

Exercies Lesson 4

Create application and add modules

1. Create a new application with ng new(docs)

```
ng new lesson04-forms --routing --style scss
```

Add two modules with ng g m(docs) for implementing a 2. template-driven form and a reactive form
Hint: You could give them a telling name like ReactiveModule and TemplateModule

```
ng g m ReactiveModule --routing
```

```
ng g m TemplateModule --routing
```

Setup template-driven form

1. Add a new component with ng g c(docs). This is here we will add a form. Make sure that the component is added to the right module. Hint: Check the documentation to do this is as a sweet one-liner

IN THE template-module dir:

```
juliezepernickjepsen@MacBook-Pro-tilhrende-Julie template-module % ng generate component TemplateForm --module=template-module
```

```
juliezepernickjepsen@MacBook-Pro-tilhrende-Julie template-module % ng generate component TemplateForm --module=template-module
CREATE src/app/template-module/template-form/template-form.component.scss (0 bytes)
CREATE src/app/template-module/template-form/template-form.component.html (28 bytes)
CREATE src/app/template-module/template-form/template-form.component.spec.ts (602 bytes)
CREATE src/app/template-module/template-form/template-form.component.ts (230 bytes)
UPDATE src/app/template-module/template-module.module.ts (423 bytes)
juliezepernickjepsen@MacBook-Pro-tilhrende-Julie template-module %
```

3. Import FormsModule in the template module.

`template-module.module.ts`

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms'; //this line added

import { TemplateModuleRoutingModule } from './template-module-
routing.module';
import { TemplateFormComponent } from './template-form/template-
form.component';

@NgModule({
```

```

    declarations: [
      TemplateFormComponent
    ],
    imports: [
      CommonModule,
      TemplateModuleRoutingModule,
      FormsModule //and this line added
    ]
  })
export class TemplateModuleModule { }

```

Setup reactive form

1. Add a new component with ng g c(docs). Make sure that the component is added to the right module.

IN THE reactive-module dir:

`ng generate component ReactiveForm --module=reactive-module`

```

● juliezepernickjepsen@MacBook-Pro-tilhrende-Julie reactive-module % ng generate component ReactiveForm --module=reactive-module
CREATE src/app/reactive-module/reactive-form/reactive-form.component.scss (0 bytes)
CREATE src/app/reactive-module/reactive-form/reactive-form.component.html (28 bytes)
CREATE src/app/reactive-module/reactive-form/reactive-form.component.spec.ts (602 bytes)
CREATE src/app/reactive-module/reactive-form/reactive-form.component.ts (230 bytes)
UPDATE src/app/reactive-module/reactive-module.module.ts (423 bytes)
○ juliezepernickjepsen@MacBook-Pro-tilhrende-Julie reactive-module %

```

3. Import ReactiveFormsModule in the template module.

`tempalte-module.module.ts

```

import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms'; //this line added

import { ReactiveModuleRoutingModule } from './reactive-module-
routing.module';
import { ReactiveFormComponent } from './reactive-form/reactive-
form.component';

@NgModule({
  declarations: [
    ReactiveFormComponent
  ],
  imports: [
    CommonModule,
    ReactiveModuleRoutingModule,

```

```

FormsModule // and this line added
]
}))
export class ReactiveModuleModule { }

```

Setup service and mock data

1. Create a file called class.type.ts and copy/paste the following content:
2. Create a file called race.type.ts and copy/paste the following content

The screenshot shows a VS Code editor with two files open. The left file is `class.type.ts` and the right file is `race.type.ts`.

class.type.ts content:

```

src > app > TS class.type.ts > ...
1 export interface Class {
2   name: string;
3   roles: Array<Role>;
4 }
5
6 interface Role {
7   name: string;
8 }
9
10 export const CLASSES = [{
11   name: 'Warrior',
12   roles: [{
13     name: "Tank"
14   }, {
15     name: "Damage"
16   }]
17 }, {
18   name: 'Paladin',
19   roles: [{
20     name: "Tank"
21   }, {
22     name: "Damage"
23   }, {
24     name: "Healer"
25   }]
26 }, {
27   name: 'Hunter',
28   roles: [{
29     name: "Damage"
30   }]
31 }, {
32   name: 'Rogue',
33   roles: [{
34     name: "Damage"
35   }]
36 }, {
37   name: 'Priest',
38   roles: [{
39     name: "Healer"
40   }, {
41     name: "Damage"
42   }]
43 }, {
44   name: 'Shaman',
45   roles: [{
46     name: "Healer"
47   }, {

```

race.type.ts content:

```

src > app > TS race.type.ts > [RACES_HORDE]
1
2 export interface Race {
3   name: string;
4 }
5
6 export const RACES_ALLIANCE = [{
7   name: "Human"
8 }, {
9   name: "Dwarf"
10 }, {
11   name: "Night Elf"
12 }, {
13   name: "Gnome"
14 }, {
15   name: "Draenei"
16 }, {
17   name: "Worgen"
18 }, {
19   name: "Pandaren"
20 }]
21
22 export const RACES_HORDE = [{
23   name: "Orc"
24 }, {
25   name: "Undead"
26 }, {
27   name: "Tauren"
28 }, {
29   name: "Troll"
30 }, {
31   name: "Blood Elf"
32 }, {
33   name: "Goblin"
34 }, {
35   name: "Pandaren"
36 }]

```

3. Create a service named WarcraftService and copy/paste content

in /app

The screenshot shows a terminal window with the following command and output:

```

ng generate service WarcraftService
CREATE src/app/warcraft-service.service.spec.ts (403 bytes)
CREATE src/app/warcraft-service.service.ts (144 bytes)

```

```
TS race.type.ts U    TS warcraft-service.service.ts U X
src > app > TS warcraft-service.service.ts > WarcraftService
1  import { Injectable } from '@angular/core';
2  import { Observable, of } from 'rxjs';
3  import { Class, CLASSES } from './class.type';
4  import { Race, RACES_ALLIANCE, RACES_HORDE } from './race.type';
5
6  @Injectable({
7    providedIn: 'root'
8  })
9  export class WarcraftService {
10
11    constructor() { }
12
13    getClasses(): Observable<Array<Class>> {
14      return of(CLASSES)
15    }
16
17    getAllianceRaces(): Observable<Array<Race>> {
18      return of(RACES_ALLIANCE)
19    }
20
21    getHordeRaces(): Observable<Array<Race>> {
22      return of(RACES_HORDE)
23    }
24  }
```

#Setup the vanilla form

Now that we have some data, we are ready to

1. Create a form with the following input fields:

- name
- password
- confirm_password
- race
- class
- level
- description

```
import { Component, OnInit } from '@angular/core';
import { WarcraftService } from 'src/app/warcraft-service.service';
```

```

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

export class TemplateFormComponent implements OnInit {
  formData: any = {
    name: '',
    password: '',
    confirm_password: '',
    race: '', // Initialize with an empty string
    class: '', // Initialize with an empty string
    level: 0, // Initialize with a default value
    description: ''
  };
  races: any[] = [];
  classes: any[] = [];

  constructor(private warcraftService: WarcraftService) {}

  ngOnInit(): void {
    this.warcraftService.getAllianceRaces().subscribe((races) => {
      this.races = races;
    });

    this.warcraftService.getClasses().subscribe((classes) => {
      this.classes = classes;
    });
  }
}

```

- You will need to think about what type on input fields(docs) to use. Fields race and class must be populated with data from WarcraftService
- Add a `ngsubmit(docs)` to the form
- Before moving on, copy/paste the form into the components we created in Exercise 2 and 3. We will apply a template-driven and a reactive approach later.

```
<p>reactive-form works!</p>
```

```
<form (ngSubmit)="onSubmit(myForm)" #myForm="ngForm">
  <div ngModelGroup="passwordGroup" appPasswordsEqual
#passwordGroup="ngModelGroup">      <!-- Name Input -->
    <div class="form-group">
      <label for="name">Name:</label>
      <input type="text" id="name" name="name" [(ngModel)]="formData.name"
required>
    </div>

    <!-- Password Input -->
    <div class="form-group">
      <label for="password">Password:</label>
      <input type="password" id="password" name="password"
[(ngModel)]="formData.password" required>
    </div>

    <!-- Confirm Password Input -->
    <div class="form-group">
      <label for="confirm_password">Confirm Password:</label>
      <input type="password" id="confirm_password" name="confirm_password"
[(ngModel)]="formData.confirm_password" required>
    </div>
  </div>

  <!-- Race and Class Selection -->
  <div class="form-group">
    <label for="race">Race:</label>
    <select id="race" name="race" [(ngModel)]="formData.race" required>
      <option *ngFor="let race of races" [value]="race.name">{{ race.name
}}</option>
    </select>
  </div>

  <div class="form-group">
    <label for="class">Class:</label>
    <select id="class" name="class" [(ngModel)]="formData.class" required>
      <option *ngFor="let classItem of classes" [value]="classItem.name">
{{ classItem.name }}</option>
    </select>
  </div>
```

```
<!-- Level Input -->
<div class="form-group">
  <label for="level">Level:</label>
  <input type="number" id="level" name="level"
[ (ngModel) ]="formData.level" required>
</div>

<!-- Description Input -->
<div class="form-group">
  <label for="description">Description:</label>
  <textarea id="description" name="description"
[ (ngModel) ]="formData.description"></textarea>
</div>

<!-- Submit Button -->
<div class="form-group">
  <button type="submit" [disabled]="!myForm.valid">Submit</button>
</div>
</form>
```