Angular routing i spajanje na REST servise

Angular Router

- Angular Router omogućava navigaciju po *view* komponentama kako bi se implementirala poslovna logika aplikacije
- Modul za routing dodaje se u aplikaciju korištenjem Angular CLI naredbe: **ng generate module app-routing --flat --module=app**
- Opcija „-flat” dodaje nove datoteke u mapu „src/app”, umjesto vlastite mape
- Opcija „- -module=app” označava dodavanje „import” polja u „AppModule”

Primjer Routing Module implementacije

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { StudentsComponent } from './students/students.component';
import { StudentDetailComponent } from './student-detail/student-detail.component';

const routes: Routes = [
  { path: 'students', component: StudentsComponent },
  { path: 'detail/:jmbag', component: StudentDetailComponent }
];
```

Primjer Routing Module implementacije

```
@NgModule({
  declarations: [],
  imports: [
    RouterModule.forRoot(routes)
  ],
  exports: [
    RouterModule
  ]
})
export class AppRoutingModule { }
```

Primjer Routing Module implementacije

- AppRoutingModule importira RouterModule i Routes kako bi se osiguralo ispravno funkcioniranje navigacije
- Komponente „StudentsComponent” i „StudentDetailComponent” definiraju stranice na koje se preusmjerava tok ekrana
- „routes” sadržava podatak na kojem se konfiguriraju rute i definiraju *view* koji je potrebno prikazati u slučaju kad korisnik aplikacije odabere neki od linkova na ekranu:

```
const routes: Routes = [
  { path: 'students', component: StudentsComponent },
  { path: 'detail/:jmbag', component: StudentDetailComponent }
];
```

Angular Route

- Uobičajena Angular Route komponenta najčešće sadrži dva parametra:
  - Path: String koji predstavlja URL stranice
  - Component: komponenta koju Router kreira u slučaju navigacije na tu rutu
- Dekorator @NgModule inicijalizira Router i prima događaje vezane uz promjenu URL-a u pregledniku
- Pomoću sljedeće linije dodaju se rute koje se koriste kod navigacije počevši od „root” razine:

```
imports: [
    RouterModule.forRoot(routes)
  ],
```

Angular Route

- Pomoću sljedeće naredbe omogućava se pristup rutama po cijeloj aplikaciji:

```
exports: [
    RouterModule
  ]
```

- Element „router-outlet” predstavlja direktivu koja se koristi za prikaz komponente koja prikazuje određenu komponentu:

```
<h1>{{title}}</h1>
<router-outlet></router-outlet>
```

Dodavanje navigacijskih linkova

- Linkove za navigaciju moguće je dodati korištenjem „routerLink” atribute unutar „nav” elementa:

```
<h1>{{title}}</h1>
<nav>
  <a routerLink="/students">Students</a>
</nav>
<router-outlet></router-outlet>
```

Spajanje na REST servise

- Za spajanje Angular aplikacije na REST service potrebno je definirati osnovni URL gdje su REST endpointovi, omogućiti izmjenu JSON formata podataka i korištenje HttpClient modula:

```
export class StudentService {

  private studentsUrl = 'http://localhost:8080/student';

  httpOptions = {
    headers: new HttpHeaders({ 'Content-Type': 'application/json' })
  };

  constructor(
    private http: HttpClient
  ) { }
```

Pozivanje REST metoda

- Primjer servisne metode koja dohvaća više zapisa može izgledati ovako:

```
getStudents(): Observable<Student[]> {
    return this.http.get<Student[]>(this.studentsUrl)
      .pipe(
        tap(_ => console.log('fetched students')),
        catchError(this.handleError<Student[]>('getStudents', []))
      );
  }
```

Pozivanje REST metoda

- Primjer servisne metode koja sprema podatke:

```
addStudent(student: Student): Observable<Student> {
    return this.http.post<Student>(this.studentsUrl, student, this.httpOptions).pipe(
      tap((newStudent: Student) => console.log(`added student w/ JMBAG=${newStudent.jmbag}`)),
      catchError(this.handleError<Student>('addStudent'))
    );
  }
```

Pozivanje REST metoda

- Primjer servisne metode koja ažurira podatke:

```
updateStudent(student: Student): Observable<any> {
    const url = `${this.studentsUrl}/${student.jmbag}`;
    return this.http.put(url, student, this.httpOptions).pipe(
      tap(_ => console.log(`updated student jmbag=${student.jmbag}`)),
      catchError(this.handleError<any>('updateStudent'))
    );
  }
```

Pozivanje REST metoda

- Primjer servisne metode koja briše podatke:

```
deleteStudent(student: Student | string): Observable<Student> {
    const jmbag = typeof student === 'string' ? student : student.jmbag;
    const url = `${this.studentsUrl}/${jmbag}`;

    return this.http.delete<Student>(url, this.httpOptions).pipe(
      tap(_ => console.log(`deleted student JMBAG=${jmbag}`)),
      catchError(this.handleError<Student>('deleteStudent'))
    );
  }
```

Obrada pogrešaka

- U slučaju pogreške je moguće pozivanje sljedeće metode:

```
private handleError<T>(operation = 'operation', result?: T) {
    return (error: any): Observable<T> => {
      console.error(operation);
      console.error(error);
      return of(result as T);
    };
  }
```

Angular funkcije „pipe” i „tap”

- Funkcija „pipe” prima podatke i transformira ih u željeni izlazni format
- Funkcija „tap” obavlja „foreach” funkcionalnost
- Za dodavanje reaktivnih funkcija potrebno je dodati sljedeći „import”:

```
import { Observable, of } from 'rxjs';
import { catchError, tap } from 'rxjs/operators';
```

Dodavanje ekrana za spremanje podataka

- Za dodavanje novih zapisa potrebno je kreirati ekran koji će omogućavati unos podataka:

```
<div>
  <h3>Unesi novog studenta</h3>
  <div>
    <label>Ime:
      <div>
        <input #studentFirstName />
      </div>
    </label>
  </div>
  <div>
    <label>Prezime:
      <div>
        <input #studentLastName />
      </div>
    </label>
  </div>
```

Dodavanje ekrana za spremanje podataka

```
<div>
    <label>Broj ECTS bodova:
      <div>
        <input type="number" #studentNumberOfECTS />
      </div>
    </label>
  </div>

  <button (click)="add(studentFirstName.value, studentLastName.value, studentJMBAG.value,
+studentNumberOfECTS.value); studentFirstName.value=''; studentLastName.value=''; studentJ
MBAG.value=''; studentNumberOfECTS.value=''">
    Unesi studenta
  </button>
</div>
```

Dodavanje ekrana za spremanje podataka

- Metoda za dodavanje novih zapisa o studentima u komponent StudentComponent može izgledati ovako:

```
add(firstName: string, lastName: string, jmbag: string, numberOfECTS: number): void {
    firstName = firstName.trim();
    lastName = lastName.trim();
    jmbag = jmbag.trim();
    if (!firstName || ! lastName || !jmbag || !numberOfECTS) { return; }

    const dateOfBirth = new Date(1990);

    this.studentService.addStudent({ firstName, lastName, jmbag, numberOfECTS, dateOfBirth } as Student)
      .subscribe(student => {
        this.students.push(student);
      });
  }
```

CORS (engl. Cross-Origin Resource Sharing)

- Mehanizam koji koristi dodatna HTTP zaglavlja koja omogućavaju pristupanje web aplikacije s istog izvora
- Moguće ga je konfigurirati korištenjem anotacije „@CrossOrigin” na sljedeći način:

```
@RestController
@RequestMapping("student")
@CrossOrigin(origins = "http://localhost:4200")
public class StudentController {
```

Pitanja?