Lecture 07

Setting up environment

1. Open the *CourseAdminSystemBackend* in Visual Studio Code
2. Open the *CourseAdminSystemAngular* in Visual Studio Code
3. Make sure the Postgres database is running.
4. Run the above projects to make sure everything works fine.

Add new component EditStudent

1. Create a new component to Edit student under */src/app* using the following command

```
ng generate compponent EditStudent
```

2. Edit the *student.component.html* to add an Edit button as follows

```
<button (click)="editStudent(student.id)">Edit</button>
```

3. Edit the *student.component.ts* and add a new function to Edit a student as follows.

```
...
editStudent(id: number) {

}
...
```

4. Edit *student.component.ts* and update the constructor as follows.

```
import { Router } from '@angular/router';

...
constructor(private studentService: StudentService, private router: Router) {}
...
```

5. Update the editStudent method as follows.

```
editStudent(id: number) {
    this.router.navigate(["edit-student", id]);
}
```

6. Edit the *app.routes.ts* file to add a new route to the EditStudent component as follows.

```
export const routes: Routes = [
   { path: "students", component: StudentListComponent },
   { path: "edit-student/:id", component: EditStudentComponent },
   { path: "", component: StudentListComponent }
];
```

7. Click on the Edit button and verify that the view gets navigated to the new EditComponent.

Configure the EditStudent

Now we need to access the id of the student being passed as a url parameter in the router config. In order to do so, we need to do the following.

1. Edit *app.config.ts* and modify it as follows.

```
...
import { provideRouter, withComponentInputBinding } from '@angular/router';
...

export const appConfig: ApplicationConfig = {
  providers: [
    provideRouter(routes, withComponentInputBinding()),
    provideHttpClient()
  ]
};
```

2. In *edit-student.component.ts* add a variable *id* as follows .

```
...
@Input() id!: string;
...
```

3. Edit *edit-student.component.html* as follows to make sure that the correct **id** is being passed to this component when clicking the edit button.

```
<p>edit-student {{id}} works!</p>
```

4. Edit the *ngOnInit* method to load the student information for the provided **id** as follows.

```
import { Component, Input, OnInit } from '@angular/core';
import { StudentService } from '../service/student.service';
import { Student } from '../model/student';

export class EditStudentComponent implements OnInit {
...
  @Input() id!: number;
  student!: Student;

  constructor(private studentService: StudentService) {

  }

  ngOnInit() {
    this.studentService.getStudent(this.id).subscribe(student => {
      this.student = student;
```

```
    });
  }
...
}
```

4. Edit the edit-student.html as follows.

```
<div>
   id: <span>{{student.id}}</span>
</div>
<div>
   First name: <span>{{student.firstName}}</span>
</div>
<div>
   Last name: <span>{{student.lastName}}</span>
</div>
<div>
   Study Program: <span>{{student.studyProgram}}</span>
</div>
<div>
   DOB: <span>{{student.dob}}</span>
</div>
```

5. Verify that the correct student's information is being shown when you click on the Edit button.

Extend web service to add additional methods

1. Edit *student.service.ts* and add the following two method

```
...
getStudent(id: number): Observable<Student> {
   return this.httpClient.get<Student>(`${this.baseUrl}/student/${id}`);
}

updateStudent(student: Student): Observable<any> {
   return this.httpClient.put(`${this.baseUrl}/student`, student);
}
...
```

Configure 2-way binding to update student information.

1. Done by enabling **ngModel**.
2. Edit the *edit-student.component.ts* file and add an Import to FormsModule as follows.

```
...
```

```
import { FormsModule } from '@angular/forms';
...
@Component({
  selector: 'app-edit-student',
  standalone: true,
  imports: [FormsModule],
  templateUrl: './edit-student.component.html',
  styleUrl: './edit-student.component.css'
})
```

3. Replace the contents of edit-student.component.html as follows. (Important to make sure the data property names match the ones received from the API)

```
<div>
   id: <span>{{this.id}}</span>
</div>
<div>
   First name: <input type="text" [(ngModel)]="student.firstName">
</div>
<div>
   Last name: <input type="text" [(ngModel)]="student.lastName">
</div>
<div>
   Study Program: <input type="text" [(ngModel)]="student.studyProgramId">
</div>
<div>
   DOB: <input type="text" [(ngModel)]="student.dob">
</div>
<button (click)="updateStudent()">Update</button>
```

4. In *edit-student.component.ts* add the method *updateStudent* as follows.

```
updateStudent() {
   this.studentService.updateStudent(this.student!).subscribe();
}
```

5. Modify the values and click on update. Verify that the data has been updated.

Automatically Refresh list on update.

1. Edit the *edit-student.component.ts* file as follows.

```
...
import { Router } from '@angular/router';
...
```

```
constructor(private studentService: StudentService, private router: Router) {
}

updateStudent() {
   this.studentService.updateStudent(this.student!).subscribe(() => {
      this.router.navigate(["student"]);
   });
}
```

Basic validation using FormsModule

1. In *edit-student.component.html* add a **required** validator to First name and last name properties as shown below.

```
...
First name: <input type="text" [(ngModel)]="student.firstName" required
#firstName="ngModel">


...
Last name: <input type="text" [(ngModel)]="student.lastName" required #lastName="ngModel">
```

2. Add **minlength, maxlength, required** and **pattern** validators to Study Program property as follows

```
...
Study Program: <input type="text" [(ngModel)]="student.studyProgramId" required
pattern="\d+" minlength="1" maxlength="2" #studyProgramId="ngModel">
```

3. Display appropriate error messages to the properties when a validation check fails, as shown below

```
...
<div>
   First name: <input type="text" [(ngModel)]="student.firstName" required
#firstName="ngModel">
   @if (firstName.invalid) {
      <span>First name must be provided</span>
   }
</div>
<div>
   Last name: <input type="text" [(ngModel)]="student.lastName" required
#lastName="ngModel">
   @if (lastName.invalid) {
      <span>Last name must be provided</span>
   }
</div>
```

```
<div>
   Study Program: <input type="text" [(ngModel)]="student.studyProgramId" required
pattern="\d+" minlength="1" maxlength="2" #studyProgramId="ngModel">
   @if (studyProgramId.errors?.['required']) {
      <div>Study program is required</div>
   }
   @if (studyProgramId.errors?.['pattern']) {
      <div>Only digits</div>
   }
</div>
...
```

4. Display a general error if any of the fields has an invalid data and show the *Update* button ONLY when all fields have valid values as shown below.

```
...
@if (firstName.invalid || lastName.invalid || dob.invalid || studyProgramId.invalid) {
   <div>Error in data</div>
}
@else {
   <button (click)="updateStudent()">Update</button>
}
...
```

Exercise

1. Create similar a similar component to edit Teacher.
    1. Add new component.
    2. Apply two-way binding
    3. Display Teacher information in edit mode.
    4. Update Teacher information to database.
    5. Add basic validations to Teacher properties