FULL STACK DEV

Why use Angular for frontend dev. ?

Presented by:

Rajeev Khoodeeram

OCTOBER 2025

KEY CHARACTERISTICS (1)

- Component-Based Architecture
  - Applications are built as a tree of self-contained, reusable components.

- TypeScript
  - Angular is built with TypeScript, that adds static typing, improving code quality and maintainability.

- Data Binding
  - Seamless synchronization of data between the model and the view.

KEY CHARACTERISTICS (2)

- Dependency Injection
  - A robust system for providing dependencies to components and services - *just like in Java !*

- Routing
  - A powerful module for navigating between different views/pages without full page reloads.
- CLI
  - A robust tool for scaffolding projects, generating code, running tests, and deploying applications.
- RESTful APIs
  - Angular is designed to easily consume RESTful APIs, which is how your Java backend will expose data

INSTALLING NODE AND ANGULAR 17

- Prerequisites for Angular Development

- Before we start, ensure you have these installed:
  - Node.js
  - npm (Node Package Manager)

- You can check on your terminal :
  - >>node -v
  - >>npm -v

WHAT IS NODE.JS?

- Node.js is a runtime environment that lets you run JavaScript outside the browser.

- Normally, JavaScript runs only in browsers (like Chrome or Firefox). But Node.js, built on Google’s V8 JavaScript engine, allows JavaScript to run as a general-purpose programming language.

- It comes with tools and libraries that let you:
  - Build web servers & APIs
  - Use package management (via npm, Node Package Manager)
  - Run build tools (like webpack, Babel, TypeScript compilers)

ANGULAR CLI

- This is your primary tool for Angular development.
- After Node.js and npm are installed, install the Angular CLI globally using npm:
  - >>npm install -g @angular/cli
  - or
  - >>npm install -g @angular/cli@17. (To specify the version number; here 17)
- >>ng version
- This will display your Angular CLI version and other relevant details.
  - Angular CLI: 17.3.17
  - Node: 24.4.1 (Unsupported)
  - Package Manager: npm 11.5.2
  - OS: darwin arm64

FULL STACK DEV

Creating your Angular frontend project

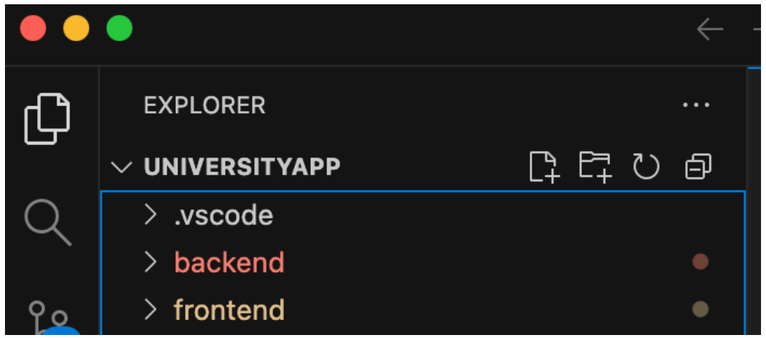
Presented by:

Rajeev Khoodeeram

OCTOBER 2025

WHERE TO CREATE THE ANGULAR FRONTEND ?

- Navigate to the directory where you want to create your Angular project.
- This is usually outside your Java backend project folder, as they are separate applications that communicate via HTTP.
-

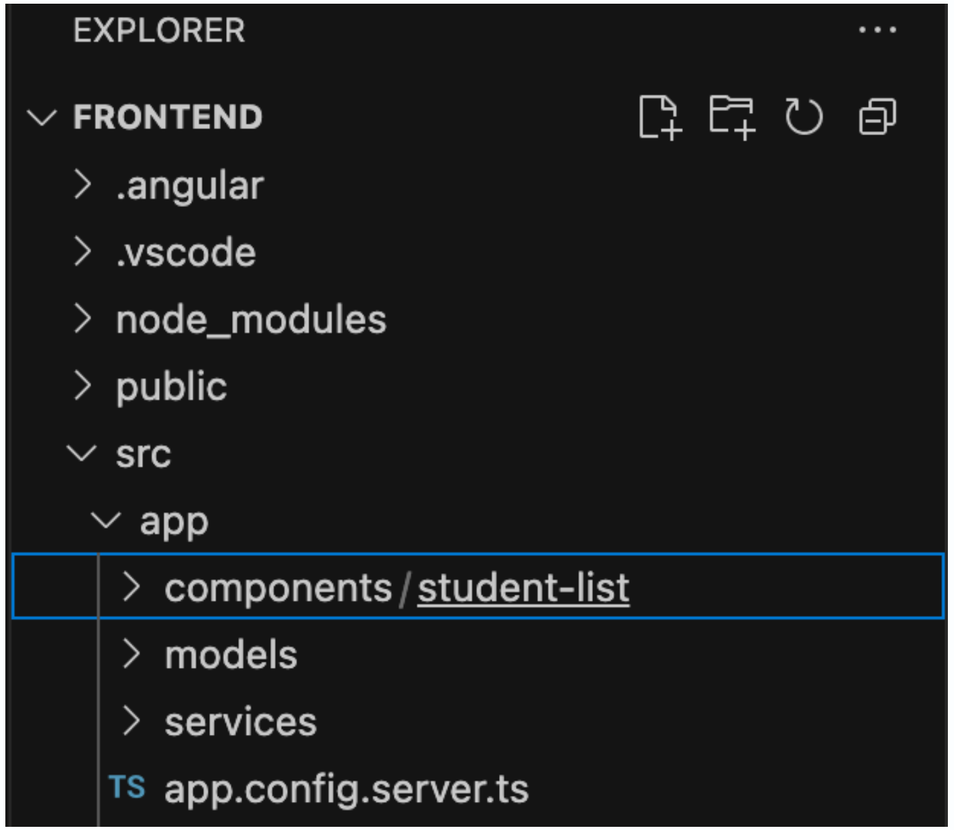


COMMAND LINE

- >> ng new **frontend** —no-standalone

- Or (best)

- ng new **frontend** \
- --standalone \
- --routing \
- --style=scss \
- --strict \
- --package-manager=pnpm



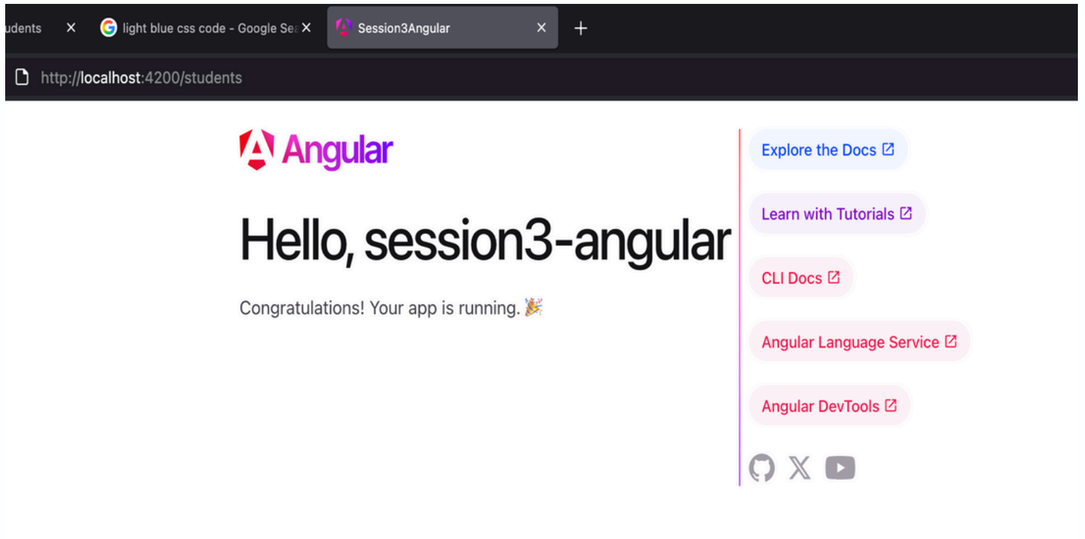
TYPES OF ANGULAR APP

- Zoneless Angular
  - Depend on Zone.js, a library that monkey-patches async APIs (like setTimeout, DOM events, promises) to detect changes and automatically trigger change detection.
  - Zone.js adds runtime overhead.
  - It patches many browser APIs, which can cause performance bottlenecks and debugging headaches.

- Zoneless Angular
  - Instead Angular uses modern browser APIs like Signals or RxJS to know when to update the UI.

HTTP SPECIFIC : GETMAPPING

- With SSR:
  - User requests /home.
  - Angular runs on the server (Node.js) and generates HTML for /home.
  - Browser gets ready-to-render HTML instantly.
  - Angular JavaScript loads and "hydrates" (adds interactivity).
-



RUNNING YOUR ANGULAR APP

- Once the folder is created, open it in VS Code.
- >> npm install (in VS Code inside the folder)

- Let us check if our angular front end is working by opening a terminal inside VS code :
  - >> ng serve

- Normally, it will be using : http://localhost:4200.

- Add this line in RestController to allow access to backend
  - @CrossOrigin(origins = "http://localhost:4200")

FRONTEND STRUCTURE (1)

- node_modules/: Contains all the third-party libraries
- src/: Your main application source code.

- app/: Contains your application's components, modules, services, etc.
- app.ts: The root component of your application.

- app.html: The HTML template for the root component (to be replaced - should only contain <router-outlet />)
- app.scss : css for the App component

- app.config.ts :for application configuration (ex api URL — see below), it is called in app.config.server.ts

FRONTEND STRUCTURE (2)

- app.spec.ts : used for testing (in jasmine for ex ; just like we have JUnit)
- app.module.ts: The root module that defines how your application's parts fit together.
- app-route.ts: Defines your application's routes
- index.html: The single entry point of your Angular SPA.
- main.ts: The entry point for your TypeScript application
- styles.scss: Global styles for your application.

APP.CONFIG.TS

- Used to initialise constants :

- For example
- export const appConfigServer = {
- apiUrl: 'http://localhost:8080/api/students'
- };

- export class StudentService {
- // harcoding
- // private apiUrl = 'http://localhost:8080/api/students';
-
- **private apiUrl = appConfigServer.apiUrl;**

APP.ROUTE.TS

- Defines your application's routes (that is how you navigate through your website / page)

```
export const routes: Routes = [
  { path: 'students', component: StudentList },
  { path: '', component: App }, // default route
];
```

FULL STACK DEV

Angular concepts for creating the frontend pipeline

Presented by:

Rajeev Khoodeeram

OCTOBER 2025

JAVA VS ANGULAR

- Now that your Angular app is running, let's look at how it will interact with your Java backend.
- In **Java** we have this process :

- Controller —> Service —> Repository —> Database
  Model

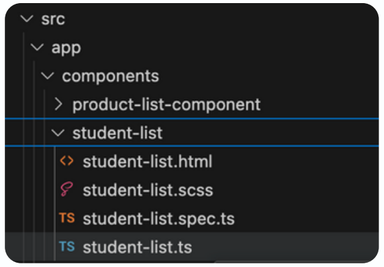
- In **Angular**, we have a similar kind of process :

- Component —> Service —> BACKEND
  Model

COMPONENTS

- Components are the building blocks of your UI. Each component has an HTML template, a TypeScript class for logic, and optional CSS styles.

- Generating a component:
  - >>ng generate component components/student-list-component
- # or the shorthand:
  - >>ng g c components/student-list-component



- This will create the components folder and the files for student-list-component (*the class is called StudentListComponent*)

SERVICES

- Services are designed to provide reusable functionality, often for data retrieval or business logic (*just like in Java !!*)
- You'll create services to make HTTP requests to your Java backend's REST APIs.
- Generating a service:
  - >>ng generate service services/student-service
- # or the shorthand:
  - >>ng g s services/student-service

- This will create the services folder and the files for student.service (two files created : one StudentService class - the main file stored in student-service.ts; and a test file called student-service.spec.ts).

MODELS

- Angular CLI does not have a built-in generator for models, because models are just TypeScript interfaces/classes.

- But you can create them manually inside a models folder:

- >>ng g class models/student --type=model

- src/app/models/student.model.ts. will be created (modify according to our Java entity)

STUDENT.MODEL.TS

```
export class Student {
    studentId: number;
    studentFirstName: string;
    studentLastName:       string;
    //enter other attributes here
    studentEnrollmentDate: Date;
    studentStatus:         string;

    constructor(
        studentId: number,
        firstName: string,
        lastName: string,
            // add other attributes here
        enrollmentDate: Date,
        status: string
    ) {
        this.studentId = studentId;
        this.studentFirstName = firstName;
        this.studentLastName = lastName;
            // add other attributes here
        this.studentEnrollmentDate = enrollmentDate;
        this.studentStatus = status;
    }

}
```

HTTPCLIENT MODULE / MAKING HTTP REQUESTS

- To make HTTP requests, you need to import HttpClient into your student-service.ts.
    - import { HttpClient } from '@angular/common/http';
    - import { Observable } from 'rxjs';
- In Angular, the service part interacts with the Spring back end
- We are going to write the business logic to connect to backend and retrieve the list of students
- Example src/app/services/student.service.ts:

CREATING THE SERVICE

- export class StudentService {
- private students: Student[] = [];
- private apiUrl = appConfigServer.apiUrl;
-
- constructor(private http: HttpClient) { }

- /**
- * Fetches the list of students from the API.
- * @returns An observable containing the list of students.
- */
- getStudents(): Observable<Student[]> {
- return this.http.get<Student[]>(this.apiUrl);
- }

RXJS - OBSERVABLE PATTERN

- It is a core concept in reactive programming which focuses on data streams and the propagation of change.

- In the context of Angular, it's primarily implemented using the RxJS (Reactive Extensions for JavaScript) library

- It plays a fundamental role in handling asynchronous operations, managing events, and dealing with data streams over time.

- It's a "push" system, meaning the Observable pushes data to its subscribers when data becomes available, rather than the subscriber constantly checking (polling) for data

INTEGRATING THE STUDENTSERVICE

- export class StudentListComponent implements OnInit {
- ngOnInit() {
- this.fetchStudents();
- }
-
- fetchStudents() {
- this.studentService.getStudents().subscribe({
- next: (data) => {
- this.students = data;
- console.log('Students fetched successfully:', data);
- },
- error: (error) => {
- console.error('Error fetching students:', error);
- }
- });
- }

DESIGNING OUR VIEW

```
<div *ngIf="students && students.length > 0">
<table>
<thead>
  <tr>
    <th>Student ID</th>
    <th>First Name</th>
    <th>Last Name</th>
    // add other fields here
        <th>Status</th>
  </tr>
</thead>
  <tr *ngFor="let student of students">
    <td>{{ student.studentId }}</td>
    <td>{{ student.studentFirstName }}</td>
    <td>{{ student.studentLastName }}</td>
      // add other fields here
        <td>{{ student.studentStatus }}</td>

  </tr>

</table>
</div>
```

HTTP SPECIFIC : DELETEMAPPING

- @DeleteMapping: Maps HTTP DELETE requests. Used for deleting resources.

- @DeleteMapping("/{id}") // Maps to /api/students/{id}
- public ResponseEntity<Void> deleteStudent(@PathVariable Long id) {
- // ... delete student by ID
- return ResponseEntity.noContent().build(); // Return 204 No Content
- }

FULL STACK DEV

UniversityApp - Add a new student

Presented by:

Rajeev Khoodeeram

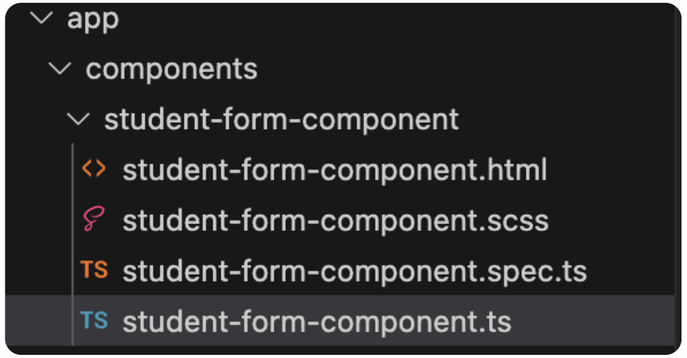
OCTOBER 2025

ADD FUNCTIONALITY - STEPS

- Create the student-form component (generate folders and files)
- Design the Add form for student - StudentFormComponent.html
- Update the StudentService - insert the addStudent method
- Implement the StudentFormComponent.ts (TypeScript)
- Configure Angular Routing for the Form (app.routes.ts)
- Test
- Commit to Github

CREATE THE STUDENT-FORM COMPONENT

- ng g c components/student-form-component




```
<div>
  <form [formGroup]="studentForm" (ngSubmit)="onSubmit()">
  <div>
    <label for="firstName">First Name:</label>
    <input id="firstName" formControlName="studentFirstName" />
  </div>
  <div>
    <label for="lastName">Last Name:</label>
    <input id="lastName" formControlName="studentLastName" />
  </div>
                // ADD OTHER ATTRIBUTES HERE !!
  <div>
    <label for="gender">Gender:</label>
    <select id="gender" formControlName="studentGender">
      <option value="male">Male</option>
      <option value="female">Female</option>
      <option value="other">Other</option>
    </select>
  </div>
  <div>
    <label for="dateOfBirth">Date of Birth:</label>
    <input id="dateOfBirth" type="date" formControlName="studentDateOfBirth" />
  </div>
  <button type="submit">Submit</button>
</form>
</div>

<div *ngIf="successMessage" class="success-message">{{ successMessage }}</div>
<div *ngIf="errorMessage" class="error-message">{{ errorMessage }}</div>
```

REQUESTMAPPING

- [formGroup]="studentForm" —> name of form used in ts file
- (ngSubmit)=“onSubmit()" —> form will submitted using Angular

- We'll use Reactive Forms, which are generally recommended for their scalability, testability, and more explicit structure.

- In Angular, Reactive Forms are a way of building and managing forms in your app programmatically in TypeScript.

MAPPING ANGULAR - JAVA COMPONENTS

Student-form-component.html

```
<p>student-form-component works!</p>
<div>
    <form [formGroup]="studentForm"
(ngSubmit)="onSubmit()">
    <div>
        <label for="firstName">First N
        <input id="firstName"
formControlName="studentFirstName" />
    </div>
    <div>
        <label for="lastName">Last Nam
        <input id="lastName"
formControlName="studentLastName" />
    </div>
    <div>
        <label for="gender">Gender:</l
        <select id="gender"
formControlName="studentGender">
            <option value="male">Male<
```

Student.java

src > main > java > ca > cloudace > backend > model > Student.java > Lang

```
10  @Entity
63      * @param studentStatus
64      */
65     public Student(
66         int studentId,
67         String studentFirstName,
68         String studentEmail,
69         String studentLastName,
70         String studentPhoneNumber,
71         String studentAddress,
72         String studentGender,
73         String studentDateOfBirth,
74         String studentEnrollmentDate,
75         String studentStatus
```

STUDENT-SERVICE.TS

- Remember : View (.html) calls Component (component.ts) which in turn calls Service (service.ts) which will call the Spring backend (controller).
-
- /**
- * Adds a new student.
- * @param student The student to add.
- * @returns An observable containing the added student.
- */
- addStudent(student: Student): Observable<Student> {
- return this.http.post<Student>(`${this.apiUrl}/add`, student);
- }
-

STUDENT-FORM-COMPONENT.TS

- @Component({
- selector: 'app-student-form-component',
- standalone: true,
- imports: [ReactiveFormsModule,CommonModule],
- templateUrl: './student-form-component.html',
- styleUrls: ['./student-form-component.scss']
- })

- **See how we do the imports**
- ReactiveFormsModule: since we are using Angular reactive form
- CommonModule : for http connection

STUDENTFORMCOMPONENT

```
export class StudentFormComponent {
  studentForm!: FormGroup;
  successMessage: string ="";
  errorMessage: string ="";

  constructor(private fb: FormBuilder, private studentService: StudentService
  ) {}

  ngOnInit() : void {
    // Initialize the form or fetch data if needed
    this.studentForm = this.fb.group({
      studentFirstName: ['', Validators.required],
      studentLastName: ['', Validators.required],
      studentEmail: ['', [Validators.required, Validators.email]],
      studentPhoneNumber: ['', Validators.required],
      studentAddress: ['', Validators.required],
      studentGender: ['', Validators.required],
      studentDateOfBirth: ['', Validators.required],
      studentEnrollmentDate: [new Date(), Validators.required],
      studentStatus: ['inactive', Validators.required]
    });
}
```

HOW TO SUBMIT ?

```
onSubmit() : void {

  if (this.studentForm.invalid) {
    // Mark all fields so errors show immediately
    this.errorMessage = 'Failed to add student. Please fill all fields.';
    this.studentForm.markAllAsTouched();
    return; // stop if invalid
  }


    // Call the service to add the student
    this.studentService.addStudent(this.studentForm.value).subscribe({
      next: () => {
        this.successMessage = 'Student added successfully!';
        this.errorMessage = "";
        this.studentForm.reset(); // this is important for success message to appear
      },
      error: (error) => {
        this.errorMessage = 'Failed to add student.';
        this.successMessage = "";
      }
    });
  }
}
```

APP.ROUTES.TS

- {path: "students/add", component: StudentFormComponent}

- Testing : http://localhost:4200/students./add

- Localhost:4200/students/add
  - sends us to StudentFormComponent which is in student-form.component.ts (initialises everything_)

  - displays the view which is student-form.component.html (using the corresponding css)

FULL STACK DEV

Completing Student CRUD with Delete and Edit functionalities

Presented by:

Rajeev Khoodeeram

OCTOBER 2025

STEPS

- Modify the list component (view / html) to add columns for edit and delete
- Update the StudentService - insert the deleteStudent method
- Modify the list component (ts) for delete functionality
- Testing

- Modify the StudentService - insert the updateStudent method and getStudentById
- Modify the form component (ts) to detect edit link has been clicked on the list and execute the updateStudent method
  - Modify the NgInit
  - Modify the OnSubmit
- Configure Angular Routing for the Form (app.routes.ts)
- Test
- Commit to Github

ADDING EDIT AND DELETE LINKS

- Modify the list component (view / html) to add a column - add edit link and delete button

- In the header :
  - <th>Actions</th>

- In the body :
  - <td>
  - <a [routerLink]="['/students/edit', student.studentId]">Edit
  - <button (click)="**deleteStudent**(student.studentId)">Delete</button>
  - </td>

UPDATE STUDENTSERVICE

- Update the StudentService - insert the deleteStudent method
- /**
- * Deletes a student by ID.
- * @param studentId The ID of the student to delete.
- * @returns An observable indicating the result of the delete operation.
- */

- **deleteStudent**(studentId: number): Observable<void> {
- return this.http.delete<void>(`${this.apiUrl}/${studentId}`);
- }

MODIFY THE FORM COMPONENT (TS)

```
  deleteStudent(studentId: number) {
   const confirmed = confirm('Are you sure you want to delete this student?');
    if (confirmed) {
     console.log('Deleting student with ID:', studentId);
     this.studentService.deleteStudent(studentId).subscribe({
      next: () => {
       console.log('Student deleted successfully');
       this.students = this.students.filter(s => s.studentId !== studentId);
      },
      error: (error) => {
       console.error('Error deleting student:', error);
      }
     });
    }
}
```

UPDATE STUDENTSERVICE

```
/**
 * Updates a student by ID.
 * @param studentId The ID of the student to update.
 * @param student The updated student data.
 * @returns An observable containing the updated student.
 */

updateStudent(studentId: number, student: Student): Observable<Student> {
 return this.http.put<Student>(`${this.apiUrl}/${studentId}`, student);
 }
```

MODIFY FORM COMPONENT

- Modify the form component (ts) to detect edit link has been clicked on the list and execute the updateStudent method

- In the class add :
  - **isEditMode: boolean = false;** // to know if we are in edit or add mode
  - **studentId!: number;** // to get id for the student we want to edit

NGONINIT() METHOD

```
//get student Id for editing student
  this.studentId = Number(this.route.snapshot.paramMap.get('studentId'));

// Check if we are in edit mode (if student id has been submitted; then populate the form for the
selected student
  if (this.studentId) {
    this.isEditMode = true;
    this.studentService.getStudentById(this.studentId).subscribe({
      next: (student) => {
        // Populate the form with the student data
        this.studentForm.patchValue(student);
      },
      error: (error) => {
        this.errorMessage = 'Failed to load student data.';
      }
    });
  }
```

INSIDE THE ONSUBMIT

```
//checks if edit mode ; if not then we should add a new student
  if (!this.isEditMode) {

    // Call the service to add the student
    this.studentService.addStudent(this.studentForm.value).subscribe({
      next: () => {
        this.successMessage = 'Student added successfully!';
        this.errorMessage = "";
        this.studentForm.reset(); // this is important for success message to appear
      },
      error: (error) => {
        this.errorMessage = 'Failed to add student.';
        this.successMessage = "";
      }
    });
  }
  else {
    // Call the service to update the student
    this.studentService.updateStudent(this.studentId, this.studentForm.value).subscribe({
      next: () => {
        this.successMessage = 'Student updated successfully!';
        this.errorMessage = "";
        this.studentForm
      },
      error: (error) => {
        this.errorMessage = 'Failed to update student.';
        this.successMessage = "";
      }
    });
  }
```

TESTING

- **app.routes.ts**
  - {path: "students/edit/:studentId", component: StudentFormComponent}

- **Committing to GitHub**
  - >> git branch
  - >> git checkout -b frontend/feature-student-edit-delete
  - >> git add .
  - >> git commit -m "Committing edit and delete functionalities for student only”
  - >> git push

- **On GitHub**
- Create a Pull Request with a comment
- Then if no conflict, Team Leader will approve and merge with the main (check main branch to see updates)