Exercies Lesson 4

Create application and add modules

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

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

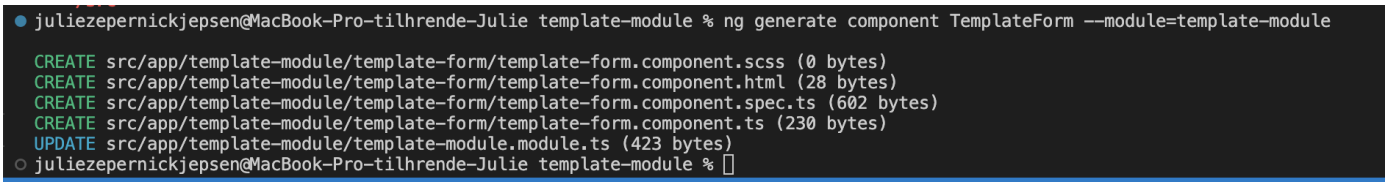
Add two modules with ng g m(docs) for implementing a 2. template-driven form and a reactive form
Hint: You could give them a telling name like ReactiveModule and TemplateModule

`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`



3. Import FormsModule in the template module.
   `template-module.module.ts``

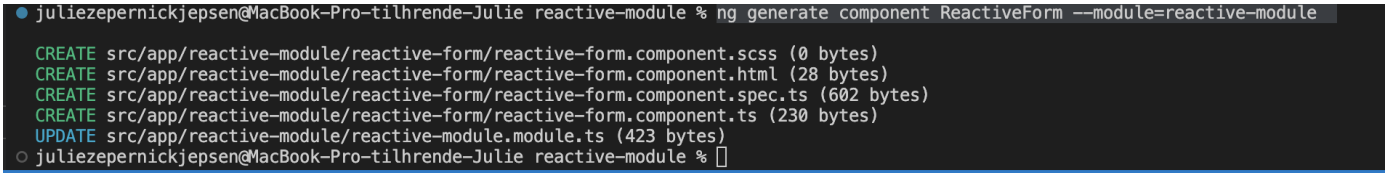
```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms'; //this line added

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``



3. Import ReactiveFormsModule in the template module.

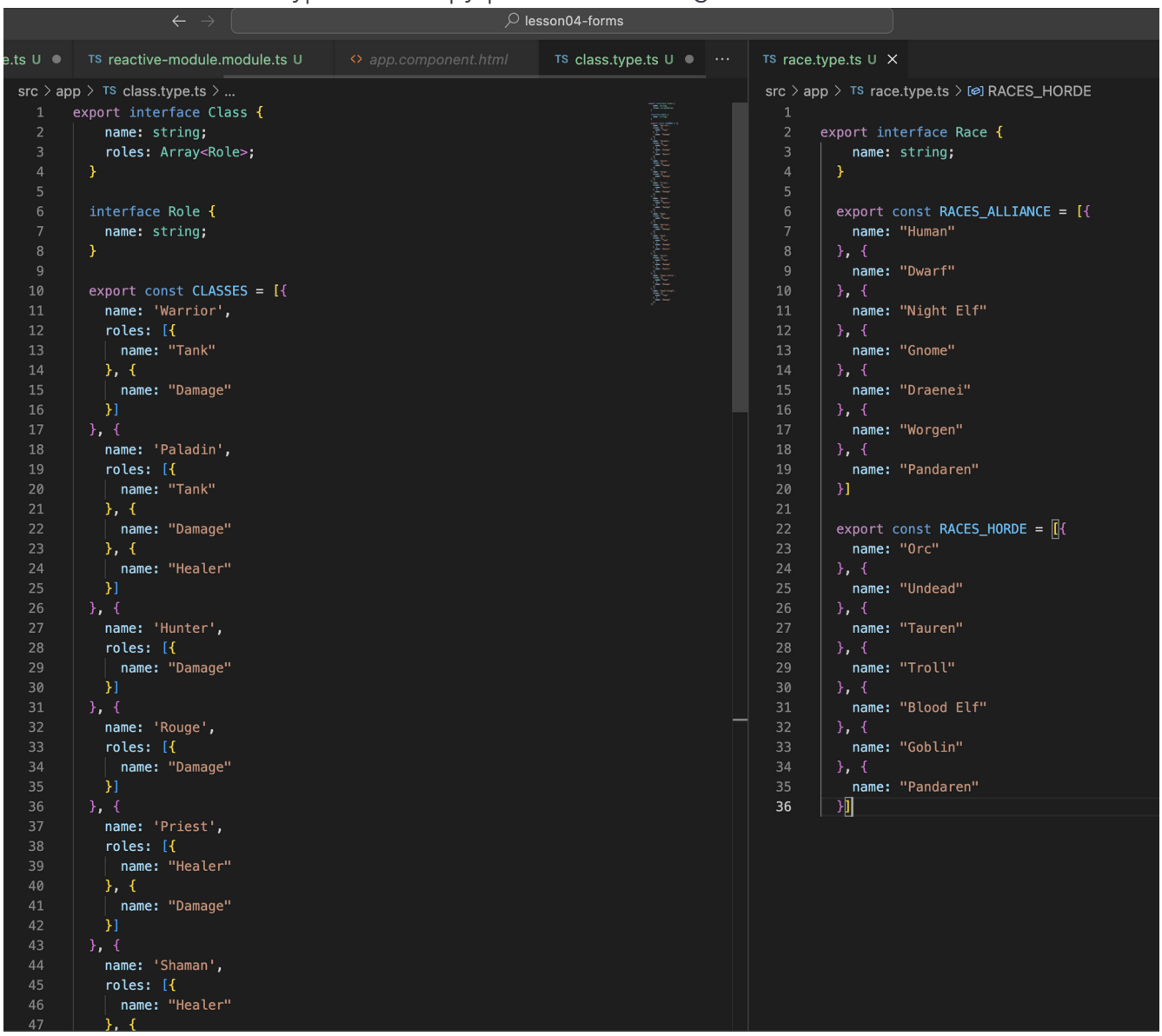
`template-module.module.ts`

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

import { 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

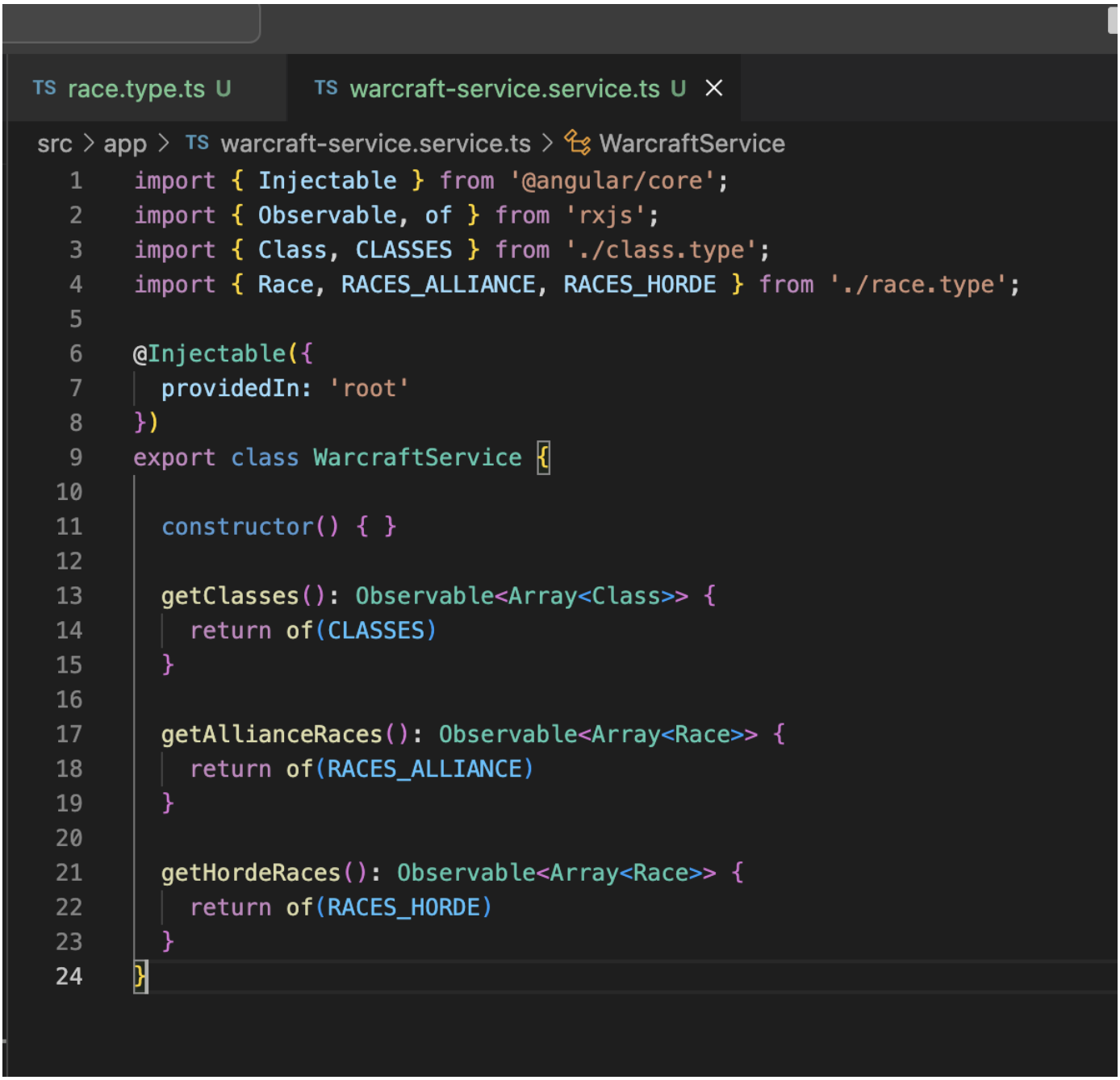


3. Create a service named WarcraftService and copy/paste content

in /app

```
ng generate service WarcraftService
```

```
juliezepernickjepsen@MacBook-Pro-tilhrende-Julie app % ng generate service WarcraftService

  CREATE src/app/warcraft-service.service.spec.ts (403 bytes)
  CREATE src/app/warcraft-service.service.ts (144 bytes)
juliezepernickjepsen@MacBook-Pro-tilhrende-Julie app %
```



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

- 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

```
 form = this.fb.group({
    name: ['', [Validators.required]],
    passwordGroup: this.fb.group({
      password: [''],
      confirm_password: [''],
    }, { validators: [passwordsEqual, Validators.required], updateOn:
'blur'}),
    race: ['', [Validators.required]],
    class: ['', [Validators.required]],
    level: [, [Validators.min(1), Validators.max(60)]],
    description: [''],
  })

  classes$: Observable<Class[]>
  races$: Observable<Race[]>

  constructor(private fb: FormBuilder, private service: WarcraftService) {
    this.classes$ = this.service.getClasses()
    this.races$ = combineLatest([this.service.getAllianceRaces(),
this.service.getHordeRaces()]).pipe(map(c => [...c[0], ...c[1]]))
  }
```

- Add a ngsubmit(docs) to the form

```
 onSubmit() {
    console.log('onSubmit')
  }

  compare(c1: Race | Class, c2: Race | Class) {
    return c1 && c2 ? c1.name === c2.name : c1 === c2;
  }

  get name() { return this.form.get('name') }
  get level() { return this.form.get('level') }
  get passwordGroup() { return this.form.get('passwordGroup') }
  get class() { return this.form.get('race') }
  get race() { return this.form.get('class') }
```

Template-driven approach

Setup the form with the template-driven directive ngModel(docs) and ngModelGroup(docs)

in `template-module.module.html`:

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

imports:[
    FormsModule
]
```

`template-form.component.html`

`ngModelGroup` is used to group together the name, password input and the confirm password.

```
<p>form works!</p>
<form #f="ngForm" (ngSubmit)="onSubmit(f)">
  <input type="text" autocomplete="username email" name="name"
placeholder="Name" required #name="ngModel" ngModel />
  <div ngModelGroup="passwordGroup" appPasswordsEqual
#passwordGroup="ngModelGroup">
    <input type="password" autocomplete="new-password" name="password"
ngModel placeholder="Password" />
    <input type="password" autocomplete="new-password"
name="confirm_password" ngModel placeholder="Confirm password" />
    <div *ngIf="passwordGroup.invalid && (passwordGroup.dirty ||
passwordGroup.touched)">
      <div *ngIf="passwordGroup.errors?.['must_match']">
        {{ passwordGroup.errors?.['must_match'] }}
      </div>
    </div>
  </div>
  <select name="class" required ngModel #race="ngModel" ngModel>
    <option [ngValue]="" disabled selected>Race</option>
    <option *ngFor="let race of races$ | async" [ngValue]="race">{{
race.name }}</option>
  </select>
  <div *ngIf="race?.invalid && (race?.dirty || race?.touched)" class="alert
alert-danger">
    <div *ngIf="race?.errors?.['required']">
      {{race?.errors! | json}}
    </div>
  </div>
  <select name="class" ngModel required #class="ngModel">
    <option [ngValue]="" disabled selected>Class</option>
    <option *ngFor="let class of classes$ | async" [ngValue]="class">{{
class.name }}</option>
  </select>
```

```
  <div *ngIf="class?.invalid && (class?.dirty || class?.touched)"
class="alert alert-danger">
    <div *ngIf="class?.errors?.['required']">
      {{class?.errors! | json}}
    </div>
  </div>
  <input name="level" id="level" #level="ngModel" ngModel required min="1"
max="60" type="number" placeholder="Level"/>
  <div *ngIf="level?.errors?.['min']">
    {{level?.errors! | json}}
  </div>
  <div *ngIf="level?.errors?.['max']">
    {{level?.errors! | json}}
  </div>
  <input name="description" ngModel required type="description"/>
  <input type="submit" value="Submit" />
</form>
```

template-form.component.ts

```
import { Component, OnInit } from '@angular/core';
import { NgForm } from '@angular/forms';
import { Observable, map, combineLatest } from 'rxjs';
import { Class } from 'src/app/class.type';
import { Race } from 'src/app/race.type';
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 {

  classes$: Observable<Class[]>
  races$: Observable<Race[]>

  constructor(private service: WarcraftService) {
    this.classes$ = this.service.getClasses()
    this.races$ = combineLatest([this.service.getAllianceRaces(),
this.service.getHordeRaces()]).pipe(map(c => [...c[0], ...c[1]]))
  }
  ngOnInit(): void {
```

```
  }
  onSubmit(form: NgForm) {
    console.warn('onSubmit()')
    console.log(form.value)
  }
}
```

Validation

Add some validation to the form fields

- name is required
- level must be in the range 1-60
- password and confirm_password is required and must match
- race is required
- class is required

Show the validation error, if any, to the user
Hint: You could create a separate component to display error messages

validation and requirements

```
form = this.fb.group({
    name: ['', [Validators.required]],
    passwordGroup: this.fb.group({
      password: [''],
      confirm_password: [''],
    }, { validators: [passwordsEqual, Validators.required], updateOn:
'blur'}),
    race: ['', [Validators.required]],
    class: ['', [Validators.required]],
    level: [, [Validators.min(1), Validators.max(60)]],
    description: [''],
  })
  ```
```
```