموذج الفرعي -->
158.           <fieldset class="scheduler-border" formGroupName="address">
159.             <legend class="scheduler-border">Address Informition</legend>
160.
161.             <label class="pr-2" style="margin-bottom: 0px">Address Type:</label>
162.             <br />
163.             <div class="form-check form-check-inline">
164.               <input
165.                 class="form-check-input"
166.                 type="radio"
167.                 formControlName="addressType"
168.                 id="temporary"
169.                 value="temporary"
170.               />
171.               <label class="form-check-label" for="temporary">Temporary</label>
172.             </div>
173.             <div class="form-check form-check-inline">
174.               <input
175.                 class="form-check-input"
176.                 type="radio"
177.                 formControlName="addressType"
178.                 id="permanent"
179.                 value="permanent"
180.               />
181.               <label class="form-check-label" for="permanent">Permanent</label>
182.             </div>
183.             <input
184.               type="date"
185.               formControlName="addressDate"
186.               [class.has-error]="currentMessageValidation.addressDate"
187.               *ngIf="addressType.value === 'temporary'"
188.               (blur)="addressDateValidation('temporary')"
189.             />
190.             <div>
191.               <small
192.                 class="text-danger"
193.                 *ngIf="currentMessageValidation.addressDate"
194.               >
195.                 {{ currentMessageValidation.addressDate }}
196.               </small>
197.             </div>
```

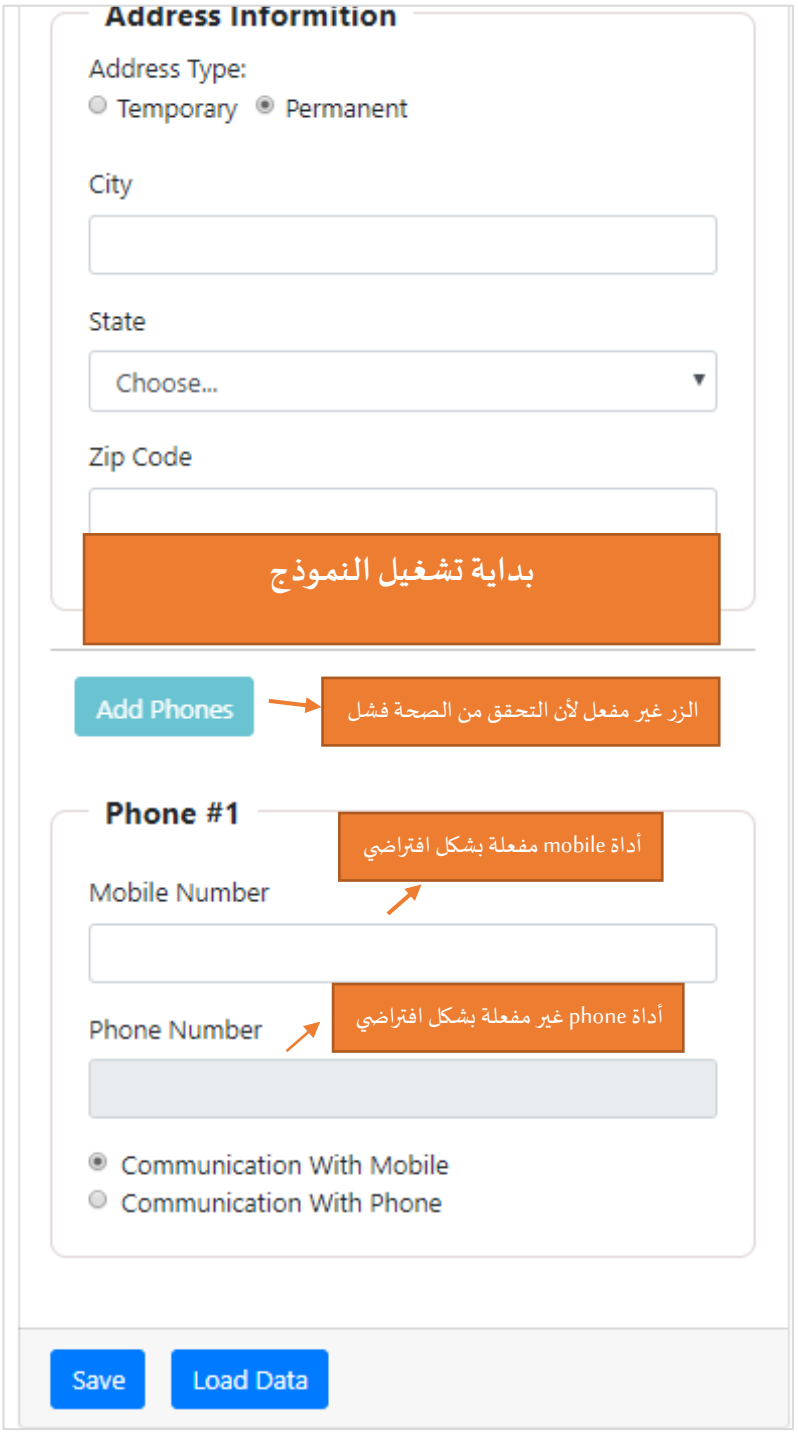
```
198.
199.             <!-- أداة إدخال اسم المدينة -->
200.             <div class="form-group pt-4">
201.               <label>City</label>
202.               <input
203.                 formControlName="city"
204.                 [ngClass]="{
205.                   'form-control': true,
206.                   'is-invalid': currentMessageValidation.city
207.                 }"
208.                 (blur)="loopThroughControls()"
209.               />
210.               <!-- جزء التحقق من الصحة لأداة إدخال اسم المدينة -->
211.               <small class="text-danger" *ngIf="currentMessageValidation.city">
212.                 {{ currentMessageValidation.city }}
213.               </small>
214.             </div>
215.             <!-- أداة اختيار اسم المنطقة او الولاية -->
216.             <div class="form-group">
217.               <label>State</label>
218.               <select
219.                 formControlName="state"
220.                 [ngClass]="{
221.                   'form-control': true,
222.                   'is-invalid': currentMessageValidation.state
223.                 }"
224.                 (blur)="loopThroughControls()"
225.               >
226.                 <option selected [ngValue]="null">Choose...</option>
227.                 <option *ngFor="let item of states" [value]="item">
228.                   {{ item }}
229.                 </option>
230.               </select>
231.               <!-- جزء التحقق من الصحة لأداة أختيار اسم المنطقة -->
232.               <small class="text-danger" *ngIf="currentMessageValidation.state">
233.                 {{ currentMessageValidation.state }}
234.               </small>
235.             </div>
236.             <!-- أداة إدخال الرمز البريدي -->
237.             <div class="form-group">
238.               <label>Zip Code</label>
239.               <input
240.                 formControlName="zipCode"
241.                 [ngClass]="{
242.                   'form-control': true,
243.                   'is-invalid': currentMessageValidation.zipCode
244.                 }"
245.                 (blur)="loopThroughControls()"
246.               />
247.               <!-- جزء التحقق من الصحة لأداة إدخال الرمز البريدي -->
248.               <small class="text-danger" *ngIf="currentMessageValidation.zipCode">
249.                 {{ currentMessageValidation.zipCode }}
250.               </small>
```

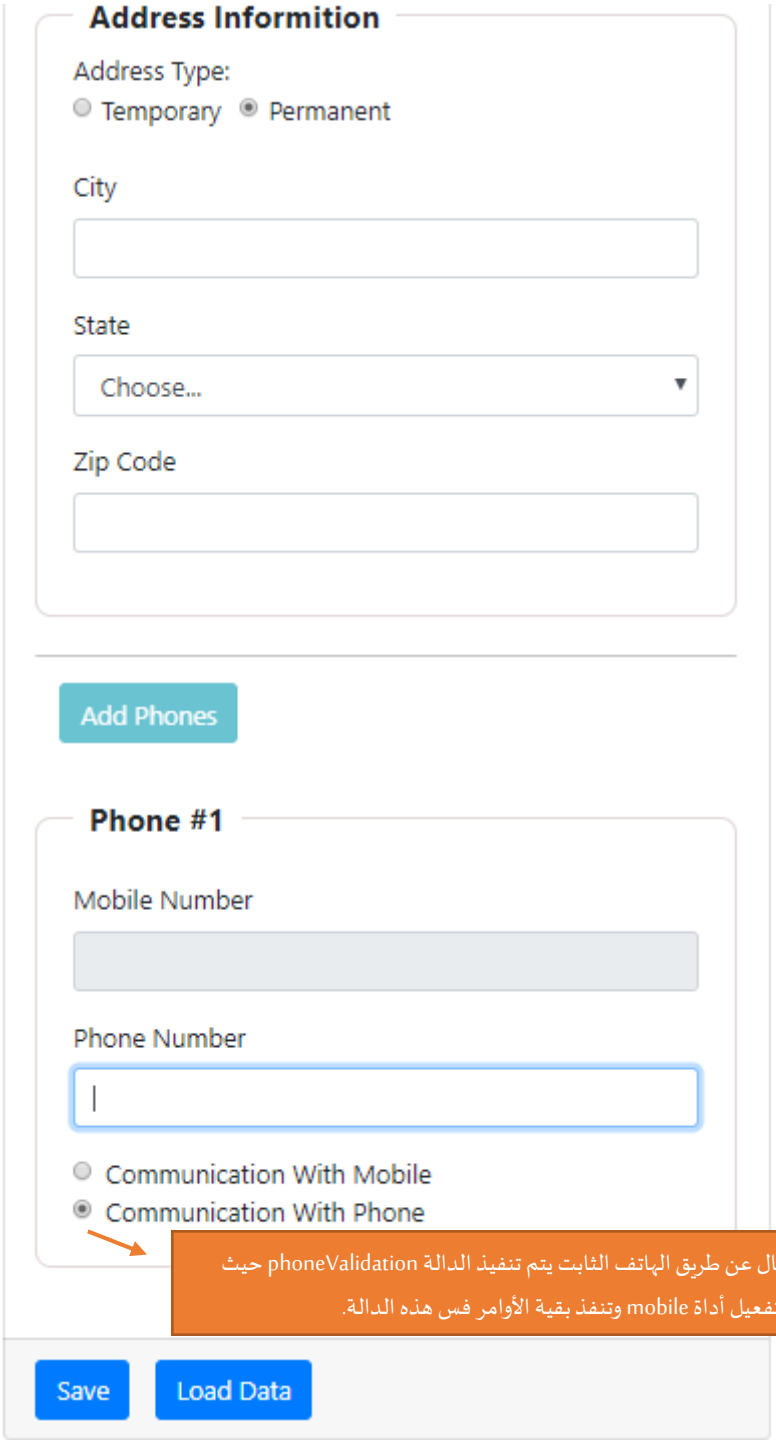
```
251.             </div>
252.           </fieldset>
253.
254.           <!-- بداية النموذج الفرعي الديناميكي -->
255.           <hr style="border: 1px solid silver" />
256.           <div class="form-group">
257.             <!-- زر إضافة نموذج فرعي -->
258.             <div class="col-md-offset-2 col-md-4">
259.               <button
260.                 type="button"
261.                 class="btn btn-info mb-3"
262.                 (click)="addPhones()"
263.                 [disabled]="phones.invalid"
264.               >
265.                 Add Phones
266.               </button>
267.             </div>
268.           </div>
269.           <!--phones المتمثلة في المصفوفة FormArray بداية النموذج الفرعي -->
270.           <fieldset
271.             class="scheduler-border"
272.             formArrayName="phones"
273.             *ngFor="let phoneArray of phones.controls; let i = index"
274.           >
275.             <legend class="scheduler-border">Phone #{{ i + 1 }}</legend>
276.             <!--FormGroup بداية النموذج الفرعي التي يتم إنشائه ديناميكياً -->
277.             <div [formGroupName]="i">
278.               <!-- بداية زر حذف لكل نموذج فرعي يتم إنشائه ديناميكياً -->
279.               <div class="form-group">
280.                 <button
281.                   type="button"
282.                   class="btn btn-danger btn-xs float-right mb-4 w-100"
283.                   (click)="removePhones(i)"
284.                   *ngIf="phones.length > 1"
285.                 >
286.                   X
287.                 </button>
288.               </div>
289.               <!-- بداية أداة أدخال رقم الجوال -->
290.               <div class="form-group">
291.                 <label [attr.for]="'MobileNumber' + i">Mobile Number</label>
292.                 <input
293.                   type="tel"
294.                   [id]="'MobileNumber' + i"
295.                   formControlName="mobile"
296.                   [ngClass]="{
297.                     'form-control': true,
298.                     'is-invalid':
299.                       phoneArray.get('mobile').invalid &&
300.                       phoneArray.get('mobile').touched &&
301.                       phoneArray.get('mobile').dirty
302.                   }"
303.                 />
```

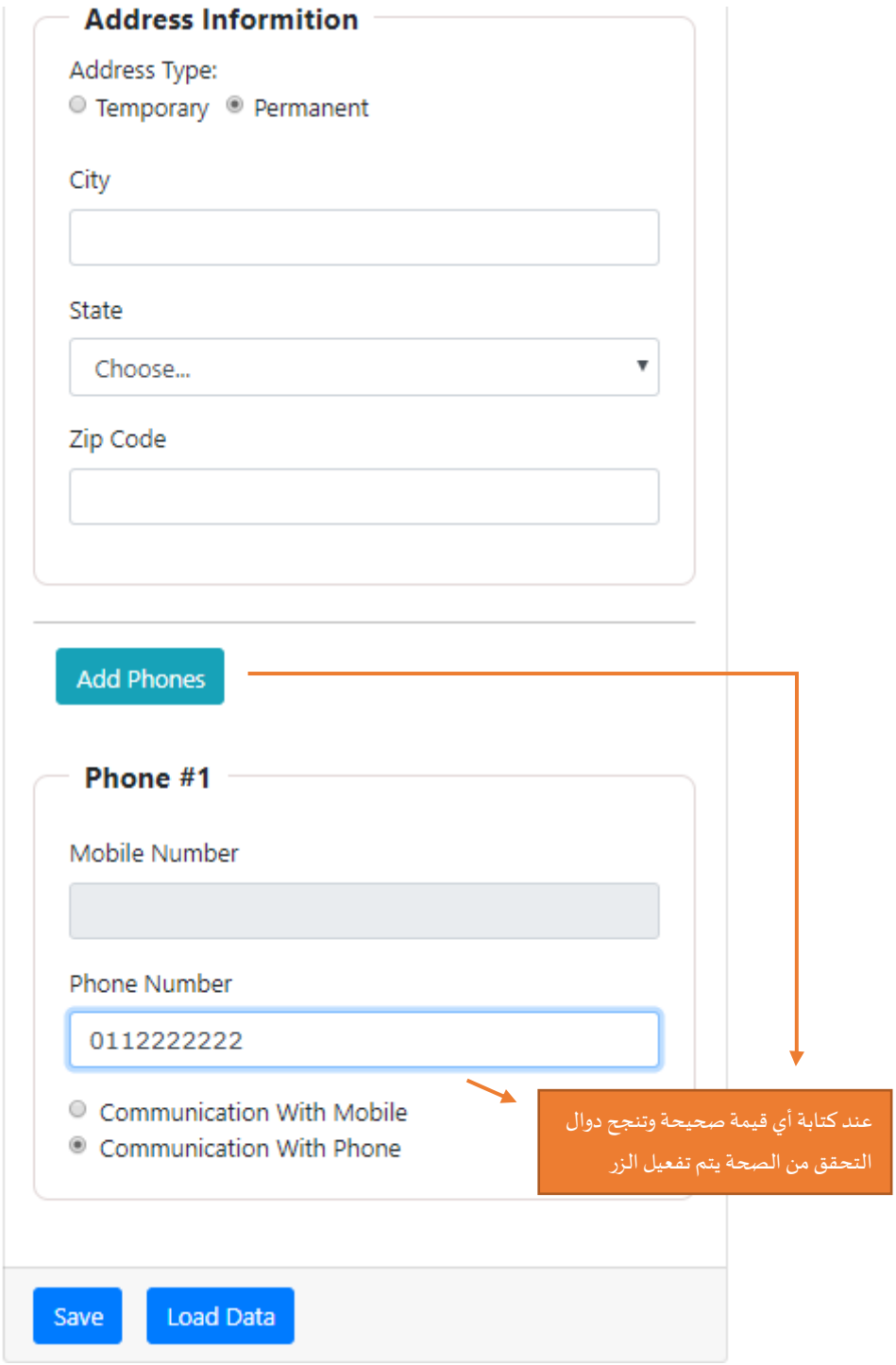
```
304.             <small
305.               class="text-danger"
306.               *ngIf="
307.                 phoneArray.get('mobile').hasError('required') &&
308.                 phoneArray.get('mobile').touched &&
309.                 phoneArray.get('mobile').dirty
310.               "
311.             >
312.               حقل رقم الهاتف الجوال مطلوب
313.             </small>
314.             <small
315.               class="text-danger"
316.               *ngIf="
317.                 phoneArray.get('mobile').hasError('pattern') &&
318.                 phoneArray.get('mobile').touched &&
319.                 phoneArray.get('mobile').dirty
320.               "
321.             >
322.               صيغة رقم الهانف الجوال غير صحيحة
323.             </small>
324.           </div>
325.           <!-- بداية أداة إدخال رقم الهاتف الثابت -->
326.           <div class="form-group">
327.             <label [attr.for]="'PhoneNumber' + i">Phone Number</label>
328.             <input
329.               type="tel"
330.               [id]="'PhoneNumber' + i"
331.               formControlName="phone"
332.               [ngClass]="{
333.                 'form-control': true,
334.                 'is-invalid':
335.                   phoneArray.get('phone').invalid &&
336.                   phoneArray.get('phone').touched &&
337.                   phoneArray.get('phone').dirty
338.               }"
339.             />
340.
341.             <small class="text-danger"
342.               *ngIf="
343.                 phoneArray.get('phone').hasError('required') &&
344.                 phoneArray.get('phone').touched &&
345.                 phoneArray.get('phone').dirty
346.               ">
347.               حقل الهاتف الثابت مطلوب
348.             </small>
349.             <small class="text-danger"
350.               *ngIf="
351.                 phoneArray.get('phone').hasError('pattern') &&
352.                 phoneArray.get('phone').touched &&
353.                 phoneArray.get('phone').dirty
354.               ">
355.               صيغة رقم الهانف الثابت غير صحيحة
356.             </small>
```

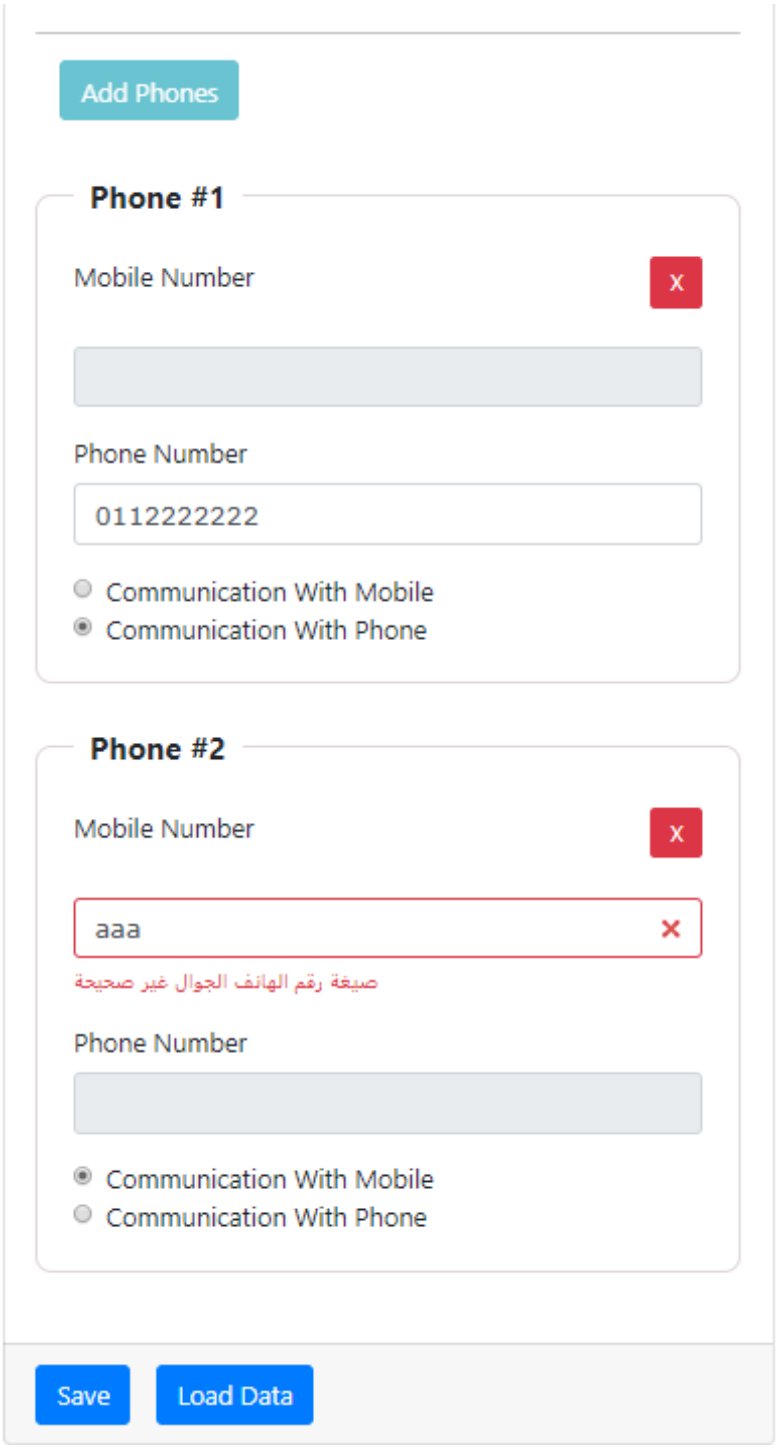
```
357.             </div>
358.             <!--radio بداية أداتي-->
359.             <!-- أداة الـradio الأولى التي إذا كانت مختارة يتم التحقق من الصحة لأداة إدخال رقم الجوال -->
360.             <div class="form-check">
361.               <input
362.                 class="form-check-input"
363.                 type="radio"
364.                 formControlName="mainCommunication"
365.                 [id]="'mobileCommunication' + i"
366.                 value="mobileCommunication"
367.                 (change)="phonesValidation(i)"
368.               />
369.               <label
370.                 class="form-check-label"
371.                 [attr.for]="'mobileCommunication' + i"
372.               >
373.                 Communication With Mobile
374.               </label>
375.             </div>
376.             <!-- أداة الـradio الثانية التي إذا كانت مختارة يتم التحقق من الصحة لأداة إدخال رقم الهاتف الثابت -->
377.             <div class="form-check">
378.               <input
379.                 class="form-check-input"
380.                 type="radio"
381.                 formControlName="mainCommunication"
382.                 [id]="'phoneCommunication' + i"
383.                 value="phoneCommunication"
384.                 (change)="phonesValidation(i)"
385.               />
386.               <label
387.                 class="form-check-label"
388.                 [attr.for]="'phoneCommunication' + i"
389.               >
390.                 Communication With Phone
391.               </label>
392.             </div>
393.           </div>
394.         </fieldset>
395.       </div>
396.
397.       <!-- بداية أدوات الأزرار -->
398.       <div class="card-footer">
399.         <button class="btn btn-primary" (click)="save()">Save</button>
400.         <button class="btn btn-primary ml-3" (click)="laodData()">
401.           Load Data
402.         </button>
403.       </div>
404.     </form>
405.   </div>
406. </div>
407.
```

الآن لنستعرض النموذج بعد التعديلات التي أجريناها عليه:




Address Information
Address Type:
Temporary
Permanent
City
State
Choose...
Zip Code
Add Phones
Phone #1
Mobile Number
Phone Number
Communication With Mobile
Communication With Phone
عند أختيار وسيلة الاتصال عن طريق الهاتف الثابت يتم تنفيذ الدالة phoneValidation حيث
تفعل أداة phone ويلغى تفعيل أداة mobile وتنفذ بقية الأوامر فس هذه الدالة.
Save
Load Data


Address Information
Address Type:
Temporary
Permanent
City
State
Choose...
Zip Code
Add Phones
Phone #1
Mobile Number
Phone Number
0112222222
Communication With Mobile
Communication With Phone
عند كتابة أي قيمة صحيحة وتنجح دوال التحقق من الصحة يتم تفعيل الزر
Save
Load Data



نلاحظ عند إنشاء نموذج فرعي جديد يصبح كل نموذج مستقل بذاته من ناحية التحقق من الصحة أو التبديل بين أي رقم هو الأساسي سواء رقم الهاتف او رقم الجوال، وكل هذه الأمور تتم بشكل ديناميكي.

وبذلك نكون أنهينا هذا المثال وهو مثال مهم ويحتوي على تفاصيل عديدة وممكن أخذ أفكار منها والأستفادة في أفكار أخرى بحسب احتياج كل مشروع.

3.5. المثال الثاني:

في هذا المثال نريد أن نعرض مجموعة من الأسئلة على النموذج بشكل ديناميكي ويتم عرضها من خلال أداتين الأداة الأولى textarea وأداة radio لأختيار الإجابة على السؤال true او false، مع العلم أن الأسئلة موجودة في مصفوفة كائنات كل كائن مخزن فيه السؤال.

الحل:

هذا المثال مشابهه للمثال السابق بنسبة كبيرة لذلك الخطوات المكررة لن أعيد شرحها، والفرق هنا أن تخليق الأدوات لن يتم إذا صغط المستخدم على الزر وإنما في بداية تشغيل النموذج، حيث يكون logic بالطريقة التالية:

أولاً: نقوم بإنشاء FormArray الرئيسي وبنفس الوقت لا ننشأ بداخله أي نموذج فرعي أو أدوات، وليكن اسم هذه المصفوفة questions، كالتالي:

app.component.ts جزء من ملف

```
1. ngOnInit() {
2.         this.form = this.fb.group({
3.             userName: [null,
4.                 [
5.                     Validators.required,
6.                     Validators.pattern('.{3,}'),
7.                     CustomValidator.forbiddenNames(this.names),
8.                     CustomValidator.isEnglishLetters
9.                 ], [CustomValidator.isUserNameTaken(this.userNames)
10.                ]
11.            ],
12.            email: [null,
13.                [
14.                    Validators.required,
15.                    CustomValidator.emailValidation
16.                ], [CustomValidator.isEmailTaken(this.emails)]
17.            ],
18.            passwordGroup: this.fb.group({
19.                password: [null,
20.                    [
21.                        Validators.required,
22.                        Validators.pattern('(?=.*[A-Za-z])(?=.*[A-Z])(?!.* )(?=.*[0-
   9])[A-Za-z0-9d$@].{5,}')
23.                    ]
24.                ],
25.                confirmPassword: [null,
26.                    [
27.                        Validators.required
28.                    ]
29.                ]
30.            }, { validator: CustomValidator.passwordValidation }),
31.            gender: [null, Validators.required],
32.            address: this.fb.group({
33.                addressType: ['permanent'],
```

```
                addressDate: [null],
                city: [null, Validators.required],
                state: [null, Validators.required],
                zipCode: [null, [Validators.required, Validators.pattern('^[0-9]{5}$')]]
            }),
            phones: this.fb.array([this.groupPhones()]),
            questions: this.fb.array([])
        });
    }
```

ثانياً: ننشأ مصفوفة الكائنات باسم Questions حيث كل كائن يحتوي على key باسم question وقيمته هي السؤال الذي نريد عرضه على النموذج، ملاحظة في البرامج الواقعية هذه الأسئلة تأتي في الغالب من قاعدة بيانات وتحتوي على اكثر من key كأن يكون لدينا على سبيل المثال key مخصص للإجابة التي اختارها المستخدم وآخر للأجابة الصحيحة ...الخ، ولكن هنا الفكرة ليس بناء نموذج اختبار الكتروني وإنما كيفية إنشاء او تخليق الأدوات على النموذج بشكل الكتروني، الآن لنقوم بإنشاء مصفوفة الكائنات:

app.component.ts جزء من ملف

```
Questions = [
  {
    question: 'In Angular, you can pass data from parent component to child component using @Input()'
  },
  {
    question: 'In Angular, you can pass data from child component to parent component using Output'
  },
  {
    question: 'You can create local HTML reference of HTML tag using variable which starts with character &'
  },
  {
    question: 'A directive which modifies DOM hierarchy is called Structural directive'
  }
];
```

ثالثاً: نقوم بإنشاء دالة وليكن اسمها setQuestions تُعيد النوع FormArray ومهمتها إنشاء نموذج فرعي FormGroup وأداتين الأولى لعرض السؤال والتي سوف نربطها بأداة teaxtarea والثانية خاصة بالأجابة والتي سوف نربطها بأداتي radio حيث يتم إنشاء كل ماسبق بشكل ديناميكي بناءً على مصفوفة الكائنات Questions ويتم عمل ذلك من خلال عمل loop داخل مصفوفة الكائنات السابقة ومع كل loop يتم تخليق الكائنات عن طريق FormBuilder وإضافتها إلى مصفوفة من النوع FormArray حيث يتم تعريفها داخل الدالة setQuestions ومن ثم يتم إعادة هذه المصفوفة، ويتم عمل ذلك كالتالي:

app.component.ts جزء من ملف

```
setQuestions(): FormArray {
    const formArray = new FormArray([]);
    this.Questions.forEach(group => {
    formArray.push(this.fb.group({
```

```
            question: [group.question],
            answer: [null, Validators.required]
        }));
    });
    return formArray;
}
```

خامساً: الدالة السابقة قيمتها هي النموذج الفرعي وأدواتها التي تم إنشاءها ديناميكياً بناءً على مصفوفة الكائنات لذلك سوف نستفيد من الدالة setControls التي يقدمها لنا angular forms لإضافة النموذج المُخلق ديناميكياً إلى المصفوفة FormArray الأساسية التي تم إنشائها في الخطوة (أولاً)، حيث أن الدالة setContrlos تستقبل باراميترين الأول أسم FormArray التي نريد تخليق النموذج الفرعي أو الأدوات بداخله والباراميتر الثاني هو logic الأدوات التي نريد إضافتها، ويتم ذلك بكتابة هذا السطر في الدالة ngOnInit، كالتالي:

```
this.form.setControl('questions', this.setQuestions());
```

الآن لنضيف الأكواد السابقة إلى ملف app.component.ts، كالتالي:

ملف app.component.ts

```
import { Component, OnInit, Renderer2 } from '@angular/core';

import {
        FormGroup,
        FormBuilder,
        Validators,
        FormArray,
        FormControl
} from '@angular/forms';

import { CustomValidator } from 'src/app/shared/custom.validators';

@Component({
    selector: 'app-root',
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css']
})

export class AppComponent implements OnInit {

    states: string[] = ['ALRiyadh', 'Makkah', 'AlSharqiyah', 'AlQasim'];

    userNames: string[] = ['faisal', 'DivFaisal'];

    form: FormGroup;

    userNameLength: any = '0';

    names: string[] = ['admin', 'administrator'];

    emails: string[] = ['faisal@gmail.com', 'google@gmail.com'];

```

```
33.     Questions = [
34.         {
35.             question: 'In Angular, you can pass data from parent component to child com
    ponent using @Input()'
36.         },
37.         {
38.             question: 'In Angular, you can pass data from child component to parent com
    ponent using Output'
39.         },
40.         {
41.             question: 'You can create local HTML reference of HTML tag using variable w
    hich starts with character &'
42.         },
43.         {
44.             question: 'A directive which modifies DOM hierarchy is called Structural di
    rective'
45.         }
46.     ];
47.
48.     messageValidation = {
49.         'userName': {
50.             'required': 'اسم المستخدم مطلوب',
51.             'pattern': 'اسم المستخدم على الاقل ثلاث خانات',
52.             'forbiddenNames': 'لايُسمح بتسجيل اسم المستخدم المُدخل',
53.             'isEnglishLetters': 'لا يُسمح بوجود المسافات او الارقام او الرموز الخاصة او حروف غير الإنجليزية',
54.             'isUserNameTaken': 'اسم المستخدم غير متاح'
55.         },
56.         'email': {
57.             'required': ' البريد الإلكتروني مطلوب',
58.             'emailValidation': 'صيغة البريد الإلكتروني غير صحيحة',
59.             'isEmailTaken': 'البريد الإلكتروني غير متاح'
60.         },
61.         'passwordGroup': {
62.             'passwordValidation': 'كلمة السر غير متطابقة'
63.         },
64.         'password': {
65.             'required': 'كلمة االسر مطلوبة',
66.             'pattern': ' كلمة السر ست خانات ارقام وحروف وعلى الاقل حرف واحد كبير'
67.         },
68.         'confirmPassword': {
69.             'required': ' الحقل مطلوب'
70.         },
71.         'gender': {
72.             'required': 'الحقل مطلوب'
73.         },
74.         'addressDate': {
75.             'required': 'حقل تاريخ إنتهاء أقامة السكن مطلوب'
76.         },
77.         'city': {
78.             'required': 'حقل اسم المدينة مطلوب'
79.         },
80.         'state': {
81.             'required': 'حقل المنطقة مطلوب'
```

```
82.         },
83.         'zipCode': {
84.             'required': 'حقل الرمز البريدي مطلوب',
85.             'pattern': 'الرمز البريدي لابد أن يكون قيمة رقمية من خمس خانات'
86.         }
87.     };
88.
89.     currentMessageValidation = {
90.         'userName': '',
91.         'email': '',
92.         'passwordGroup': '',
93.         'password': '',
94.         'confirmPassword': '',
95.         'gender': '',
96.         'addressDate': '',
97.         'city': '',
98.         'state': '',
99.         'zipCode': ''
100.       };
101.
102.       constructor(private fb: FormBuilder, private renderer: Renderer2) { }
103.
104.       ngOnInit() {
105.           this.form = this.fb.group({
106.               userName: [null,
107.                   [
108.                       Validators.required,
109.                       Validators.pattern('.{3,}'),
110.                       CustomValidator.forbiddenNames(this.names),
111.                       CustomValidator.isEnglishLetters
112.                   ], [CustomValidator.isUserNameTaken(this.userNames)
113.                   ]
114.               ],
115.               email: [null,
116.                   [
117.                       Validators.required,
118.                       CustomValidator.emailValidation
119.                   ], [CustomValidator.isEmailTaken(this.emails)]
120.               ],
121.               passwordGroup: this.fb.group({
122.                   password: [null,
123.                       [
124.                           Validators.required,
125.                           Validators.pattern('(?=.*[A-Za-z])(?=.*[A-Z])(?!.* )(?=.*[0-
    9])[A-Za-z0-9d$@].{5,}')
126.                       ]
127.                   ],
128.                   confirmPassword: [null,
129.                       [
130.                           Validators.required
131.                       ]
132.                   ]
133.               }, { validator: CustomValidator.passwordValidation }),
```

```
134.                 gender: [null, Validators.required],
135.                 address: this.fb.group({
136.                     addressType: ['permanent'],
137.                     addressDate: [null],
138.                     city: [null, Validators.required],
139.                     state: [null, Validators.required],
140.                     zipCode: [null, [Validators.required, Validators.pattern('^[0-
    9]{5}$')]]
141.                 }),
142.                 phones: this.fb.array([this.groupPhones()]),
143.                 questions: this.fb.array([])
144.             });
145. 
146.             this.userName.valueChanges.subscribe((value: string) => {
147.                 this.userNameLength = value.length;
148.             });
149. 
150.             this.form.valueChanges.subscribe(data => {
151.                 this.loopThroughControls(this.form);
152.             });
153. 
154.             this.addressType.valueChanges.subscribe(data => {
155.                 this.addressDateValidation(data);
156.             });
157. 
158.             this.form.setControl('questions', this.setQuestions());
159. 
160.         }
161. 
162.         groupPhones(): FormGroup {
163.             return this.fb.group({
164.                 mobile: [null,
165.                     [
166.                         Validators.required,
167.                         Validators.pattern('(?:NaN|-?(?:(?:\\d+|\\d*\\.\\d+)(?:[E|e][+|-
    ]?\\d+)?|Infinity))')
168.                     ]
169.                 ],
170.                 phone: [{ value: null, disabled: true }],
171.                 mainCommunication: ['mobileCommunication']
172.             });
173.         }
174. 
175.         setQuestions(): FormArray {
176.             const formArray = new FormArray([]);
177.             this.Questions.forEach(group => {
178.                 formArray.push(this.fb.group({
179.                     question: [group.question],
180.                     answer: [null, Validators.required]
181.                 }));
182.             });
183.             return formArray;
184.         }
```

```
185.
186.       addPhones() {
187.           this.phones.push(this.groupPhones());
188.       }
189.
190.       removePhones(index: number) {
191.           this.phones.removeAt(index);
192.       }
193.
194.       phonesValidation(index: number) {
195.           const mainCommunication = this.phones.controls[index].get('mainCommunication
   ');
196.           const mobile = this.phones.controls[index].get('mobile');
197.           const phone = this.phones.controls[index].get('phone');
198.           const Reg = '(?:NaN|-?(?:(?:\\d+|\\d*\\.\\d+)(?:[E|e][+|-
   ]?\\d+)?|Infinity))';
199.           mobile.reset();
200.           phone.reset();
201.           if (mainCommunication.value === 'mobileCommunication') {
202.               mobile.setValidators([Validators.required, Validators.pattern(Reg)]);
203.               mobile.enable();
204.               this.renderer.selectRootElement('#MobileNumber' + index).focus();
205.               phone.clearValidators();
206.               phone.disable();
207.           } else {
208.               phone.setValidators([Validators.required, Validators.pattern(Reg)]);
209.               phone.enable();
210.               this.renderer.selectRootElement('#PhoneNumber' + index).focus();
211.               mobile.clearValidators();
212.               mobile.disable();
213.           }
214.           mobile.updateValueAndValidity();
215.           phone.updateValueAndValidity();
216.       }
217.
218.       addressDateValidation(data: string) {
219.           if (data === 'temporary') {
220.               this.addressDate.setValidators(Validators.required);
221.           } else {
222.               this.addressDate.clearValidators();
223.               this.addressDate.markAsUntouched();
224.               this.addressDate.reset();
225.           }
226.           this.addressDate.updateValueAndValidity();
227.       }
228.
229.       save() {
230.           console.log(this.email.errors);
231.       }
232.
233.       validateAllFormFields(formGroup: FormGroup = this.form) {
234.           Object.keys(formGroup.controls).forEach(field => {
235.               const control = formGroup.get(field);
```

```
236.             control.markAsTouched({ onlySelf: true });
237.             if (control instanceof FormGroup) {
238.                 this.validateAllFormFields(control);
239.             }
240.         });
241.     }
242.
243.     loopThroughControls(formGroup: FormGroup = this.form) {
244.         Object.keys(formGroup.controls).forEach((key: string) => {
245.             const controlName = formGroup.get(key);
246.             this.currentMessageValidation[key] = '';
247.
248.             if (controlName && controlName.invalid && (controlName.touched || contro
    lName.dirty)) {
249.                 const messages = this.messageValidation[key];
250.                 for (const controlError in controlName.errors) {
251.                     if (controlError) {
252.                         this.currentMessageValidation[key] +=
253.                             messages[controlError] + ' ';
254.                     }
255.                 }
256.             }
257.
258.             if (controlName instanceof FormGroup) {
259.                 this.loopThroughControls(controlName);
260.             }
261.         });
262.     }
263.
264.     laodData() {
265.         this.form.patchValue({
266.             userName: 'DivFaisal',
267.             email: 'test@test.com',
268.             passwordGroup: {
269.                 password: 'Aa1111',
270.                 confirmPassword: 'Aa1111'
271.             },
272.             gender: 'male',
273.             address: {
274.                 city: 'Riyadh',
275.                 state: 'ALRiyadh',
276.                 zipCode: '87678'
277.             }
278.         });
279.     }
280.
281.     get userName() {
282.         return this.form.get('userName');
283.     }
284.     get email() {
285.         return this.form.get('email');
286.     }
287.     get password() {
```

```
288.          return this.form.get('password');
289.      }
290.      get confirmPassword() {
291.          return this.form.get('confirmPassword');
292.      }
293.      get gender() {
294.          return this.form.get('gender');
295.      }
296.      get address() {
297.          return this.form.get('address');
298.      }
299.      get city() {
300.          return this.form.get('address').get('city');
301.      }
302.      get state() {
303.          return this.form.get('address').get('state');
304.      }
305.      get zipCode() {
306.          return this.form.get('address').get('zipCode');
307.      }
308.      get addressType() {
309.          return this.form.get('address').get('addressType');
310.      }
311.      get addressDate() {
312.          return this.form.get('address').get('addressDate');
313.      }
314.      get phones() {
315.          return this.form.get('phones') as FormArray;
316.      }
317.      get questions() {
318.          return this.form.get('questions') as FormArray;
319.      }
320.
```

راجع الاسطر (من 33 إلى 46)

راجع السطر 143

راجع السطر 158

راجع الاسطر (175 إلى 184)

راجع الاسطر (من 317 إلى 319)

أما في ملف app.component.html فلا يوجد اختلاف كثير عن المثال السابق فسوف نقوم بتكرار ما قمنا به من حيث ربط الأدوات وعمل loop على مصفوفة FormArray الرئيسية للنموذج ذات الأسم questions وإظهار وإخفاء التحقق من الصحة، لذلك سوف أقوم باستعراض ملف app.component.html كاملاً وأُشير إلى التعديلات والإضافات فيه مع ملاحظة أنه تم إضافة بعض أكواد css لإعطاء شكل جمالي لأدوات عرض الأسئلة وأداتي radio، كالتالي:

app.component.html ملف

```
1. <div class="container-fluid">
2.   <div class="card mx-auto col-sm-12 col-md-6 col-lg-6 col-xl-6 pt-sm-4">
3.     <form [formGroup]="form">
4.       <div class="card-header">
5.         <h4 class="text-center">Reactive Forms</h4>
6.       </div>
7.
8.       <div class="card-body">
9.         <!-- أداة إدخال اسم المستخدم -->
10.        <div class="form-group">
11.          <label>User Name</label>
12.          <div class="form-inline">
13.            <input
14.              formControlName="userName"
15.              [ngClass]="{
16.                'col-10': true,
17.                'form-control': true,
18.                'is-
   invalid': currentMessageValidation.userName || userName.hasError('isUserNameTaken'),
19.                'is-valid':
20.                  !userName.hasError('isUserNameTaken') &&
21.                  !userName.pending &&
22.                  userName.value !== null &&
23.                  userNameLength >= 3
24.              }"
25.              (blur)="loopThroughControls()"
26.              (input)="loopThroughControls()"
27.            />
28.            <label class="col-2">{{ userNameLength }}</label>
29.          </div>
30.          <!-- جزء التحقق من الصحة الخاص باسم المستخدم -->
31.          <small class="text-danger" *ngIf="currentMessageValidation.userName">
32.            {{ currentMessageValidation.userName }}
33.          </small>
34.          <small
35.            class="text-danger"
36.            *ngIf="userName.hasError('isUserNameTaken') && userName.dirty && !currentMe
   ssageValidation.userName"
37.          >
38.            اسم المستخدم غير متاح
39.          </small>
40.          <div class="alert alert-info col-10" *ngIf="userName.pending">
41.            جاري التحقق من اتاحة اسم المستخدم
42.          </div>
43.        </div>
44.
45.        <!-- أداة إدخال البريد الإلكتروني -->
46.        <div class="form-group">
47.          <label>Email</label>
48.          <input
49.            type="email"
```

```
50.             formControlName="email"
51.             [ngClass]="{
52.               'form-control': true,
53.               'is-
   invalid': currentMessageValidation.email || email.hasError('isEmailTaken'),
54.               'is-valid':
55.                 !email.hasError('isEmailTaken') &&
56.                 !email.pending &&
57.                 email.value !== null &&
58.                 email.value !== '' &&
59.                 !email.hasError('emailValidation')
60.             }"
61.             (blur)="loopThroughControls()"
62.           />
63.
64.           <!-- جزء التحقق من الصحة الخاص بالبريد الإلكتروني -->
65.           <small class="text-danger" *ngIf="currentMessageValidation.email">
66.             {{ currentMessageValidation.email }}
67.           </small>
68.           <div class="alert alert-info" *ngIf="email.pending">
69.             جاري التحقق من إتاحة البريد الإلكتروني
70.           </div>
71.         </div>
72.
73.         <!-- بداية النموذج الفرعي لكلمة السر وإعادة إدخال كلمة السر -->
74.         <div formGroupName="passwordGroup">
75.           <!-- أداة إدخال كلمة السر -->
76.           <div class="form-group">
77.             <label>Password</label>
78.             <input
79.               type="password"
80.               autocomplete="of"
81.               formControlName="password"
82.               [ngClass]="{
83.                 'form-control': true,
84.                 'is-invalid': currentMessageValidation.password
85.               }"
86.               (blur)="loopThroughControls()"
87.             />
88.             <!-- جزء التحقق من الصحة الخاص بكلمة السر -->
89.             <small class="text-danger" *ngIf="currentMessageValidation.password">
90.               {{ currentMessageValidation.password }}
91.             </small>
92.           </div>
93.
94.           <!-- أداة إعادة إدخال كلمة السر -->
95.           <div class="form-group">
96.             <label>Confirm Password</label>
97.             <input
98.               type="password"
99.               autocomplete="of"
100.                formControlName="confirmPassword"
101.                [ngClass]="{
```

```
                'form-control': true,
                'is-invalid':
                currentMessageValidation.confirmPassword ||
                currentMessageValidation.passwordGroup
              }"
              (blur)="loopThroughControls()"
            />

            <!-- جزء التحقق من الصحة الخاص بأداة إعادة إدخال كلمة السر -->
            <small
              class="text-danger"
              *ngIf="
              currentMessageValidation.confirmPassword ||
              currentMessageValidation.passwordGroup"
            >
              {{
                currentMessageValidation.confirmPassword
                  ? currentMessageValidation.confirmPassword
                  : currentMessageValidation.passwordGroup
              }}
            </small>
          </div>
        </div>
        <!-- أداة تحديد نوع الجنس ذكر أو أنثى -->
        <label class="pr-2">Gender</label>
        <div class="form-check form-check-inline">
          <input
            type="radio"
            formControlName="gender"
            id="maleGender"
            value="male"
            [ngClass]="{
              'form-check-input': true,
              'is-invalid': currentMessageValidation.gender
            }"
            (blur)="loopThroughControls()"
          />
          <label class="form-check-label" for="gender">Male</label>
        </div>
        <div class="form-check form-check-inline">
          <input
            type="radio"
            formControlName="gender"
            id="femaleGender"
            value="femail"
            [ngClass]="{
              'form-check-input': true,
              'is-invalid': currentMessageValidation.gender
            }"
            (blur)="loopThroughControls()"
          />
          <label class="form-check-label" for="femaleGender">Femail</label>
        </div>
```

```
155.           <!-- الجزء الخاص بتحقق من الصحة لأداة تحديد نوع الجنس -->
156.           <small class="text-danger" *ngIf="currentMessageValidation.gender">
157.             {{ currentMessageValidation.gender }}
158.           </small>
159.
160.           <!-- بداية النموذج الفرعي -->
161.           <fieldset class="scheduler-border" formGroupName="address">
162.             <legend class="scheduler-border">Address Information</legend>
163.
164.             <label class="pr-2" style="margin-bottom: 0px">Address Type:</label>
165.             <br />
166.             <div class="form-check form-check-inline">
167.               <input
168.                 class="form-check-input"
169.                 type="radio"
170.                 formControlName="addressType"
171.                 id="temporary"
172.                 value="temporary"
173.               />
174.               <label class="form-check-label" for="temporary">Temporary</label>
175.             </div>
176.             <div class="form-check form-check-inline">
177.               <input
178.                 class="form-check-input"
179.                 type="radio"
180.                 formControlName="addressType"
181.                 id="permanent"
182.                 value="permanent"
183.               />
184.               <label class="form-check-label" for="permanent">Permanent</label>
185.             </div>
186.             <input
187.               type="date"
188.               formControlName="addressDate"
189.               [class.has-error]="currentMessageValidation.addressDate"
190.               *ngIf="addressType.value === 'temporary'"
191.               (blur)="addressDateValidation('temporary')"
192.             />
193.             <div>
194.               <small class="text-
    danger" *ngIf="currentMessageValidation.addressDate">
195.                 {{ currentMessageValidation.addressDate }}
196.               </small>
197.             </div>
198.
199.             <!-- أداة إدخال اسم المدينة -->
200.             <div class="form-group pt-4">
201.               <label>City</label>
202.               <input
203.                 formControlName="city"
204.                 [ngClass]="{
205.                   'form-control': true,
206.                   'is-invalid': currentMessageValidation.city
```

```
207.                 }"
208.                 (blur)="loopThroughControls()"
209.               />
210.               <!-- جزء التحقق من الصحة لأداة إدخال اسم المدينة -->
211.               <small class="text-danger" *ngIf="currentMessageValidation.city">
212.                 {{ currentMessageValidation.city }}
213.               </small>
214.             </div>
215.             <!-- أداة اختيار اسم المنطقة او الولاية -->
216.             <div class="form-group">
217.               <label>State</label>
218.               <select
219.                 formControlName="state"
220.                 [ngClass]="{
221.                   'form-control': true,
222.                   'is-invalid': currentMessageValidation.state
223.                 }"
224.                 (blur)="loopThroughControls()"
225.               >
226.                 <option selected [ngValue]="null">Choose...</option>
227.                 <option *ngFor="let item of states" [value]="item">
228.                   {{ item }}
229.                 </option>
230.               </select>
231.               <!-- جزء التحقق من الصحة لأداة أختيار اسم المنطقة -->
232.               <small class="text-danger" *ngIf="currentMessageValidation.state">
233.                 {{ currentMessageValidation.state }}
234.               </small>
235.             </div>
236.             <!-- أداة إدخال الرمز البريدي -->
237.             <div class="form-group">
238.               <label>Zip Code</label>
239.               <input
240.                 formControlName="zipCode"
241.                 [ngClass]="{
242.                   'form-control': true,
243.                   'is-invalid': currentMessageValidation.zipCode
244.                 }"
245.                 (blur)="loopThroughControls()"
246.               />
247.               <!-- جزء التحقق من الصحة لأداة إدخال الرمز البريدي -->
248.               <small class="text-danger" *ngIf="currentMessageValidation.zipCode">
249.                 {{ currentMessageValidation.zipCode }}
250.               </small>
251.             </div>
252.           </fieldset>
253.
254.           <!-- بداية النموذج الفرعي الديناميكي -->
255.           <hr style="border: 1px solid silver" />
256.           <div class="form-group">
257.             <!-- زر اضافة نموذج فرعي -->
258.             <div class="col-md-offset-2 col-md-4">
259.               <button
```

```
                type="button"
                class="btn btn-info mb-3"
                (click)="addPhones()"
                [disabled]="phones.invalid">
                Add Phones
            </button>
          </div>
        </div>
        <!--phones المتمثلة في المصفوفة FormArray بداية النموذج الفرعي-->
        <fieldset
          class="scheduler-border"
          formArrayName="phones"
          *ngFor="let phoneArray of phones.controls; let i = index"
        >
          <legend class="scheduler-border">Phone #{{ i + 1 }}</legend>
          <!--FormGroup بداية النموذج الفرعي التي يتم إنشائه ديناميكياً-->
          <div [formGroupName]="i">
            <!--بداية زر حذف لكل نموذج فرعي يتم إنشائه ديناميكياً-->
            <div class="form-group">
              <button
                type="button"
                class="btn btn-danger btn-xs float-right mb-4 w-100"
                (click)="removePhones(i)"
                *ngIf="phones.length > 1"
              >
                X
              </button>
            </div>
            <!--بداية أداة أدخال رقم الجوال-->
            <div class="form-group">
              <label [attr.for]="'MobileNumber' + i">Mobile Number</label>
              <input
                type="tel"
                [id]="'MobileNumber' + i"
                formControlName="mobile"
                [ngClass]="{
                  'form-control': true,
                  'is-invalid':
                    phoneArray.get('mobile').invalid &&
                    phoneArray.get('mobile').touched &&
                    phoneArray.get('mobile').dirty
                }"
              />
              <small
                class="text-danger"
                *ngIf="
                  phoneArray.get('mobile').hasError('required') &&
                  phoneArray.get('mobile').touched &&
                  phoneArray.get('mobile').dirty
                "
              >
                حقل الهاتف الجوال مطلوب
              </small>
```

```
313.                 <small
314.                   class="text-danger"
315.                   *ngIf="
316.                     phoneArray.get('mobile').hasError('pattern') &&
317.                     phoneArray.get('mobile').touched &&
318.                     phoneArray.get('mobile').dirty
319.                   "
320.                 >
321.                    صيغة رقم الهاتف الجوال غير صحيحة
322.                 </small>
323.               </div>
324.               <!--بداية أداة إدخال رقم الهاتف الثابت-->
325.               <div class="form-group">
326.                 <label [attr.for]="'PhoneNumber' + i">Phone Number</label>
327.                 <input
328.                   type="tel"
329.                   [id]="'PhoneNumber' + i"
330.                   formControlName="phone"
331.                   [ngClass]="{
332.                     'form-control': true,
333.                     'is-invalid':
334.                       phoneArray.get('phone').invalid && phoneArray.get('phone').touch
    ed && phoneArray.get('phone').dirty
335.                   }"
336.                 />
337.                 <small
338.                   class="text-danger"
339.                   *ngIf="
340.                     phoneArray.get('phone').hasError('required') &&
341.                     phoneArray.get('phone').touched &&
342.                     phoneArray.get('phone').dirty
343.                   "
344.                 >
345.                    حقل الهاتف الثابت مطلوب
346.                 </small>
347.                 <small
348.                   class="text-danger"
349.                   *ngIf="
350.                     phoneArray.get('phone').hasError('pattern') &&
351.                     phoneArray.get('phone').touched &&
352.                     phoneArray.get('phone').dirty
353.                   "
354.                 >
355.                    صيغة رقم الهاتف الثابت غير صحيحة
356.                 </small>
357.               </div>
358.               <!--radio بداية أداتي-->
359.               <!--أداة الـ radio الأولى التي إذا كانت مختارة يتم التحقق من الصحة لأداة إدخال رقم الجوال-->
360.               <div class="form-check">
361.                 <input
362.                   class="form-check-input"
363.                   type="radio"
364.                   formControlName="mainCommunication"
```

```
365.                 [id]="'mobileCommunication' + i"
366.                 value="mobileCommunication"
367.                 (change)="phonesValidation(i)"
368.               />
369.               <label class="form-check-
   label" [attr.for]="'mobileCommunication' + i">
370.                 Communication With Mobile
371.               </label>
372.             </div>
373.             <!-- أداة الـ radio الثانية التي إذا كانت مختارة يتم التحقق من الصحة لأداة إدخال رقم الهاتف الثابت -->
374.             <div class="form-check">
375.               <input
376.                 class="form-check-input"
377.                 type="radio"
378.                 formControlName="mainCommunication"
379.                 [id]="'phoneCommunication' + i"
380.                 value="phoneCommunication"
381.                 (change)="phonesValidation(i)"
382.               />
383.               <label class="form-check-
   label" [attr.for]="'phoneCommunication' + i">
384.                 Communication With Phone
385.               </label>
386.             </div>
387.           </div>
388.         </fieldset>
389.
390.         <!-- بداية النموذج الفرعي الديناميكي الخاص بالإسئلة -->
391.         <div formArrayName="questions" *ngFor="let q of questions.controls; let i =
   index"
392.           <div class="form-row" [formGroupName]="i" *ngIf="questions.length > 0">
393.             <!--question أداة استعراض الأسئلة وتم ربطها بـ -->
394.             <div class="col-10 form-group mg-pa">
395.               <textarea
396.                 class="form-control wrap"
397.                 readonly
398.                 wrap="soft"
399.                 formControlName="question"
400.                 style="resize: none">
401.               </textarea>
402.             </div>
403.             <!--answer عرض الإجابات وتم ربطها بـ radio أداتي -->
404.             <div class="col-2">
405.               <label class="container" [attr.for]="'trueAnswer' + i">True
406.                 <input type="radio" [id]="'trueAnswer' + i" value="True" formControl
   Name="answer"/>
407.                 <span class="checkmark"></span>
408.               </label>
409.               <label class="container" [attr.for]="'falseAnswer' + i">False
410.                         <input type="radio" [id]="'falseAnswer' + i" value="False" f
   ormControlName="answer"/>
411.                 <span class="checkmark"></span>
412.               </label>
```

```
413.                 </div>
414.                 <!-- جزء التحقق من الصحة وعرض رسائل الخطأ لكل نموذج فرعي ديناميكي على حدا -->
415.                       <small class="text-
   danger" *ngIf="q.get('answer').hasError('required') && q.get('answer').touched">
416.                     لم يتم الإجابة عن السؤال المحدد
417.                 </small>
418.               </div>
419.               <hr style="border: 1px solide silver" *ngIf="questions.length > 1" />
420.             </div>
421.
422.           </div>
423.
424.           <!-- بداية أدوات الأزرار -->
425.           <div class="card-footer">
426.             <button class="btn btn-primary" (click)="save()">Save</button>
427.             <button class="btn btn-primary ml-3" (click)="laodData()">
428.               Load Data
429.             </button>
430.           </div>
431.         </form>
432.       </div>
433.   </div>
```

راجع الاسطر (من 390 إلى 422)

والآن لنستعرض النموذج على المتصفح ولنرى ما قمنا به من تعديلات:

Add Phones

Phone #1

Mobile Number

Phone Number

- (•) Communication With Mobile
- ( ) Communication With Phone

In Angular, you can pass data from parent component to child component using @Input()

- (•) True
- ( ) False

In Angular, you can pass data from child component to parent component using

- (•) True
- ( ) False

You can create local HTML reference of HTML tag using variable which starts with character

- ( ) True
- (•) False

A directive which modifies DOM hierarchy is called Structural directive ✕

- ( ) True
- ( ) False

لم يتم الإجابة عن السؤال المحدد

Save Load Data

4.5. المثال الثالث:

في هذا المثال نريد أن أن نقوم بتوليد أو تخليق مجموعة من أدوات CheckBox تمثل لغات البرمجة المفضلة لدى المستخدم حيث يتم رسمها على النموذج بشكل ديناميكي بناءً على مصفوفة معرفة مسبقاً، مع وجود زر لعمل تحديد لكافة هذه الأدوات ونفس الزر إذا تم ضغطه يتم الغاء التحديد، مع وجود التحقق من الصحة validation في حال أن المستخدم لم يقم بتحديد أي خيار من الخيارات جميعها.

الحل:

ايضاً هذا المثال مشابهه لما قبله مع بعض الأختلافات، حيث الاختلاف الجوهري هنا هي توليد أداة واحدة لعدة مرات وليس كما سبق توليد مجموعة أدوات ونعمل لهم grouping في FormGroup واحدة ومن ثم نقوم بتوليد هذا الFormGroup لعدة مرات على حسب الأحتياج، والاختلاف الجوهري الثاني هو اننا لانريد القيمة التي يُرجعها لنا النموذج وانما نُريد الاسم الموجود على الأداة، لأن أدوات CheckBox تَرجع value هو true في حال قام المستخدم بتحديدها والقيمة false في حال ان المستخدم لم يقم بتحديدها، وهذا لا يتوائم مع احتياجنا في هذا المثال لأننا نريد الأسم الموجود على الأداة فلو حدد مثلاً أداة وكان الاسم الموجود عليها java او python فنريد حفظ هذا الاسم وليس قيمتها التي تُرجعها وهو true ويتم ذلك بعدة طرق منها إنشاء مصفوفة أخرى وكلما قام المستخدم بتحديد أداة يتم نقل الاسم لهذه الأداة إلى هذه المصفوفة وكل ما الغى التحديد نزيلها من هذه المصفوفة، أما الأختلافات الأخرى فسوف اذكرها في مجموعة نقاط، كالتالي:

- ✓ طريقة بناء النموذج تختلف هنا، ففي المثال السابق كنا نستخدم setControl لإضافة النماذج والأدوات إلى المصفوفة الرئيسية FormArray، أما هنا فسوف استخدم طريقة ثانية وهي توليد هذه الأدوات جميعها في دالة منفصلة ومن ثم إسناد هذه الدالة بشكل مباشر إلى المصفوفة الرئيسية FormArray، مع العلم أن كلا الطريقتين تنفعان لهذا المثال والمثال السابق ولكن من باب التنويع سوف استعمل هذه الطريقة لهذا المثال.
- ✓ بناء التحقق من الصحة هنا يختلف قليلاً، فمع أنه يوجد دالة built-in هي required إلا أننا سوف نقوم ببناءها بإنفسنا كنوع من التدريب والتنوع، وسوف نستخدم لذلك دالتين الدالة الأولى هذي custome validation والثاني هي لإضافة هذا custome validation للأدوات checkbox وفق شروط معينة.

والآن لنقوم ببناء هذه الخطوات ومن ثم نضيفها جميعاً إلى ملف app.component.ts، كالتالي:

أولاً: ننشأ مصفوفة نصية تحتوي على مجموعة من أسماء أُطر ولغات البرمجة المشهورة، وليكن أسم هذه المصفوفة هو Favorites، كالتالي:

```
Favorites: string[] = ['java', 'C#', 'C++', 'javascript', 'HTML', 'css', 'angular', 'react', 'php'
, 'nodejs', 'python'];
```

ثانياً: نقوم بإضافة المصفوفة الرئيسية FormArray لكود إنشاء النموذج البرمجي وليكن اسمها favoritesLang في الدالة ngOnInit ونعطيها قيمة فارغة حالياً، كالتالي:

app.component.ts جزء من ملف

```
ngOnInit() {
    this.form = this.fb.group({
        userName: [null,
            [
                Validators.required,
                Validators.pattern('.{3,}'),
                CustomValidator.forbiddenNames(this.names),
                CustomValidator.isEnglishLetters
            ], [CustomValidator.isUserNameTaken(this.userNames)
            ]
        ],
        email: [null,
            [
                Validators.required,
                CustomValidator.emailValidation
            ], [CustomValidator.isEmailTaken(this.emails)]
        ],
        passwordGroup: this.fb.group({
            password: [null,
                [
                    Validators.required,
                    Validators.pattern('(?=.*[A-Za-z])(?=.*[A-Z])(?!.* )(?=.*[0-9])[A-Za-z0-9d$@].{5,}')
                ]
            ],
            confirmPassword: [null,
                [
                    Validators.required
                ]
            ]
        }, { validator: CustomValidator.passwordValidation }),
        gender: [null, Validators.required],
        address: this.fb.group({
            addressType: ['permanent'],
            addressDate: [null],
            city: [null, Validators.required],
            state: [null, Validators.required],
            zipCode: [null, [Validators.required, Validators.pattern('^[0-9]{5}$')]]
        }),
        phones: this.fb.array([this.groupPhones()]),
        questions: this.fb.array([]),
        favoritesLang: ""
    });

    this.userName.valueChanges.subscribe((value: string) => {
        this.userNameLength = value.length;
    });

    this.form.valueChanges.subscribe(data => {
        this.loopThroughControls(this.form);
    });
```

```
51.
52.     this.addressType.valueChanges.subscribe(data => {
53.         this.addressDateValidation(data);
54.     });
55.
56.     this.form.setControl('questions', this.setQuestions());
57.}
58.
```

راجع السطر 41

ثالثاً: ننشأ دالة جديدة وليكن أسمها setFavorites وهذه الدالة تُرجع FormArray، ومهمتها تخليق الأدوات بغض النظر عن نوع هذه الأدوات هل هي checkbox أو غيرها لأنه كما قلنا سابقاً أن الكلاس FormControl يرث من الكلاس الرئيسي AbstractControl لذلك angular reactive forms يُعامل الأدوات بشكل واحد فجميعها تمتلك نفس الخصائص والدوال، لذلك هذه المرة سوف نستخدم FormControl داخل FormArray لتخليق الأدوات ديناميكياً بناءً على المصفوفة Favorites، لأننا هنا نستخدم أداة واحدة نقوم بتوليدها عدد من المرات وليس مجموعة أدوات لكي نستخدم FormGroup كما كنا نفعل في الأمثلة السابقة، ويتم ذلك بالطريقة التالية:

```
1.  setFavorites(): FormArray {
2.    return this.fb.array(
3.      this.Favorites.map(() => {
4.        this.fb.control(false);
5.      })
6.    );
7.  }
```

نلاحظ استخدمنا الكلاس FormControl وليس FormGroup وممرنا له القيمة الأفتراضية false لأن قيم value لهذا النوع من الأدوات true في حال انها مختارة false في حال انها غير مختارة، ومررنا false لأننا نريد أن تكون جميع الأدوات غير مختارة في بداية تشغيل النموذج

ومن ثم نسند هذه الدالة إلى المصفوفة الرئيسية faovritesLang المنشأة في الخطوة ثانياً بدون علامات التنصيص.

رابعاً: ننشأ مصفوفة نصية فارغة أخرى وليكن أسمها selectedFavorite ومهمتها تخزين أسماء كل أداة من أدوات checkbox في حال قام المستخدم بتحديدها.

```
1.  selectedFavorite: string[] = [];
```

خامساً: ننشأ دالة وليكن أسمها selectFavorite وتستقبل باراميتر واحد هو index الخاص بكل أداة من أدوات checkbox، ومهمتها إضافة اسم الأداة المختارة إلى المصفوفة selectedFavorite في حال ان المستخدم قام بتحديدها والغاءها من هذه المصفوفة selectedFavorite في حال أن المستخدم قام بإلغاء التحديد عن الأداة، كالتالي:

```
1.  selectFavorite(index: number) {
2.    if (this.favoritesLang.controls[index].value) {
3.      this.selectedFavorite.push(this.Favorites[index]);
4.    } else {
5.      for (let i = 0; i < this.selectedFavorite.length; i++) {
6.        if (this.selectedFavorite[i] === this.Favorites[index]) {
7.          this.selectedFavorite.splice(i, 1);
8.        }
9.      }
10.   }
11. }
```

إضافتها إلى المصفوفة في حال تحديد الإداة

حذفها من المصفوفة في حال الغاء تحديد الأداة من قبل المستخدم

سادساً: ننشأ دالة أخرى وليكن اسمها checkAll، تنفذ في حال أن المستخدم قام بالضغط على زر تحديد الكل لكي يتم تحديد جميع أدوات checkbox وبنفس الوقت إذا قام المستخدم بالضغط على هذا الزر مرة أخرى يتم الغاء تحديد جميع الأدوات، وهذه تستقبل باراميتر واحد ونوعه HtmlButtonElement بمعنى أن هذه الباراميتر هو يمثل أداة الزر التي ضغطها المستخدم، والسبب في تمريرنا لهذا الباراميتر لهذه الدالة هو أن نقوم بوضع شرط في حال أن value لهذه الأداة هو checked سوف نقوم بثلاث أمور الأولى نقوم بجعل القيم value لكل الأدوات داخل المصفوفة الرئيسية favoritesLang قيمتها true – عن طريق الدالة setValue التي شرحناها سابقاً – وفي حال أصبحت القيمة true لهذه الأدوات فأنها سوف يتم التأشير عليها بشكل تلقائي، والأمر الثاني نجعل قيمة value للزر هو checked والأمر الأخير نفرغ المصفوفة النصية selectedFavorites من جميع عناصرها – في حال احتوائها على عناصر – لنجهزها للقيم الجديدة ومن ثم نعمل loop على جميع الأدوات في المصفوفة الرئيسية favoritesLang ونضيف الأسماء عليها للمصفوفة selectedFavorites، هذا في حال أن قيمة الزر كانت checked وفي حال لم تكن checked أي كانت unchecked نقوم بعمل عكس الخطوات السابقة، كالتالي:

```
checkAll(btn: HTMLButtonElement) {
  if (btn.value === 'checked') {
    this.favoritesLang.setValue(this.favoritesLang.value.map(() => true));
    btn.value = 'unchecked';
    this.selectedFavorite = [];
    for (const control in this.favoritesLang.controls) {
      if (control) {
        this.selectedFavorite.push(this.Favorites[control]);
      }
    }
  } else {
    this.favoritesLang.setValue(this.favoritesLang.value.map(() => false));
    btn.value = 'checked';
    this.selectedFavorite = [];
  }
}
```

سابعاً: ننشأ دالة التحقق من الصحة custom validation وليكن اسمها requiredCheckBoxValue وسوف نستخدم ميزة Closure فيها التي قمنا بشرحها سابقاً، ونقوم بوضع شرط في حال أن المصفوفة النصية selectedFavorites قيمتها فارغة يعني أن هنالك خطأ والمستخدم لم يقم بتحديد أي أداة من أدوات checkbox لأن عند تحديده أي أداة يُضاف اسمها لهذه المصفوفة كما فعلنا في الخطوة خامساً لذلك نضع الشرط على هذه المصفوفة ونتأكد هل هي فارغة أو لا، وبما أننا لن نتعامل مع الأداة لذلك لن نمرر باراميتر لها من النوع abstractControl كما كنا نفعل سابقاً في هذا النوع من التحقق من الصحة، لأنه كما قلت قبل قليل التعامل مع المصفوفة فقط، ويتم ذلك كالتالي:

```
requiredCheckBoxValue() {
  return (): { [key: string]: boolean } | null => {
    return this.selectedFavorite.length === 0 ? { required: true } : null;
  };
}
```

ثامناً: ننشأ دالة وليكن اسمها validatCheckBox وهذه الدالة مهمتها إضافة دالة التحقق من الصحة السابقة إلى أدوات FormArray ذات الاسم favoritesLang في حال أن المصفوفة selectedFavorites فارغة وتزيها في حال أن المصفوفة فيها عناصر، كالتالي:

```
validatCheckBox() {
  if (this.selectedFavorite.length === 0) {
    this.favoritesLang.setValidators(this.requiredCheckBoxValue());
  } else {
    this.favoritesLang.clearValidators();
  }
  this.favoritesLang.updateValueAndValidity();
}
```

تاسعاً: ننفذ الدالة السابقة في كذا موضع الموضع الأول في الدالة selectFavorite التي أنشأناها في الخطوة خامساً، لأننا نريد أن ننفذ هذه الدالة في حال أن المستخدم قام بأختيار مجموعو خيارات ثم ألغى التحديد عن جميع الخيارات ففي هذه الحالة تنفذ الدالة، كالتالي:

```
selectFavorite(index: number) {
  if (this.favoritesLang.controls[index].value) {
    this.selectedFavorite.push(this.Favorites[index]);
  } else {
    for (let i = 0; i < this.selectedFavorite.length; i++) {
      if (this.selectedFavorite[i] === this.Favorites[index]) {
        this.selectedFavorite.splice(i, 1);
      }
    }
  }
  this.validatCheckBox();
}
```

والموضع الثاني دالة checkAll التي أنشأناها في الخطوة سادساً لأننا نريد أن تنفذ في حال أن المستخدم ضغط على الزر وتم تحديد جميع الأدوات وضغط مرة أخرى وألغي التحديد ففي هذه الحال نريد إظهار رسالة خطأ بأنه لابد من تحديد أحد الخيارات، مع ملاحظة أضافة سطر برمجي آخر هنا وهو عمل حلقة loop وجعل جميع الأدوات touched، كالتالي:

```
checkAll(btn: HTMLButtonElement) {
  if (btn.value === 'checked') {
    this.favoritesLang.setValue(this.favoritesLang.value.map(() => true));
    btn.value = 'unchecked';
    this.selectedFavorite = [];
    for (const control in this.favoritesLang.controls) {
      if (control) {
        this.selectedFavorite.push(this.Favorites[control]);
      }
    }
  } else {
    this.favoritesLang.setValue(this.favoritesLang.value.map(() => false));
    btn.value = 'checked';
    this.selectedFavorite = [];
  }
  this.favoritesLang.controls.forEach(c => c.markAsTouched());
  this.validatCheckBox();
}
```

والموضع الأخير هو في الدالة ngDoCheck وهي من مجموعة دوال lifecycle hook التي تقدمها لنا angular كالدالة ngOnInit التي تعاملنا معها في أكثر من موضع بالإضافة إلى مجموعة دوال أخرى ليس هنا المقام لشرحها، مع ملاحظة أن يجب عمل لها import من angular core وايضاً نعمل لها implements في الكلاس الرئيسي للملف app.component.ts، والسبب أننا نريد تنفيذ الدالة هنا هو في حال أن المستخدم لم يقم بأختيار أي خيار وضغط على الزر save، كالتالي:

```
ngDoCheck() {
  this.validatCheckBox();
```

```
3. }
```

الآن لنقم بأضافة جميع هذه الأكواد إلى ملف app.component.ts، كالتالي:

app.component.ts ملف

```
1. import { Component, OnInit, Renderer2, DoCheck } from '@angular/core';
2.
3. import { FormGroup, FormBuilder, Validators, FormArray } from '@angular/forms';
4. import { CustomValidator } from 'src/app/shared/custom.validators';
5.
6. @Component({
7.     selector: 'app-root',
8.     templateUrl: './app.component.html',
9.     styleUrls: ['./app.component.css']
10.})
11.export class AppComponent implements OnInit, DoCheck {
12.    // tslint:disable: object-literal-key-quotes
13.    states: string[] = ['ALRiyadh', 'Makkah', 'AlSharqiyah', 'AlQasim'];
14.    userNames: string[] = ['faisal', 'DivFaisal'];
15.    form: FormGroup;
16.    userNameLength: any = '0';
17.    names: string[] = ['admin', 'administrator'];
18.    emails: string[] = ['faisal@gmail.com', 'google@gmail.com'];
19.    Favorites: string[] = ['java', 'C#', 'C++', 'javascript', 'HTML', 'css', 'angular',
    'react', 'php', 'nodejs', 'python'];
20.    selectedFavorite: string[] = [];
21.    Questions = [
22.        {
23.            question: 'In Angular, you can pass data from parent component to child com
   ponent using @Input()'
24.        },
25.        {
26.            question: 'In Angular, you can pass data from child component to parent com
   ponent using Output'
27.        },
28.        {
29.            question: 'You can create local HTML reference of HTML tag using variable w
   hich starts with character &'
30.        },
31.        {
32.            question: 'A directive which modifies DOM hierarchy is called Structural di
   rective'
33.        }
34.    ];
35.
36.    messageValidation = {
37.        'userName': {
38.            'required': 'اسم المستخدم مطلوب',
39.            'pattern': 'اسم المستخدم على الاقل ثلاث خانات',
40.            'forbiddenNames': 'لايُسمح بتسجيل اسم المستخدم المُدخل',
41.            'isEnglishLetters': 'لا يُسمح بوجود المسافات او الارقام او الرموز الخاصة او حروف غير الإنجليزية',
42.            'isUserNameTaken': 'اسم المستخدم غير متاح'
43.        },
```

```
        'email': {
            'required': ' البريد الإلكتروني مطلوب',
            'emailValidation': 'صيغة البريد الإلكتروني غير صحيحة',
            'isEmailTaken': 'البريد الإلكتروني غير متاح'
        },
        'passwordGroup': {
            'passwordValidation': 'كلمة السر غير متطابقة'
        },
        'password': {
            'required': 'كلمة السر مطلوبة',
            'pattern': ' كلمة السر ست خانات ارقام وحروف وعلى الاقل حرف واحد كبير'
        },
        'confirmPassword': {
            'required': ' الحقل مطلوب'
        },
        'gender': {
            'required': 'الحقل مطلوب'
        },
        'addressDate': {
            'required': 'حقل تاريخ إنتهاء أقامة السكن مطلوب'
        },
        'city': {
            'required': 'حقل اسم المدينة مطلوب'
        },
        'state': {
            'required': 'حقل المنطقة مطلوب'
        },
        'zipCode': {
            'required': 'حقل الرمز البريدي مطلوب',
            'pattern': 'الرمز البريدي لابد أن يكون قيمة رقمية من خمس خانات'
        }
    };

    currentMessageValidation = {
        'userName': '',
        'email': '',
        'passwordGroup': '',
        'password': '',
        'confirmPassword': '',
        'gender': '',
        'addressDate': '',
        'city': '',
        'state': '',
        'zipCode': ''
    };

    constructor(private fb: FormBuilder, private renderer: Renderer2) { }

    ngOnInit() {
        this.form = this.fb.group({
            userName: [null,
                [
                    Validators.required,
```

```
                    Validators.pattern('.{3,}'),
                    CustomValidator.forbiddenNames(this.names),
                    CustomValidator.isEnglishLetters
                ], [CustomValidator.isUserNameTaken(this.userNames)
                ]
            ],
            email: [null,
                [
                    Validators.required,
                    CustomValidator.emailValidation
                ], [CustomValidator.isEmailTaken(this.emails)]
            ],
            passwordGroup: this.fb.group({
                password: [null,
                    [
                        Validators.required,
                        Validators.pattern('(?=.*[A-Za-z])(?=.*[A-Z])(?!.* )(?=.*[0-9])[A-Za-z0-9d$@].{5,}')
                    ]
                ],
                confirmPassword: [null,
                    [
                        Validators.required
                    ]
                ]
            }, { validator: CustomValidator.passwordValidation }),
            gender: [null, Validators.required],
            address: this.fb.group({
                addressType: ['permanent'],
                addressDate: [null],
                city: [null, Validators.required],
                state: [null, Validators.required],
                zipCode: [null, [Validators.required, Validators.pattern('^[0-9]{5}$')]]
            }),
            phones: this.fb.array([this.groupPhones()]),
            questions: this.fb.array([]),
            favoritesLang: this.setFavorites()
        });

        this.userName.valueChanges.subscribe((value: string) => {
            this.userNameLength = value.length;
        });

        this.form.valueChanges.subscribe(data => {
            this.loopThroughControls(this.form);
        });

        this.addressType.valueChanges.subscribe(data => {
            this.addressDateValidation(data);
        });

        this.form.setControl('questions', this.setQuestions());
```

```
148.      }
149.
150.      ngDoCheck() {
151.          this.validatCheckBox();
152.      }
153.
154.      groupPhones(): FormGroup {
155.          return this.fb.group({
156.              mobile: [null,
157.                  [
158.                      Validators.required,
159.                      Validators.pattern('(?:NaN|-?(?:(?:\\d+|\\d*\\.\\d+)(?:[E|e][+|-
    ]?\\d+)?|Infinity))')
160.                  ]
161.              ],
162.              phone: [{ value: null, disabled: true }],
163.              mainCommunication: ['mobileCommunication']
164.          });
165.      }
166.
167.      setQuestions(): FormArray {
168.          const formArray = new FormArray([]);
169.          this.Questions.forEach(group => {
170.              formArray.push(this.fb.group({
171.                  question: [group.question],
172.                  answer: [null, Validators.required]
173.              }));
174.          });
175.          return formArray;
176.      }
177.
178.      setFavorites(): FormArray {
179.          return this.fb.array(
180.              this.Favorites.map(() => {
181.                  this.fb.control(false);
182.              })
183.          );
184.      }
185.
186.      selectFavorite(index: number) {
187.          if (this.favoritesLang.controls[index].value) {
188.              this.selectedFavorite.push(this.Favorites[index]);
189.          } else {
190.              for (let i = 0; i < this.selectedFavorite.length; i++) {
191.                  if (this.selectedFavorite[i] === this.Favorites[index]) {
192.                      this.selectedFavorite.splice(i, 1);
193.                  }
194.              }
195.          }
196.          this.validatCheckBox();
197.      }
198.
```

```
199.      checkAll(btn: HTMLButtonElement) {
200.          if (btn.value === 'checked') {
201.              this.favoritesLang.setValue(this.favoritesLang.value.map(() => true));
202.              btn.value = 'unchecked';
203.              this.selectedFavorite = [];
204.              for (const control in this.favoritesLang.controls) {
205.                  if (control) {
206.                      this.selectedFavorite.push(this.Favorites[control]);
207.                  }
208.              }
209.          } else {
210.              this.favoritesLang.setValue(this.favoritesLang.value.map(() => false));
211.              btn.value = 'checked';
212.              this.selectedFavorite = [];
213.          }
214.          this.favoritesLang.controls.forEach(c => c.markAsTouched());
215.          this.validatCheckBox();
216.      }
217.
218.      requiredCheckBoxValue() {
219.          return (): { [key: string]: boolean } | null => {
220.              return this.selectedFavorite.length === 0 ? { required: true } : null;
221.          };
222.      }
223.
224.      validatCheckBox() {
225.          if (this.selectedFavorite.length === 0) {
226.              this.favoritesLang.setValidators(this.requiredCheckBoxValue());
227.          } else {
228.              this.favoritesLang.clearValidators();
229.          }
230.          this.favoritesLang.updateValueAndValidity();
231.      }
232.
233.      addPhones() {
234.          this.phones.push(this.groupPhones());
235.      }
236.
237.      removePhones(index: number) {
238.          this.phones.removeAt(index);
239.      }
240.
241.      phonesValidation(index: number) {
242.          const mainCommunication = this.phones.controls[index].get('mainCommunication
    ');
243.          const mobile = this.phones.controls[index].get('mobile');
244.          const phone = this.phones.controls[index].get('phone');
245.          const Reg = '(?:NaN|-?(?:(?:\\d+|\\d*\\.\\d+)(?:[E|e][+|-
    ]?\\d+)?|Infinity))';
246.          mobile.reset();
247.          phone.reset();
248.          if (mainCommunication.value === 'mobileCommunication') {
249.              mobile.setValidators([Validators.required, Validators.pattern(Reg)]);
```

```
250.                 mobile.enable();
251.                 this.renderer.selectRootElement('#MobileNumber' + index).focus();
252.                 phone.clearValidators();
253.                 phone.disable();
254.             } else {
255.                 phone.setValidators([Validators.required, Validators.pattern(Reg)]);
256.                 phone.enable();
257.                 this.renderer.selectRootElement('#PhoneNumber' + index).focus();
258.                 mobile.clearValidators();
259.                 mobile.disable();
260.             }
261.             mobile.updateValueAndValidity();
262.             phone.updateValueAndValidity();
263.         }
264.
265.         addressDateValidation(data: string) {
266.             if (data === 'temporary') {
267.                 this.addressDate.setValidators(Validators.required);
268.             } else {
269.                 this.addressDate.clearValidators();
270.                 this.addressDate.markAsUntouched();
271.                 this.addressDate.reset();
272.             }
273.             this.addressDate.updateValueAndValidity();
274.         }
275.
276.         save() {
277.             console.log(this.form);
278.         }
279.
280.         loopThroughControls(formGroup: FormGroup = this.form) {
281.             Object.keys(formGroup.controls).forEach((key: string) => {
282.                 const controlName = formGroup.get(key);
283.                 this.currentMessageValidation[key] = '';
284.                 if (controlName instanceof FormArray) {
285.                     return null;
286.                 }
287.                 if (controlName && controlName.invalid && (controlName.touched || contro
    lName.dirty)) {
288.                     const messages = this.messageValidation[key];
289.                     for (const controlError in controlName.errors) {
290.                         if (controlError) {
291.                             this.currentMessageValidation[key] +=
292.                                 messages[controlError] + ' ';
293.                         }
294.                     }
295.                 }
296.                 if (controlName instanceof FormGroup) {
297.                     this.loopThroughControls(controlName);
298.                 }
299.             });
300.         }
301.
```

```
302.      laodData() {
303.          this.form.patchValue({
304.              userName: 'DivFaisal',
305.              email: 'test@test.com',
306.              passwordGroup: {
307.                  password: 'Aa1111',
308.                  confirmPassword: 'Aa1111'
309.              },
310.              gender: 'male',
311.              address: {
312.                  city: 'Riyadh',
313.                  state: 'ALRiyadh',
314.                  zipCode: '87678'
315.              }
316.          });
317.      }
318.
319.      get userName() {
320.          return this.form.get('userName');
321.      }
322.      get email() {
323.          return this.form.get('email');
324.      }
325.      get password() {
326.          return this.form.get('password');
327.      }
328.      get confirmPassword() {
329.          return this.form.get('confirmPassword');
330.      }
331.      get gender() {
332.          return this.form.get('gender');
333.      }
334.      get address() {
335.          return this.form.get('address');
336.      }
337.      get city() {
338.          return this.form.get('address').get('city');
339.      }
340.      get state() {
341.          return this.form.get('address').get('state');
342.      }
343.      get zipCode() {
344.          return this.form.get('address').get('zipCode');
345.      }
346.      get addressType() {
347.          return this.form.get('address').get('addressType');
348.      }
349.      get addressDate() {
350.          return this.form.get('address').get('addressDate');
351.      }
352.      get phones() {
353.          return this.form.get('phones') as FormArray;
354.      }
```

```
355.       get questions() {
356.           return this.form.get('questions') as FormArray;
357.       }
358.       get favoritesLang() {
359.           return this.form.get('favoritesLang') as FormArray;
360.       }
361.   }
362.
```

راجع السطر 1 (أستدعينا الدالة DoCheck)

راجع السطر 11 (وبما أنها في الأساس هي عبارة عن interface عملنا لها implements في الكلاس الرئيسي للملف)

راجع السطر 132

راجع الاسطر (من 150 إلى 152)

راجع الاسطر (من 178 إلى 231)

راجع الاسطر (من 284 إلى 286) (لابد من إضافة هذا الشرط هنا تفادياً لظهور أخطاء عند تنفيذ دالة التحقق من الصحة لأن هذه الدالة تقوم بعمل حلقة loop على جميع الأدوات في النموذج الرئيسي form وعند وجود خطأ تظهره عن طريق الرسائل الموجودة في الكائن messageValidatio بعكس هنا حيث رسائل الأخطاء كتبناها في ملف html مباشرة لذلك نقول له في حال أن الأسم البرمجي كان من النوع FormArray اخرج ولا تعمل شيء)

أما في ملف app.component.html فهو تكرار لما فعناه في الأمثلة السابقة، لذلك سوف استعرض الملف كاملاً ومن ثم وضع علامات عن الإضافات الجديدة، كالتالي:

app.component.html ملف

```
1. <div class="container-fluid">
2.   <div class="card mx-auto col-sm-12 col-md-6 col-lg-6 col-xl-6 pt-sm-4">
3.     <form [formGroup]="form">
4.       <div class="card-header">
5.         <h4 class="text-center">Reactive Forms</h4>
6.       </div>
7.
8.       <div class="card-body">
9.         <!-- أداة إدخال اسم المستخدم -->
10.        <div class="form-group">
11.          <label>User Name</label>
12.          <div class="form-inline">
13.            <input formControlName="userName" [ngClass]="{
14.                'col-10': true,
15.                'form-control': true,
16.                'is-
   invalid': currentMessageValidation.userName || userName.hasError('isUserNameTaken'),
17.                'is-valid':
18.                  !userName.hasError('isUserNameTaken') &&
19.                  !userName.pending &&
```

```
                  userName.value !== null &&
                  userNameLength >= 3
                }" (blur)="loopThroughControls()" (input)="loopThroughControls()" />
              <label class="col-2">{{ userNameLength }}</label>
            </div>
            <!-- جزء التحقق من الصحة الخاص باسم المستخدم -->
            <small class="text-danger" *ngIf="currentMessageValidation.userName">
              {{ currentMessageValidation.userName }}
            </small>
            <!-- <small
              class="text-danger"
              *ngIf="userName.hasError('isUserNameTaken') && userName.dirty && !currentMessageValidation.userName"
            >
              اسم المستخدم غير متاح
            </small> -->
            <div class="alert alert-info col-10" *ngIf="userName.pending">
              جاري التحقق من اتاحة اسم المستخدم
            </div>
          </div>

          <!-- أداة إدخال البريد الإلكتروني -->
          <div class="form-group">
            <label>Email</label>
            <input type="email" formControlName="email" [ngClass]="{
                'form-control': true,
                'is-invalid': currentMessageValidation.email || email.hasError('isEmailTaken'),
                'is-valid':
                  !email.hasError('isEmailTaken') &&
                  !email.pending &&
                  email.value !== null &&
                  email.value !== '' &&
                  !email.hasError('emailValidation')
              }" (blur)="loopThroughControls()" />

            <!-- جزء التحقق من الصحة الخاص بالبريد الإلكتروني -->
            <small class="text-danger" *ngIf="currentMessageValidation.email">
              {{ currentMessageValidation.email }}
            </small>
            <div class="alert alert-info" *ngIf="email.pending">
              جاري التحقق من إتاحة البريد الإلكتروني
            </div>
          </div>

          <!-- بداية النموذج الفرعي لكلمة السر وإعادة إدخال كلمة السر -->
          <div formGroupName="passwordGroup">
            <!-- أداة إدخال كلمة السر -->
            <div class="form-group">
              <label>Password</label>
              <input type="password" autocomplete="of" formControlName="password" [ngClass]="{
                  'form-control': true,
```

```
70.                 'is-invalid': currentMessageValidation.password
71.               }" (blur)="loopThroughControls()" />
72.             <!-- جزء التحقق من الصحة الخاص بكلمة السر -->
73.             <small class="text-danger" *ngIf="currentMessageValidation.password">
74.               {{ currentMessageValidation.password }}
75.             </small>
76.           </div>
77.
78.           <!-- أداة إعادة إدخال كلمة السر -->
79.           <div class="form-group">
80.             <label>Confirm Password</label>
81.             <input type="password" autocomplete="of" formControlName="confirmPassword"
   [ngClass]="{
82.                 'form-control': true,
83.                 'is-
   invalid': currentMessageValidation.confirmPassword || currentMessageValidation.password
   Group
84.               }" (blur)="loopThroughControls()" />
85.
86.             <!-- جزء التحقق من الصحة الخاص بأداة إعادة إدخال كلمة السر -->
87.             <small class="text-danger"
88.               *ngIf="currentMessageValidation.confirmPassword || currentMessageValidati
   on.passwordGroup">
89.               {{
90.                 currentMessageValidation.confirmPassword
91.                   ? currentMessageValidation.confirmPassword
92.                   : currentMessageValidation.passwordGroup
93.               }}
94.             </small>
95.           </div>
96.         </div>
97.         <!-- أداة تحديد نوع الجنس ذكر أو أنثى -->
98.         <label class="pr-2">Gender</label>
99.         <div class="form-check form-check-inline">
100.             <input type="radio" formControlName="gender" id="maleGender" value="male"
   [ngClass]="{
101.                 'form-check-input': true,
102.                 'is-invalid': currentMessageValidation.gender
103.               }" (blur)="loopThroughControls()" />
104.             <label class="form-check-label" for="gender">Male</label>
105.           </div>
106.           <div class="form-check form-check-inline">
107.             <input type="radio" formControlName="gender" id="femaleGender" value="fema
   il" [ngClass]="{
108.                 'form-check-input': true,
109.                 'is-invalid': currentMessageValidation.gender
110.               }" (blur)="loopThroughControls()" />
111.             <label class="form-check-label" for="femaleGender">Femail</label>
112.           </div>
113.           <!-- الجزء الخاص بتحقق من الصحة لأداة تحديد نوع الجنس -->
114.           <small class="text-danger" *ngIf="currentMessageValidation.gender">
115.             {{ currentMessageValidation.gender }}
116.           </small>
```

```
117.
118.          <!-- بداية النموذج الفرعي -->
119.          <fieldset class="scheduler-border" formGroupName="address">
120.            <legend class="scheduler-border">Address Information</legend>
121.
122.            <label class="pr-2" style="margin-bottom: 0px">Address Type:</label>
123.            <br />
124.            <div class="form-check form-check-inline">
125.              <input class="form-check-
   input" type="radio" formControlName="addressType" id="temporary"
126.                value="temporary" />
127.              <label class="form-check-label" for="temporary">Temporary</label>
128.            </div>
129.            <div class="form-check form-check-inline">
130.              <input class="form-check-
   input" type="radio" formControlName="addressType" id="permanent"
131.                value="permanent" />
132.              <label class="form-check-label" for="permanent">Permanent</label>
133.            </div>
134.            <input type="date" formControlName="addressDate" [class.has-
   error]="currentMessageValidation.addressDate"
135.              *ngIf="addressType.value === 'temporary'" (blur)="addressDateValidation(
   'temporary')" />
136.            <div>
137.              <small class="text-danger" *ngIf="currentMessageValidation.addressDate">
138.                {{ currentMessageValidation.addressDate }}
139.              </small>
140.            </div>
141.
142.            <!-- أداة إدخال اسم المدينة -->
143.            <div class="form-group pt-4">
144.              <label>City</label>
145.              <input formControlName="city" [ngClass]="{
146.                  'form-control': true,
147.                  'is-invalid': currentMessageValidation.city
148.                }" (blur)="loopThroughControls()" />
149.              <!-- جزء التحقق من الصحة لأداة إدخال اسم المدينة -->
150.              <small class="text-danger" *ngIf="currentMessageValidation.city">
151.                {{ currentMessageValidation.city }}
152.              </small>
153.            </div>
154.            <!-- أداة اختيار اسم المنطقة او الولاية -->
155.            <div class="form-group">
156.              <label>State</label>
157.              <select formControlName="state" [ngClass]="{
158.                  'form-control': true,
159.                  'is-invalid': currentMessageValidation.state
160.                }" (blur)="loopThroughControls()">
161.                <option selected [ngValue]="null">Choose...</option>
162.                <option *ngFor="let item of states" [value]="item">
163.                  {{ item }}
164.                </option>
165.              </select>
```

```
166.               <!-- جزء التحقق من الصحة لأداة أختيار اسم المنطقة -->
167.               <small class="text-danger" *ngIf="currentMessageValidation.state">
168.                 {{ currentMessageValidation.state }}
169.               </small>
170.             </div>
171.             <!-- أداة إدخال الرمز البريدي -->
172.             <div class="form-group">
173.               <label>Zip Code</label>
174.               <input formControlName="zipCode" [ngClass]="{
175.                   'form-control': true,
176.                   'is-invalid': currentMessageValidation.zipCode
177.                 }" (blur)="loopThroughControls()" />
178.               <!-- جزء التحقق من الصحة لأداة إدخال الرمز البريدي -->
179.               <small class="text-danger" *ngIf="currentMessageValidation.zipCode">
180.                 {{ currentMessageValidation.zipCode }}
181.               </small>
182.             </div>
183.           </fieldset>
184.
185.           <!-- بداية النموذج الفرعي الديناميكي -->
186.           <hr style="border: 1px solid silver" />
187.           <div class="form-group">
188.             <!-- زر إضافة نموذج فرعي -->
189.             <div class="col-md-offset-2 col-md-4">
190.               <button type="button" class="btn btn-info mb-
   3" (click)="addPhones()" [disabled]="phones.invalid">
191.                 Add Phones
192.               </button>
193.             </div>
194.           </div>
195.           <!--phones المتمثلة في المصفوفة FormArray بداية النموذج الفرعي -->
196.           <fieldset class="scheduler-border" formArrayName="phones"
197.             *ngFor="let phoneArray of phones.controls; let i = index">
198.             <legend class="scheduler-border">Phone #{{ i + 1 }}</legend>
199.             <!--FormGroup بداية النموذج الفرعي التي يتم إنشائه ديناميكياً -->
200.             <div [formGroupName]="i">
201.               <!-- بداية زر حذف لكل نموذج فرعي يتم إنشائه ديناميكياً -->
202.               <div class="form-group">
203.                 <button type="button" class="btn btn-danger btn-xs float-right mb-4 w-
   100" (click)="removePhones(i)"
204.                   *ngIf="phones.length > 1">
205.                   X
206.                 </button>
207.               </div>
208.               <!-- بداية أداة أدخال رقم الجوال -->
209.               <div class="form-group">
210.                 <label [attr.for]="'MobileNumber' + i">Mobile Number</label>
211.                 <input type="tel" [id]="'MobileNumber' + i" formControlName="mobile" [
   ngClass]="{
212.                     'form-control': true,
213.                     'is-invalid':
214.                       phoneArray.get('mobile').invalid &&
215.                       phoneArray.get('mobile').touched &&
```

```
                        phoneArray.get('mobile').dirty
                    }" />
                  <small class="text-danger" *ngIf="
                      phoneArray.get('mobile').hasError('required') &&
                      phoneArray.get('mobile').touched &&
                      phoneArray.get('mobile').dirty
                    ">
                    حقل الهاتف الجوال مطلوب
                  </small>
                  <small class="text-danger" *ngIf="
                      phoneArray.get('mobile').hasError('pattern') &&
                      phoneArray.get('mobile').touched &&
                      phoneArray.get('mobile').dirty
                    ">
                    صيغة رقم الهاتف الجوال غير صحيحة
                  </small>
                </div>
                <!--بداية أداة إدخال رقم الهاتف الثابت-->
                <div class="form-group">
                  <label [attr.for]="'PhoneNumber' + i">Phone Number</label>
                  <input type="tel" [id]="'PhoneNumber' + i" formControlName="phone" [ng
   Class]="{
                      'form-control': true,
                      'is-invalid':
                        phoneArray.get('phone').invalid && phoneArray.get('phone').touch
   ed && phoneArray.get('phone').dirty
                    }" />
                  <small class="text-danger" *ngIf="
                      phoneArray.get('phone').hasError('required') &&
                      phoneArray.get('phone').touched &&
                      phoneArray.get('phone').dirty
                    ">
                    حقل الهاتف الثابت مطلوب
                  </small>
                  <small class="text-danger" *ngIf="
                      phoneArray.get('phone').hasError('pattern') &&
                      phoneArray.get('phone').touched &&
                      phoneArray.get('phone').dirty
                    ">
                    صيغة رقم الهاتف الثابت غير صحيحة
                  </small>
                </div>
                <!--radio بداية أداتي-->
                <!--أداة الـradio الأولى التي إذا كانت مختارة يتم التحقق من الصحة لأداة إدخال رقم الجوال-->
                <div class="form-check">
                  <input class="form-check-
   input" type="radio" formControlName="mainCommunication"
                    [id]="'mobileCommunication' + i" value="mobileCommunication" (change
   )="phonesValidation(i)" />
                  <label class="form-check-
   label" [attr.for]="'mobileCommunication' + i">
                    Communication With Mobile
                  </label>
```

```
264.              </div>
265.              <!-- أداة الـ radio الثانية التي إذا كانت مختارة يتم التحقق من الصحة لأداة إدخال رقم الهاتف الثابت -->
266.              <div class="form-check">
267.                <input class="form-check-
   input" type="radio" formControlName="mainCommunication"
268.                  [id]="'phoneCommunication' + i" value="phoneCommunication" (change)=
   "phonesValidation(i)" />
269.                <label class="form-check-label" [attr.for]="'phoneCommunication' + i">
270.                  Communication With Phone
271.                </label>
272.              </div>
273.            </div>
274.          </fieldset>
275.
276.          <!-- بداية النموذج الفرعي الديناميكي الخاص بالإسئلة -->
277.          <div formArrayName="questions" *ngFor="let q of questions.controls; let i =
   index" <div class="form-row"
278.            [formGroupName]="i" *ngIf="questions.length > 0">
279.            <!--question أداة استعراض الأسئلة وتم ربطها بـ -->
280.            <div class="col-10 form-group mg-pa">
281.              <textarea class="form-
   control wrap" readonly wrap="soft" formControlName="question" style="resize: none">
282.                </textarea>
283.            </div>
284.            <!--answer عرض الإجابات وتم ربطها بـ radio أداتي -->
285.            <div class="col-2">
286.              <label class="container" [attr.for]="'trueAnswer' + i">True
287.                <input type="radio" [id]="'trueAnswer' + i" value="True" formControlNa
   me="answer" />
288.                <span class="checkmark"></span>
289.              </label>
290.              <label class="container" [attr.for]="'falseAnswer' + i">False
291.                <input type="radio" [id]="'falseAnswer' + i" value="False" formControl
   Name="answer" />
292.                <span class="checkmark"></span>
293.              </label>
294.            </div>
295.            <!-- جزء التحقق من الصحة وعرض رسائل الخطأ كل نموذج فرعي ديناميكي على حدا -->
296.            <small class="text-
   danger" *ngIf="q.get('answer').hasError('required') && q.get('answer').touched">
297.                لم يتم الإجابة عن السؤال المحدد
298.            </small>
299.          </div>
300.          <hr style="border: 1px solide silver" *ngIf="questions.length > 1" />
301.        </div>
302.
303.        <!--check box بداية النموذج الديناميكي لاستعراض الهوايات عن طريق أدوات -->
304.        <fieldset class="scheduler-border" formArrayName="favoritesLang">
305.          <legend class="scheduler-border">Favorite Programing Language</legend>
306.          <!-- بداية حلقة التكرار وربط الأدوات عن طريق الإندكس لكل أداة -->
307.          <div
308.              class="form-check form-check-inline"
309.              *ngFor="let control of favoritesLang.controls; let i = index">
```

```
310.
311.             <input
312.                 class="form-check-input"
313.                 [formControlName]="i"
314.                 type="checkbox"
315.                 [id]="'inlineCheckbox' + i"
316.                 (change)="selectFavorite(i)"
317.             />
318.             <!-- إظهار الأسماء على الأدوات بواسطة الـ index الخاص بها -->
319.             <label
320.                 class="form-check-label"
321.                 [attr.for]="'inlineCheckbox' + i"
322.             >
323.                {{ Favorites[i] }}
324.             </label>
325.           </div>
326.           <!-- زر تحديد أو الغاء تحديد الأدوات -->
327.           <div class="pt-3" *ngIf="favoritesLang.length">
328.             <button
329.               class="btn btn-dark btn-sm"
330.               #checkUncheckButton
331.               (click)="checkAll(checkUncheckButton)"
332.               value="checked">
333.               check/uncheck All
334.             </button>
335.           </div>
336.           <div>
337.             <!-- جزء التحقق من الصحة واظهار الأدوات -->
338.             <small class="text-
   danger" *ngIf="favoritesLang.hasError('required') && favoritesLang.touched">
339.                 الرجاء اختيار خيار واحد على الأقل
340.             </small>
341.           </div>
342.         </fieldset>
343.     </div>
344.
345.     <!-- بداية أدوات الأزرار -->
346.     <div class="card-footer">
347.       <button class="btn btn-primary" (click)="save()">Save</button>
348.       <button class="btn btn-primary ml-3" (click)="laodData()">
349.         Load Data
350.       </button>
351.     </div>
352.     </form>
353.   </div>
354.   </div>
355.
```

راجع الاسطر (من 304 إلى 326)

راجع السطر 313 (ربط الأدوات عن طريق index الخاص بكل أداة بعكس الأمثلة السابقة كنا ربط الFormGroup بالindex والأدوات بالاسم البرمجي)

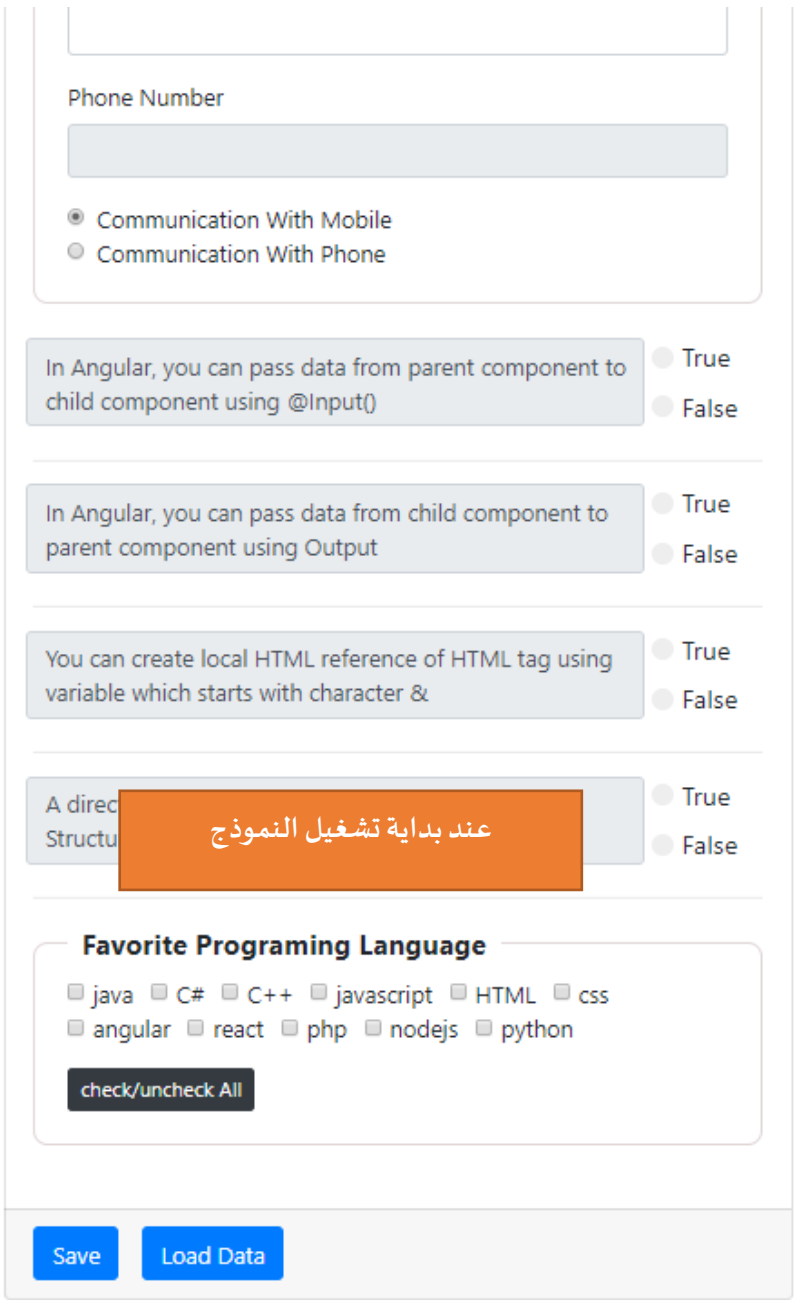
راجع السطر 316 (تنفيذ الدالة selectFavorites في الحدث change لأدوات checkbox وممرنا لها الindex للأداة المُخزنة قيمته في المتغير i والذي قمنا بتعريفه في حلقة for في السطر 309)

راجع السطر 323 (إظهار الاسماء في المصفوفة ووضعها في الأدوات)

راجع السطر 331 (تنفيذ الدالة checkAll في حدث click للزر ومررنا لها reference template للزر والذي قمنا بتعريفه بالسطر 330)

راجع السطر 332 (الخاصية value للزر جعلنا قيمتها الأفتراضية checked وهي الخاصية التي استفدنا منها في الدالة checkAll التي شرحناها سابقاً مع العلم أن المسميات اختيارية checked...الخ)

الآن لنرى التعديلات على النموذج في المتصفح:


Phone Number
Communication With Mobile
Communication With Phone
In Angular, you can pass data from parent component to child component using @Input()
True
False
In Angular, you can pass data from child component to parent component using Output
True
False
You can create local HTML reference of HTML tag using variable which starts with character &
True
False
A direc
Structu
True
False
عند بداية تشغيل النموذج
Favorite Programing Language
java C# C++ javascript HTML css
angular react php nodejs python
check/uncheck All
Save
Load Data

Phone Number

Communication With Mobile
Communication With Phone

In Angular, you can pass data from parent component to child component using @Input()
True
False

In Angular, you can pass data from child component to parent component using Output
True
False

You can create local HTML reference of HTML tag using variable which starts with character &
True
False

A directive which modifies DOM hierarchy is called Structural d
True
False

في حال الضغط على زرتحديد الكل

Favorite Programing Language

java C# C++ javascript HTML css
angular react php nodejs python

check/uncheck All

Save Load Data

Phone Number

Communication With Mobile
Communication With Phone

In Angular, you can pass data from parent component to child component using @Input()
True
False

In Angular, you can pass data from child component to parent component using Output
True
False

You can create local HTML reference of HTML tag using variable which starts with character &
True
False

A directive which modifies DOM hierarchy is called
True

في حال الضغط مرة أخرى على الزرونلاحظ ظهوررسالة الخطأ

Favorite Programing Language

java C# C++ javascript HTML css
angular react php nodejs python

check/uncheck All

الرجاء اختيار خيار واحد على الأقل

Save Load Data

Phone Number

Communication With Mobile
Communication With Phone

In Angular, you can pass data from parent component to child component using @Input()

True
False

In Angular, you can pass data from child component to parent component using Output

True
False

You can create local HTML reference of HTML tag using variable which starts with character &

True
False

True

في حال اختيار أي خيار من الخيار من المستخدم تختفي رسالة الخطأ

Favorite Programing Language

java C# C++ javascript HTML css
angular react php nodejs python

check/uncheck All

Save Load Data

Phone Number

Communication With Mobile
Communication With Phone

In Angular, you can pass data from parent component to child component using @Input()
True
False

In Angular, you can pass data from child component to parent component using Output
True
False

You can create local HTML reference of HTML tag using variable which starts with character &
True
False

A directive which modifies DOM hierarchy is called
True

في حال الغاء تحديد الأدوات من قبل المستخدم تظهر رسالة الخطأ مرة أخرى

Favorite Programing Language

java C# C++ javascript HTML css
angular react php nodejs python

check/uncheck All

الرجاء اختيار خيار واحد على الأقل

Save Load Data

5.5. التحقق من الصحة على مستوى النموذج Form Validations وأرسال البيانات Submite Data:

بعد الإنتهاء من بناء النموذج والتحقق من صحته، نريد أرسال هذه البيانات إلى السيرفر، وذلك من خلال وجود زر في النموذج وإذا ضغطه المستخدم تُرسل هذه البيانات، وبما أننا هنا نتكلم عن Front-End فقط فلذلك سوف أتكلم عن كيفية الوصول إلى قيم هذا النموذج وتجميعها بحيث تصبح جاهزه للأرسال، ولكن قبل الوصول إلى قيم النموذج لابد من وجود طريقة أو آلية معينة تتيح لنا التعامل مع المستخدم في حال قام بالضغط على الزر ولم تكتمل البيانات أو هنالك أخطاء في التحقق من الصحة، وفي الحقيقة هنالك طريقتين مشهورة الأولى هي تعطيل هذا الزر إذا كانت البيانات غير مكتملة أو يوجد هنالك أخطاء ويُفعل في حال اكتمال البيانات ولايوجد أي أخطاء، وهذا ما عملناه في TDF، اما الطريقة الثانية فهي نترك الزر مفعل وكلما ضغط المستخدم على الزر تظهر له رسالة عامة أن هنالك أخطاء في النموذج بالإضافة إلى إظهار الأخطاء المحددة لكل أداة، وهذا ما سوف نعمله هنا لكي يصبح هذا الكتاب شامل لجميع الطرق، مع العلم أن هذه الطرق تُسمى التحقق من الصحة على مستوى النموذج وهذا ماسوف نتكلم عنه الآن.

1.5.5. التحقق من الصحة على مستوى النموذج:

النموذج الأساسي والذي اسميناه form هو من نوع الكلاس FormGroup وكما قلنا أكثر من مرة أن هذا النوع يرث من الكلاس AbstractControl لذلك يمتلك جميع خصائصه ودواله وبما أن النموذج الأساسي form هو من النوع FormGroup لذلك هو ايضاً يمتلك جميع الخصائص والدوال التي تعاملنا معها سابقاً في الأدوات والنماذج الفرعية مثل إضافة البيانات والتحقق من الصحة وغيره ومايهمنا هنا هو الخاصية invalid التي تكون قيمتها true في حال أن النموذج فيه أخطاء، وتصبح قيمتها false في حال أن النموذج لا يوجد فيه أخطاء وفي حالة valid، وسوف نتعامل مع هذه الخاصية من خلال وضع شرط في الدالة save التي أنشاناها سابقاً وتُنفذ في الحدث click للزر المسمى save، كالتالي:

```
save() {
    if (this.form.invalid) {

    هنا يتم كتابة logic في حال أن النموذج في حالة invalid

    } else {

    هنا يتم كتابة logic في حال أن النموذج في حالة valid

    }
}
```

في حال تحقق الشرط الأول وكان invalid قيمتها true نريد ننفذ دالتين، دالة منشأة سابقاً ودالة سوف ننشأها حالياً، أما الدالة المنشأ سابقاً هي الدالة loopThroughControls والتي أنشاناها لكي تعمل loop على أدوات النموذج وتظهر خطأ في حال أن هنالك أخطاء اما الثانية وليكن أسمها touchedAllFormFields ومهمتها ايضاً عمل loop على جميع أدوات النموذج ولكن هنا تجعل الأدوات في حالة touched أي بمعنى تم لمسها أي الخاصية touched لجميع الأدوات قيمتها true، ونستفيد من جعل قيمة الخاصية touched هو true لأننا لو لاحظنا في جميع شروط التحقق من الصحة نحدد أنه في حال كان هنالك خطأ أن لايظهر هذا الخطأ إلا إذا تم لمس الأداة، لذلك نستفيد من هذا الأمر بالقيام بالمس جميع الأدوات برمجياً أي جعل قيمة الخاصية touched لهم true وفي هذه الحالة سوف تظهر أي رسائل الخطأ للأدوات في حال كان هنالك خطأ في أداة

معينة، أما السبب في تنفيذ كلا الدالتين هنا لأن الدالة touchedAllFormFields تُظهر رسائل الخطأ الموجودة التي كتبناها في ملف html مباشرة أما رسائل الخطأ الموجودة في الكائن messageValidation فلاتظهر إلا بتنفيذ الدالة loopThroughControls، ومن ناحية محتوى الدالة touchedAllFormFields فهو مشابه للدالة loopThroughControls والفرق انها هنا تجعل الخاصية touched لكل الأدوات true عن طريق الدالة markAsTouched، كالتالي:

```
1.  touchedAllFormFields(formGroup: FormGroup = this.form) {
2.    Object.keys(formGroup.controls).forEach(field => {
3.      const control = formGroup.get(field);
4.      control.markAsTouched();
5.      if (control instanceof FormGroup) {
6.        this.touchedAllFormFields(control);
7.      }
8.      if (control instanceof FormArray) {
9.        for (const c of control.controls) {
10.         if (c instanceof FormGroup) {
11.           this.touchedAllFormFields(c);
12.         }
13.       }
14.     }
15.   });
16. }
```

الآن لنضيف هذين الدالتين إلى الدالة save، كالتالي:

```
1.  save() {
2.    if (this.form.invalid) {
3.      this.touchedAllFormFields();
4.      this.loopThroughControls();
5.    } else {
6.
7.
8.    }
9.    console.log(this.form.value);
10. }
```

إلى الآن الوضع جيد لكن هنالك مشكلة بسيطة وهي أننا عندما ننتقل إلى ملف app.component.html لإضافة رسالة الخطأ العامة التي (لا تظهر) إلا إذا قام المستخدم بالضغط على الزر save، سوف نضع لها شرط أن لاتظهر هذه الرسالة إلا إذا كان النموذج في حالة invalid، والمشكلة هنا أن النموذج في بداية تشغيله يكون في حالة invalid وهنا تظهر هذه المشكلة وهي ظهور رسالة الخطأ في بداية تشغيل النموذج، وحل هذه المشكلة بسيط وهو عن طريق تعريف متغير من النوع boolean وليكن اسمه isInvalidForm، ولا نعطيه أي قيمة مبدأية، ومهمة هذا المتغير هو تغيير حالته ليصبح true إذا كان النموذج في حالة invalid وتتغير قيمته إلى false إذا كان النموذج valid، ويتم تتغير قيمة هذا المتغير في الدالة save، لذلك لنقم أولاً بتعريف متغير مع مجموعة المتغيرات في بداية ملف app.component.ts، كالتالي:

```
1.  isInvalidForm: boolean;
```

ومن ثم نقوم بتغيير قيمته في الدالة save، كالتالي:

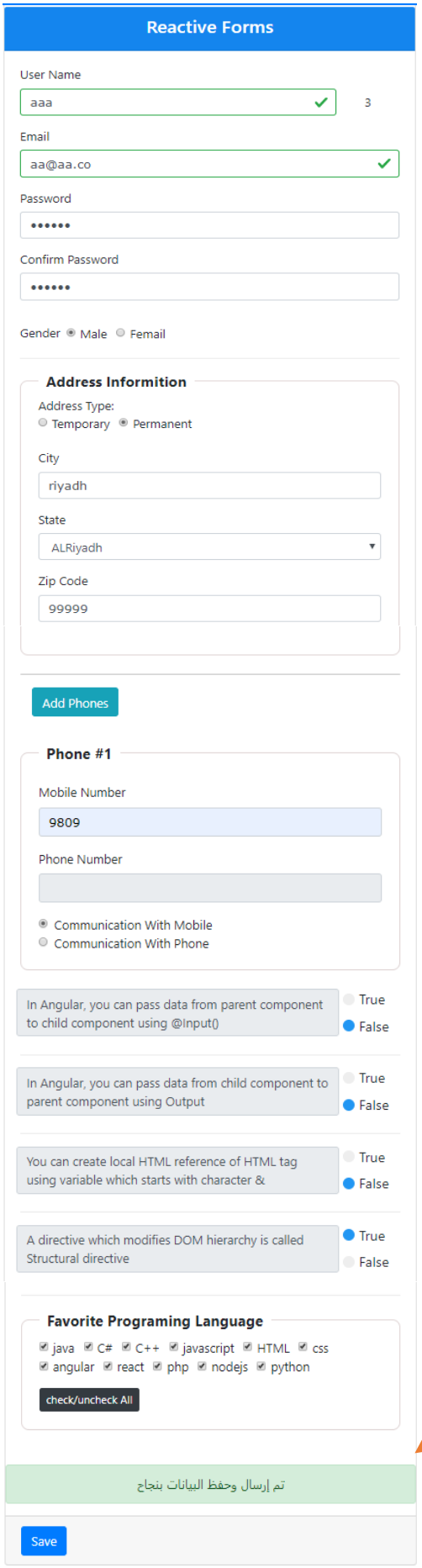
```
1.  save() {
2.    if (this.form.invalid) {
3.      this.isInvalidForm = true;
4.      this.touchedAllFormFields();
5.      this.loopThroughControls();
```

```
    } else {
      this.isInvalidForm = false;
    }
  }
```

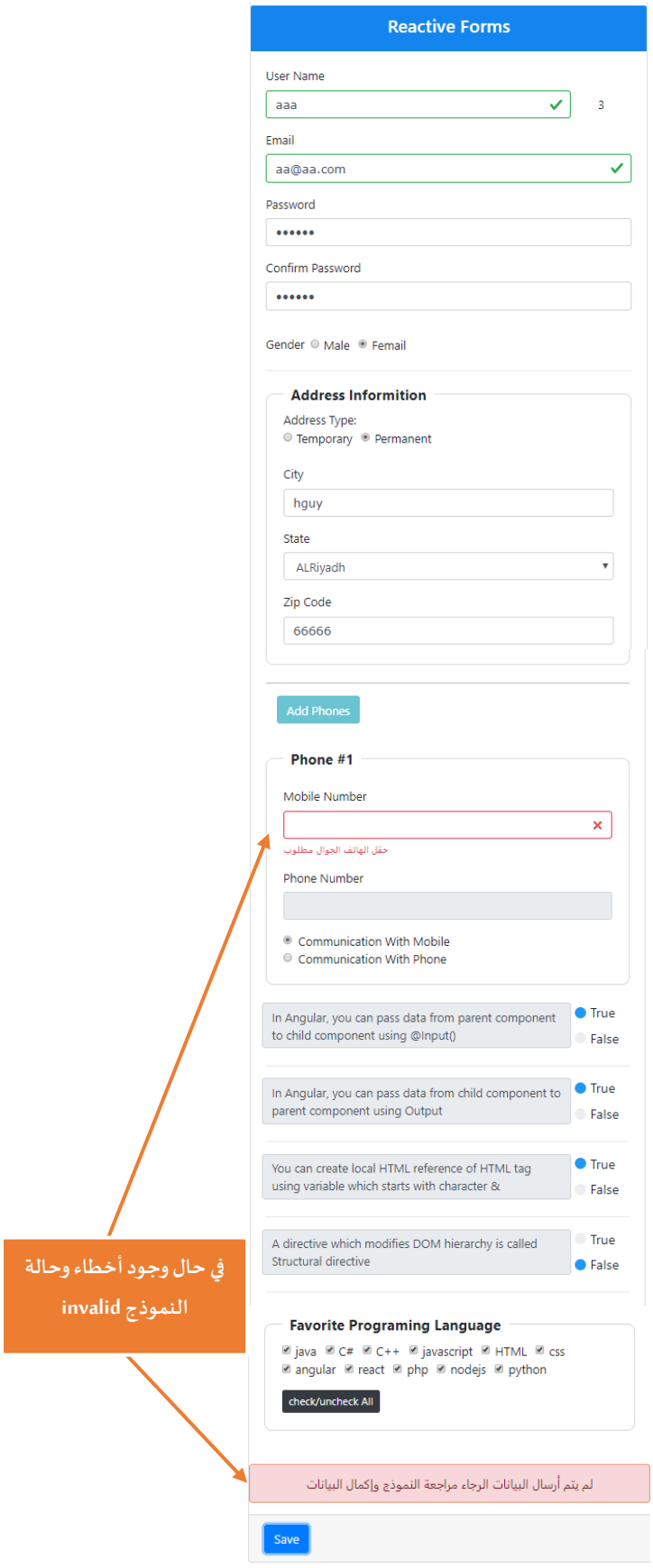
الآن لننتقل إلى ملف app.component.html، ونضيف عنصرين html وهما عبارة عن div وكل واحد منهما يحتوي على رسالة، الأولى تحتوي على رسالة عامة في حال أن هنالك أخطاء ولايظهر هذا div إلا إذا كان النموذج في حالة invalid وقيمة المتغير isInvalidForm قيمته true، والdiv الثانية لا تظهر إلا إذا كان النموذج في حالة valid والمتغير قيمته false، وبما أن الأكواد في ملف html أصبحت كثيرة لذلك سوف استعرض الإضافات فقط، كالتالي:

```
<!--جزء الرسائل العامة-->
<div class="alert alert-danger text-center" *ngIf="form.invalid && isInvalidForm">
  لم يتم أرسال البيانات الرجاء مراجعة النموذج وإكمال البيانات
</div>
<div class="alert alert-success text-center" *ngIf="!form.invalid && !isInvalidForm ">
  تم إرسال وحفظ البيانات بنجاح
</div>

<!-- بداية أدوات الأزرار -->
<div class="card-footer">
  <button class="btn btn-primary" (click)="save()">Save</button>
</div>
```

الآن لنشاهد النموذج بعد التعديلات:



شكل النموذج في حال كونه valid ولايوجد به أخطاء


Reactive Forms
User Name
aaa
3
Email
aa@aa.com
Password
Confirm Password
Gender Male Femail
Address Information
Address Type:
Temporary Permanent
City
hguy
State
ALRiyadh
Zip Code
66666
Add Phones
Phone #1
Mobile Number
حقل الهاتف الجوال مطلوب
Phone Number
Communication With Mobile
Communication With Phone
In Angular, you can pass data from parent component to child component using @Input()
True
False
In Angular, you can pass data from child component to parent component using Output
True
False
You can create local HTML reference of HTML tag using variable which starts with character &
True
False
A directive which modifies DOM hierarchy is called Structural directive
True
False
Favorite Programing Language
java C# C++ javascript HTML css
angular react php nodejs python
check/uncheck All
لم يتم أرسال البيانات الرجاء مراجعة النموذج وإكمال البيانات
Save
في حال وجود أخطاء وحالة النموذج invalid

2.5.5. أرسال قيم النموذج:

بعد معرفتنا لكيفية التحقق من الصحة على مستوى النموذج، نستطيع الآن أرسال بيانات وقيم النموذج، وكما قلنا سابقاً أنه لايوجد لدينا سيرفر Back-End لأرسال البيانات إليه وحفظها في قاعدة البيانات، ولكن سوف استعرض البيانات في console الخاص بالمتصفح google chrome، وكل الذي سوف نقوم به هو كتابة هذا الأمر في الدالة save، كالتالي:

```
1.  save() {
2.    if (this.form.invalid) {
3.      this.isInvalidForm = true;
4.      this.touchedAllFormFields();
5.      this.loopThroughControls();
6.    } else {
7.      this.isInvalidForm = false;
8.      console.log(this.form.value)
9.    }
10. }
```

والآ لنقم بتشغيل النموذج على المتصفح ومن ثم نقوم بتعبئة النموذج ببيانات صحيحة ومكتملة، ومن ثم نضغط على الزر save لكي ننفذ الدالة التي لها نفس الأسم، وأخيراً نفتح console الخاص بالمتصفح لكي نرى النتيجة، حيث تكون النتيجة كالتالي:

ما سبق هو القيم في النموذج كاملة، وهي كما توقعنا ونفس النتيجة التي ننتظرها إلا الجزء الخاص بالمصفوفة FormArray والتي وضعت عليها مربع أحمر نلاحظ أنها تحتوي على 11 عنصر والindex الخاص بها يبدأ من الصفر إلى 10 لكن القيم لكل

index ليس كما توقعنا ولا هي النتيجة التي ننتظر الحصول عليها فجميع القيم مابين null او true او false، مع العلم أنني في النموذج أخترت الخيارات التالية:

والقيم التي ننتظر أن تحتويها او التي يجب أن تكون هي javascript وHTML وcss وangular وnodejs، وحقيقة حل هذه المشكلة بسيط وقد قمنا بتجهيز الحل لهذه المشكلة عند بنائنا للمصفوفة favoritesLang حين قمنا بإنشاء المصفوفة selectedFavorite حيث كل اختيار يختاره المستخدم يتم حفظ أسم الأداة التي اختارها في هذه المصفوفة، لذلك كل الذي سوف نعمله هو استبدال القيم في المصفوفة favoritesLang بالقيم الجديدة في المصفوفة selectedFavorite، ويتم عمل ذلك عن طريق أخذ نسخة من قيم النموذج كاملة وتخزينها في كائن object جديد، وهذا الكائن هو عبارة عن كائن عادي لا يمتلك أي من الدوال والخصائص التي يمتلكها الكائنات التي عملنا لها instance من الكلاسات FormGroup او FormControl أو حتى المصفوفة FormArray ، وكما قلنا سابقاً هذا الكائن نخزن فيه جميع القيم من النموذج بما فيها النماذج الفرعية والمصفوفات الفرعية keys لكل نموذج فرعي أو أداة أو مصفوفة مع القيم، ولذلك كل الذي سوف نقوم به هو حذف القيم في المصفوفة الفرعية favoritesLang في الكائن الجديد وأستبدال قيمها بالقيم الموجودة في المصفوفة selectedFavorites ومن ثم أرسال قيم هذا الكائن الجديد للسيرفر أو كيفما تُريد، وهنا سوف نعرض هذه القيم في console، كالتالي:

```
1.  save() {
2.    if (this.form.invalid) {
3.      this.isInvalidForm = true;
4.      this.touchedAllFormFields();
5.      this.loopThroughControls();
6.    } else {
7.      this.isInvalidForm = false;
8.      const obj = this.form.value;
9.      obj.favoritesLang = [];
10.     this.selectedFavorite.map(value => {
11.       obj.favoritesLang.push(value);
12.     });
13.     console.log(obj);
14.   }
15. }
```

نخزن قيم النموذج في كائن جديد وليكن اسمه obj

نحذف جميع القيم من المصفوفة favoritesLang الموجودة في الكائن obj

نأخذ نسخة من القيم الموجودة في المصفوفة selectedFavorites التي تحتوي على أسماء الأدوات المختارة عن طريق الدالة map ومن ثم نعمل لها إضافة عن طريق الدالة push في المصفوفة favoritesLang

نعرض قيمة الكائن الجديد في console

الآن لنرى النتيجة، في console:

وبذلك أصبح لدينا كائن جاهز يحتوي على قيم النموذج ويمكن إنشاء intrerface بنفس ترتيب هذا الكائن والعناصر التي يحتويها وعن طريق service وباستخدام HttpClient نستطيع أرسال هذه البيانات لسيرفر.

المراجع

References

References:

1. Freeman, **Adam, Pro Angular 9**, Apress, 2020.

2. Agarwal, Uttam, **Hands-On Full Stack Development with Angular 5 and Firebase**, Packt, 2018.

3. Delaney, Jeff, **The Angular Firebase Survival Guide**, leanpub, 2018.

4. Saha, Depasis, **Angular 7.0 for Beginners**, 2018.

5. Vijay, Santhosh, **Material for Angular Developer Course**, Leanpub, 2019.

6. Hussain, Asim, **Angular From Theory to Practice**, 2017.

7. D. Booth, Joseph, **Angular Succinctly**, Syncfusion, 2019.

8. Squad, Ninja, **Become a ninja with Angular**, 2019.

9. Steyer, Manfred, **Enterprise Angular**, Leanpub, 2020.

10. Clow, Mark, **Angular 5 Projects**, Apress 2018.

11. Nate Murray, Felipe Coury, Ari Lerner, and Carlos Taborda, **ng-book The Complete Guide to Angular**, Fullstack.io, 2018.

12. Bouchefra, Ahmed, **Practical Angular: Build your first web apps with Angular 8**, Leanpub, 2019.

13. Hajian, Majid, **Progressive Web Apps with Angular**, Apress, 2019.

14. Savkin, Victor, **Angular Router**, Leanpub, 2018.