Lecture 06

Setup the database

1. Check Postgres database is running.
2. Check PgAdmin4

Setup the API Project to run on your computers

1. Make sure the project from Lecture04 is running on your computers.
2. If you do NOT have your own working copy, then download the API project ***Lecture04-ApiDb.zip*** from Canvas (under Module name Lec 04 —> Supporting Files) and use that.
    1. Unzip it and open the project in Visual Studio Code. **Please note** that the API project requires that your databases are setup on your computers as described in Lecture 04.
3. Under the project *CourseAdminSystem.API*, open the *appsettings.json* file to make sure that the *ConnectionStrings —> AppProgDb* is correctly configured.
4. Run the project *CourseAdminSystem.API* by right clicking on it and select *Debug —> Start New Instance*.
5. Alternatively, you can open the terminal and run the following command to start the application, and afterwards browse to http://localhost:5016/swagger/index.html

```
dotnet run --project CourseAdminSystem.API
```

5. Try the various endpoints using Swagger UI.
6. Keep the application running in the background.

Setup the Angular Web project to run on your computers

1. Make sure the project from Lecture05 is running on your computer.
2. If you do NOT have your own working copy, then download the Angular project ***CourseAdminSystemAngular.zip*** from Canvas and use that.
    1. Copy it to a suitable path and unzip the file.
3. Open Visual Studio Code and open the folder *CourseAdminSystemAngular.*
4. Make sure that the project is running by opening the terminal and running the following commands.

```
# To install all the dependencies
npm install

# To build and run the ngular project
ng serve
```

Connect the API with our Angular Web application

1. Create a new Service

1. Create a new service called StudentService using the following command.

```
# The following command creates the service under a folder 'services'
ng generate service services/Student
```

2. Open the newly created *student.service.ts* class.
3. Make the following changes to the newly created *StudentService* class.
    1. Add a new property *baseUrl* setting its value to the url of our api. **NOTE**: Make sure that the *port numbe*r matches the one on your machine.
    2. Add an *HttpClient* property to the constructor. An import line will be added automatically.
4. The code should look like the following (changes highlighted in bold).

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class StudentService {
  baseUrl: string = "http://localhost:5057/api";
  constructor(private http: HttpClient) { }
}
```

5. Add the methods *GetStudents*, *GetStudent*, *CreateStudent* and *DeleteStudent* to the *StudentService* class so that the code looks like the following.

```
...
...
getStudents(): Observable<Student[]> {
  return this.http.get<Student[]>(`${this.baseUrl}/student`);
}

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

createStudent(student: Student): Observable<any> {
  return this.http.post(`${this.baseUrl}/student`, student);
}

deleteStudent(id: number): Observable<any> {
  return this.http.delete(`${this.baseUrl}/student/${id}`);
}
...
...
```

6. Open the *app.config.ts* file and make the following (highlighted) change.

```
...
...
```

```
import { provideHttpClient } from '@angular/common/http';

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

2. Update the StudentList component to use StudentService

1. Open the *student-list.component.ts* file and add a constructor which takes a *StudentService* property so that we can use the service from within this component.

```
...
import { StudentService } from '../service/student.service';


...

export class StudentListComponent {
  constructor(private studentService: StudentService) {}
  ...
}
```

2. Update the StudentListComponent so that it implements *OnInit* interface as follows

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

export class StudentListComponent implements OnInit {
  constructor(private studentService: StudentService) {}
  ngOnInit(): void {
    throw new Error('Method not implemented.');
  }
...
...
```

3. Update the *ngOnInit* method to call the *StudentService*'s GetStudents methods to retrieve a list of student from the API, instead of the hard codes one. The code should like the following

```
...
...
student: Student[] = [] // Initialized with an empty array

// Whenever the component initializes, it'll load student from API
ngOnInit(): void {
  this.studentService.getStudents().subscribe(students => {
    this.students = students;
```

```
  });
}
```

4. **NOTE:** If there is no connection to the api, then make sure to add the following line of code to your API project in *Program.cs* file.

```
app.UseCors(policy => policy.AllowAnyHeader().AllowAnyMethod().AllowAnyOrigin());
```

5. Add new *Students* from the Swagger UI and refresh the Web application in the browser to make sure that updated records are retrieved.
6. Delete a *Student* from the Swagger UI and refresh the Web application in the browser to make sure that deleted records are removed.
7. Can you figure out why the *DOB* is not being displayed? How to fix it?

3. Update the Student component to use StudentService to implement delete functionality

1. Open the *student.component.html*.
2. Add a button to delete a particular student

```
...
...
<div><button (click)="deleteStudent()">Delete</button></div>
```

3. Open the *student.component.ts*.
4. Add a constructor which takes a *StudentService* property as follows.

```
...
...
constructor(private studentService: StudentService) {}
...
...
```

5. Add the *deleteStudent()* function which we declared in the html file, as follows

```
deleteStudent(): void {
  this.studentService.deleteStudent(this.student.id).subscribe();
}
```

6. Test the functionality by deleting one of the students and refreshing the page. The student should now be deleted.

Exercise

1. Extend the API project to add a TeacherController similar to StudentController.
2. Add a new *TeacherService* similar to the *StudentService*.
3. Update the *TeacherList* component so it fetches data from the API.
4. Update the *Teacher* component to add a functionality to delete a *Teacher.*