

---

# System Requirements Specification

Index

For

College Management  
System

Version 1.0

# TABLE OF CONTENTS

BACKEND-SPRING BOOT RESTFUL APPLICATION	3
1 Project Abstract	3
2 Assumptions, Dependencies, Risks / Constraints	4
2.1 Department Constraints:	4
2.2 Student Constraints	4
2.3 Teacher Constraints	4
3 Business Validations	5
4 Database Operations	5
5 Rest Endpoints	6
5.1 DepartmentController	6
5.2 StudentController	7
5.3 TeacherController	8
6 Template Code Structure	8
6.1 Package: com.collegemanagement	8
6.2 Package: com.collegemanagement.entity	9
6.3 Package: com.collegemanagement.dto	10
6.4 Package: com.collegemanagement.repository	11
6.5 Package: com.collegemanagement.service	12
6.6 Package: com.collegemanagement.exception	13
6.7 Package: com.collegemanagement.controller	15
7 Method Descriptions	16
8 Considerations	21
FRONTEND-ANGULAR SPA	22
1 Problem Statement	22
2 Proposed Donation Management Wireframe	23
2.1 Home page	23
2.2 Students page	23
2.3 Teachers page	24
2.4 Departments page	24
3 Business-Requirement:	25
4 Constraints	38
7 Execution Steps to Follow for Backend	39
8 Execution Steps to Follow for Frontend	40

# College Management APPLICATION

## System Requirements Specification

---

You need to consume APIs exposed by Backend application in Angular to make application work as FULLSTACK

## BACKEND-SPRING BOOT RESTFUL APPLICATION

### 1 PROJECT ABSTRACT

**College Management** Application is Spring boot RESTful application with MySQL, where it manages students, departments and teachers.

**Following is the requirement specifications:**

	College Management System Application	
Modules		
	1	Department
	2	Student
	3	Teacher
Department Module Functionalities		
	1	Create a Department
	2	Update the existing department details
	3	Get department info by department id
	4	Get all registered departments
	5	Search department Info by department name
	6	Delete an existing department
Student Module Functionalities		
	1	Create a student
	2	Update the existing student
	3	Get a student by Id
	4	Fetch all registered students
	5	Delete an existing student
	6	Search student Info by student name
Teacher Module Functionalities		
	1	Create a teacher

2	Update the existing teacher
3	Get a teacher by Id
4	Fetch all registered teachers
5	Delete an existing teacher
6	Search teacherInfo by teacher name

## 2 ASSUMPTIONS, DEPENDENCIES, RISKS / CONSTRAINTS

### 2.1 DEPARTMENT CONSTRAINTS:

- While updating a department, if departmentId does not exist then the operation should throw a DepartmentNotFoundException with message "Department with Id - " + id + " not found!".
- While fetching the department details by id, if departmentId does not exist then the operation should throw a DepartmentNotFoundException with message "Department with Id - " + id + " not found!".

### 2.2 STUDENT CONSTRAINTS:

- While creating the student, if departmentId does not exist then the operation should throw DepartmentNotFoundException with message "Department with Id - " + id + " not found!".
- While updating a student, if studentId does not exist then the operation should throw StudentNotFoundException with message "Student with Id - " + id + " not found!".
- While fetching the student details by id, if studentId does not exist then the operation should throw StudentNotFoundException with message "Student with Id - " + id + " not found!".
- The student Details are connected through a field department name – applying integrity constraint

### 2.3 TEACHER CONSTRAINTS

- While fetching the teacher details by id, if teacherId does not exist then the operation should throw TeacherNotFoundException with message "Teacher with Id - " + id + " not found!".
- While creating the teacher, if department id does not exist then the operation should throw a DepartmentNotFoundException with message "Department with Id - " + id + " not found!".
- While updating a teacher, if teacherId does not exist then the operation should throw TeacherNotFoundException with message "Teacher with Id - " + id + " not found!".
- The teacher Details are connected through a field department name – applying integrity constraint

## Common Constraints

- For all rest endpoints receiving @RequestBody, validation check must be done and must throw custom exception if data is invalid
- All the business validations must be implemented in dto classes only.
- All the database operations must be implemented on entity object only. Do not change, add, remove any existing methods in service layer
- In Repository interfaces, custom methods can be added as per requirements.
- All RestEndpoint methods and Exception Handlers must return data wrapped in **ResponseEntity**

## 3 BUSINESS VALIDATIONS

### 3.1 Department DTO

- Department name should not be empty, min 3 and max 100 characters.

### 3.2 Student DTO

- Student name should not be empty, min 3 and max 100 characters.
- Department id should not be null.

### 3.3 Teacher Entity

- Teacher name should not be empty, min 3 and max 100 characters.
- Department id should not be null.

## 4 DATABASE OPERATIONS

### 4.1 Department Entity

- **Department** class should be bound to a table named **departments**.
- **id** should be the **primary key**, generated with **IDENTITY**, mapped to column **id**.
- **students** should be a **one-to-many** relationship (**mappedBy = "department"**, **cascade ALL**, and annotated with **JsonManagedReference** to handle JSON serialization)..
- **teachers** should be a **one-to-many** relationship (**mappedBy = "department"**, **cascade ALL**, and annotated with **JsonManagedReference** to handle JSON serialization).

### 4.2 Student Entity

- **Student** class should be bound to a table named **students**.
- **id** should be the **primary key**, generated with **IDENTITY**, mapped to column **id**.
- **department** should be a **many-to-one** relationship to **Department**, joined via column **department\_id** on the **students** table.

### 4.3 Teacher Entity

- **Teacher** class should be bound to a table named **teachers**.
- **id** should be the **primary key**, generated with **IDENTITY**, mapped to column **id**.
- **department** should be a **many-to-one** relationship to **Department**, joined via column **department\_id** on the **teachers** table.

## 5 REST ENDPOINTS

Rest End-points to be exposed in the controller along with method details for the same to be created

### 5.1 DEPARTMENT CONTROLLER

URL Exposed		Purpose
1. /departments		Create a Department
Http Method	POST	
Parameter 1	DepartmentDTO	
Return	ResponseEntity<DepartmentDTO>	
2. /departments/{id}		Update the Department
Http Method	PUT	
Parameter 1	Long (id)	
Parameter 2	DepartmentDTO	
Return	ResponseEntity<DepartmentDTO>	
3. /departments/{id}		Fetches the details of Department by Id
Http Method	GET	
Parameter 1	Long(id)	
Return	ResponseEntity<DepartmentDTO>	
4. /departments/{id}		Delete the Department
Http Method	DELETE	
Parameter 1	Long (id)	
Return	Boolean	
5. /departments/		Fetch all registered Departments
Http Method	GET	
Parameter	-	
Return	ResponseEntity<List<DepartmentDTO>>	
6. /departments/search?name={name}		Fetches the department with the given name
Http Method	GET	
Parameter 1	String (name)	
Return	ResponseEntity<List<D	

	epartmentDTO>>	
--	----------------	--

## 5.2 STUDENT CONTROLLER

URL Exposed		Purpose
1. /students/		Create a Student
Http Method	POST	
Parameter 1	StudentDTO	
Return	ResponseEntity<StudentDTO>	
2. /students/{id}		Update the Student
Http Method	PUT	
Parameter 1	Long (id)	
Parameter 2	StudentDTO	
Return	ResponseEntity<StudentDTO>	
3. /students/{id}		Fetches the details of Student by Id
Http Method	GET	
Parameter 1	Long(id)	
Return	ResponseEntity<StudentDTO>	
4. /students/{id}		Delete the Student
Http Method	DELETE	
Parameter 1	Long (id)	
Return	ResponseEntity<Boolean>	
5. /students/		Fetch all registered Students
Http Method	GET	
Parameter	-	
Return	ResponseEntity<List<Students>>	
6. /students/search?name={name}		Fetches the students with the given name
Http Method	GET	
Parameter 1	String (name)	
Return	ResponseEntity<List<Students>>	

## 5.3 TEACHER CONTROLLER

URL Exposed		Purpose
1. /teachers/		Create a Teacher
Http Method	POST	
Parameter 1	TeacherDTO	
Return	ResponseEntity<TeacherDTO>	
2. /teachers/{id}		Update the Teacher
Http Method	PUT	
Parameter 1	Long (id)	
Parameter 2	TeacherDTO	
Return	ResponseEntity<TeacherDTO>	
3. /teachers/{id}		Fetches the details of Teacher by Id
Http Method	GET	
Parameter 1	Long(id)	
Return	ResponseEntity<TeacherDTO>	
4. /teachers/{id}		Delete the Teacher
Http Method	DELETE	
Parameter 1	Long (id)	
Return	ResponseEntity<Boolean>	
5. /teachers/		Fetch all registered Teachers
Http Method	GET	
Parameter	-	
Return	ResponseEntity<List<TeacherDTO>>	
6. /teachers/search?name={name}		Fetches the teachers with the given name
Http Method	GET	
Parameter 1	String (name)	
Return	ResponseEntity<List<TeacherDTO>>	



## 6 TEMPLATE CODE STRUCTURE

### 6.1 PACKAGE: COM.COLLEGEMANAGEMENT

#### Resources

CollegeManagementApplication (Class)	This is the Spring Boot starter class of the application.	Already Implemented
--------------------------------------	---	---------------------

### 6.2 PACKAGE: COM.COLLEGEMANAGEMENT.ENTITY

#### Resources

Class/Interface	Description	Status
Department (Class)	<ul style="list-style-type: none"><li>• This class is partially implemented.</li><li>• Annotate this class with proper annotation to declare it as an entity class with <b>departmentId</b> as primary key.</li><li>• Map this class with a <b>department table</b>.</li><li>• Generate the <b>departmentId</b> using the IDENTITY strategy.</li></ul>	Partially implemented.
Student (Class)	<ul style="list-style-type: none"><li>• This class is partially implemented.</li><li>• Annotate this class with proper annotation to declare it as an entity class with <b>studentId</b> as primary key.</li><li>• Map this class with a <b>student table</b>.</li><li>• Generate the <b>studentId</b> using the IDENTITY strategy.</li></ul>	Partially implemented.

<b>Teacher (Class)</b>	<ul style="list-style-type: none"> <li>• This class is partially implemented.</li> <li>• Annotate this class with proper annotation to declare it as an entity class with <b>teacherId</b> as primary key.</li> <li>• Map this class with a <b>teacher table</b>.</li> <li>• Generate the <b>teacherId</b> using the IDENTITY strategy.</li> </ul>	Partially implemented.
------------------------	--	------------------------

### 6.3 PACKAGE: COM.COLLEGE MANAGEMENT.DTO

#### Resources

Class/Interface	Description	Status
<b>DepartmentDTO (class)</b>	Use appropriate annotations for validating attributes of this class. (Refer <b>Business Validation</b> section for validation rules).	Partially implemented.
<b>StudentDTO (class)</b>	Use appropriate annotations for validating attributes of this class. (Refer <b>Business Validation</b> section for validation rules).	Partially implemented.
<b>TeacherDTO (class)</b>	Use appropriate annotations for validating attributes of this class. (Refer <b>Business Validation</b> section for validation rules).	Partially implemented.

	(Refer <b>Business Validation</b> section for validation rules).	
--	--	--

## 6.4 PACKAGE: COM.COLLEGE MANAGEMENT.REPOSITORY

### Resources

Class/Interface	Description	Status
<b>DepartmentRepository</b> (interface)	<ol style="list-style-type: none"> <li>Repository interface exposing CRUD functionality for <b>Department</b> Entity.</li> <li>You can go ahead and add any custom methods as per requirements.</li> </ol>	Partially implemented
<b>StudentRepository</b> (interface)	<ol style="list-style-type: none"> <li>Repository interface exposing CRUD functionality for <b>Student</b> Entity.</li> <li>You can go ahead and add any custom methods as per requirements.</li> </ol>	Partially implemented
<b>TeacherRepository</b> (interface)	<ol style="list-style-type: none"> <li>Repository interface exposing CRUD functionality for <b>Teacher</b> Entity.</li> <li>You can go ahead and add any custom methods as per requirements.</li> </ol>	Partially implemented

## 6.5 PACKAGE: COM.COLLEGEMANAGEMENT.SERVICE

### Resources

Class/Interface	Description	Status
<b>DepartmentService(class)</b>	<ul style="list-style-type: none"><li>• Need to provide implementation for Department related functionalities.</li><li>• Add required repository dependency.</li><li>• Do not modify, add or delete any method signature.</li></ul>	To be implemented.
<b>StudentService (class)</b>	<ul style="list-style-type: none"><li>• Need to provide implementation for Student related functionalities</li><li>• Add required repository dependency</li><li>• Do not modify, add or delete any method signature.</li></ul>	To be implemented.
<b>TeacherService (class)</b>	<ul style="list-style-type: none"><li>• Need to provide implementation for Teacher related functionalities</li><li>• Add required repository dependency</li><li>• Do not modify, add or delete any method signature</li></ul>	To be implemented.

## 6.6 PACKAGE: COM.COLLEGEMANAGEMENT.EXCEPTION

### Resources

Class/Interface	Description	Status
<b>GlobalExceptionHandler</b> (class)	<ul style="list-style-type: none"><li>● RestControllerAdvice Class for defining global exception handlers.</li><li>● Contains Exception Handler for <b>InvalidDataException</b> class.</li><li>● Use this as a reference for creating exception handler for other custom exception classes.</li></ul>	Partially implemented.
<b>ResourceNotFound</b> <b>Exception (Class)</b>	<ul style="list-style-type: none"><li>● Custom Exception to be thrown when trying to fetch or delete the department info which does not exist.</li><li>● Need to create Exception Handler for same wherever needed (local or global).</li></ul>	Already created.
<b>DepartmentNotFound</b> <b>Exception (Class)</b>	<ul style="list-style-type: none"><li>● Custom Exception to be thrown when trying to fetch or delete a department info which does not exist.</li><li>● Need to create Exception Handler for same wherever needed (local or global)</li></ul>	Already created.

<b>StudentNotFoundException (Class)</b>	<ul style="list-style-type: none"> <li>• Custom Exception to be thrown when trying to fetch or delete a student info which does not exist.</li> <li>• Need to create Exception Handler for same wherever needed (local or global)</li> </ul>	Already created.
<b>TeacherNotFoundException (Class)</b>	<ul style="list-style-type: none"> <li>• Custom Exception to be thrown when trying to fetch or delete a teacher info which does not exist.</li> <li>• Need to create Exception Handler for same wherever needed (local or global)</li> </ul>	Already created.
<b>ErrorResponse (Class)</b>	<ul style="list-style-type: none"> <li>• Object of this class is supposed to be returned in case of exception through exception handlers</li> </ul>	Already created.

## 6.7 PACKAGE: COM.COLLEGEMANAGEMENT.CONTROLLER

### Resources

Class/Interface	Description	Status
<b>DepartmentController (Class)</b>	<ul style="list-style-type: none"> <li>• Controller class to expose all rest-endpoints for Department related activities.</li> <li>• May also contain local exception handler methods.</li> </ul>	To be implemented

<b>StudentController (Class)</b>	<ul style="list-style-type: none"> <li>Controller class to expose all rest-endpoints for Student related activities.</li> <li>May also contain local exception handler methods.</li> </ul>	To be implemented
<b>TeacherController (Class)</b>	<ul style="list-style-type: none"> <li>Controller class to expose all rest-endpoints for Teacher related activities.</li> <li>May also contain local exception handler methods.</li> </ul>	To be implemented

## 7 METHOD DESCRIPTIONS

### 1. Service Class - Method Descriptions

#### A. DepartmentService – Method Descriptions

Method	Task	Implementation Details
<b>getAllDepartments()</b>	Fetches all departments from the database.	<ul style="list-style-type: none"> <li>- Calls <code>departmentRepository.findAll()</code></li> <li>- Maps the result to list of <code>DepartmentDTO</code> using <code>modelMapper</code></li> <li>- Returns the list of <code>DepartmentDTO</code></li> </ul>
<b>getDepartmentById()</b>	Fetches a department by its ID.	<ul style="list-style-type: none"> <li>- Accepts <code>id</code> as input</li> <li>- Calls <code>departmentRepository.findById(id)</code></li> <li>- If not found, throws <code>DepartmentNotFoundException</code> with message: Department with Id - {id} not found!</li> <li>- Maps <code>Department</code> to <code>DepartmentDTO</code> using <code>modelMapper</code></li> <li>- Returns the <code>DepartmentDTO</code></li> </ul>
<b>createDepartment()</b>	Creates a new department.	<ul style="list-style-type: none"> <li>- Accepts <code>DepartmentDTO</code> as input</li> <li>- Maps <code>DepartmentDTO</code> to <code>Department</code></li> <li>- Saves the department using <code>departmentRepository.save()</code></li> <li>- Maps the saved <code>Department</code> back to <code>DepartmentDTO</code></li> <li>- Returns the created <code>DepartmentDTO</code></li> </ul>
<b>updateDepartment()</b>	Updates an existing department by ID.	<ul style="list-style-type: none"> <li>- Accepts <code>id</code> and <code>DepartmentDTO</code> as input</li> <li>- Calls <code>departmentRepository.findById(id)</code></li> <li>- If not found, throws <code>DepartmentNotFoundException</code> with message: Department with Id - {id} not found!</li> </ul>

		<ul style="list-style-type: none"> <li>- Maps `DepartmentDTO` to `Department` and saves it</li> <li>- Maps updated `Department` to `DepartmentDTO`</li> <li>- Returns the updated `DepartmentDTO`</li> </ul>
<b>deleteDepartment()</b>	Deletes a department by its ID.	<ul style="list-style-type: none"> <li>- Accepts `id` as input</li> <li>- Calls `departmentRepository.deleteById(id)`</li> <li>- Returns `true`</li> </ul>
<b>searchDepartmentsByName()</b>	Searches departments containing name (case-insensitive).	<ul style="list-style-type: none"> <li>- Accepts `name` as input</li> <li>- Calls `departmentRepository.findByNameContainingIgnoreCase(name)`</li> <li>- Maps result list to `DepartmentDTO` list</li> <li>- Returns list of `DepartmentDTO`</li> </ul>

## B. StudentService – Method Descriptions

Method	Task	Implementation Details
<b>getAllStudents()</b>	Fetches all students from the database.	<ul style="list-style-type: none"> <li>- Uses `studentRepository.findAll()`</li> <li>- Maps entity list to DTO list using `modelMapper` and returns it</li> </ul>
<b>getStudentById()</b>	Fetches a student by ID.	<ul style="list-style-type: none"> <li>- Uses `studentRepository.findById(id)`</li> <li>- Throws `StudentNotFoundException` if a student is not found with the message: Student with Id - {id} not found!</li> <li>- Maps entity to DTO using `modelMapper` and returns it</li> </ul>
<b>createStudent()</b>	Creates a new student.	<ul style="list-style-type: none"> <li>- Fetches `Department` by `departmentId` from DTO</li> <li>- Throws `DepartmentNotFoundException` if not found with message: Department with Id - {id} not found!</li> <li>- Maps DTO to entity using `modelMapper`</li> <li>- Saves the student using `studentRepository.save()`</li> <li>- Maps saved entity to DTO and returns it</li> </ul>
<b>updateStudent()</b>	Updates an existing student.	<ul style="list-style-type: none"> <li>- Fetches `Student` by ID using `studentRepository.findById(id)`</li> <li>- Throws `StudentNotFoundException` if not found with message: Student with Id - {id} not found!</li> <li>- Fetches `Department` by `departmentId` from DTO</li> <li>- Throws `DepartmentNotFoundException` if not found with message: Department with Id - {id} not found!</li> <li>- Sets new department to student</li> <li>- Maps DTO to entity and returns mapped DTO</li> </ul>
<b>deleteStudent()</b>	Deletes a student by ID.	<ul style="list-style-type: none"> <li>- Calls `studentRepository.deleteById(id)`</li> <li>- Returns `true` upon successful deletion</li> </ul>



<b>searchStudents ByName()</b>	Searches students by name.	<ul style="list-style-type: none"> <li>- Uses <code>`studentRepository.findByNameContainingIgnoreCase(name)`</code></li> <li>- Maps result list to DTO list and returns it</li> </ul>
------------------------------------	----------------------------	---

### C. TeacherService – Method Descriptions

Method	Task	Implementation Details
<b>getAllTeachers()</b>	Fetch all teachers from the repository and map to DTO list	<ul style="list-style-type: none"> <li>- Uses <code>`teacherRepository.findAll()`</code></li> <li>- Maps list of entities to DTOs using <code>`modelMapper`</code></li> <li>- Returns list of <code>`TeacherDTO`</code></li> </ul>
<b>getTeacherById()</b>	Retrieve a teacher by ID	<ul style="list-style-type: none"> <li>- Uses <code>`teacherRepository.findById(id)`</code></li> <li>- Throws <code>`TeacherNotFoundException`</code> if teacher not found with message: Teacher with Id - {id} not found!</li> <li>- Maps entity to DTO using <code>`modelMapper`</code> and returns it</li> </ul>
<b>createTeacher()</b>	Create a new teacher and assign department	<ul style="list-style-type: none"> <li>- Maps teacherDTO to new <code>`Teacher`</code> entity and sets name</li> <li>- Fetches department using <code>`departmentRepository.findById(...)`</code></li> <li>- Throws <code>`DepartmentNotFoundException`</code> if department not found with message: Department with Id - {id} not found!</li> <li>- Sets department on teacher entity and saves it using <code>`teacherRepository.save(...)`</code></li> <li>- Returns saved entity mapped to <code>`TeacherDTO`</code></li> </ul>
<b>updateTeacher()</b>	Update an existing teacher	<ul style="list-style-type: none"> <li>- Uses <code>`teacherRepository.findById(id)`</code></li> <li>- If found, updates name and department</li> <li>- Throws <code>`DepartmentNotFoundException`</code> if department not found with message: Department with Id - {id} not found!</li> <li>- Throws <code>`TeacherNotFoundException`</code> if teacher not found with message: Teacher with Id - {id} not found!</li> <li>- Saves updated teacher and returns mapped DTO</li> </ul>
<b>deleteTeacher()</b>	Delete teacher by ID	<ul style="list-style-type: none"> <li>- Calls <code>`teacherRepository.deleteById(id)`</code></li> <li>- Returns <code>`true`</code></li> </ul>
<b>searchTeachers ByName()</b>	Search for teachers whose names contain the input string (case-insensitive)	<ul style="list-style-type: none"> <li>- Calls <code>`teacherRepository.findByNameContainingIgnoreCase(name)`</code></li> <li>- Maps list of entities to DTOs and returns</li> </ul>

## 2. Controller Class - Method Descriptions

### A. DepartmentController – Method Descriptions

Method	Task	Implementation Details
<b>getAllDepartments()</b>	Fetches all departments from the database.	<ul style="list-style-type: none"><li>- Calls <code>departmentService.getAllDepartments()</code>.</li><li>- Returns a list of <code>ResponseEntity&lt;DepartmentDTO&gt;</code> with <code>HttpStatus.OK</code>.</li></ul>
<b>getDepartmentById()</b>	Fetches a department by its ID.	<ul style="list-style-type: none"><li>- Accepts <code>id</code> as a path variable.</li><li>- Calls <code>departmentService.getDepartmentById(id)</code>.</li><li>- Returns a <code>ResponseEntity&lt;DepartmentDTO&gt;</code> object with <code>HttpStatus.OK</code> response.</li></ul>
<b>createDepartment()</b>	Creates a new department.	<ul style="list-style-type: none"><li>- Accepts <code>DepartmentDTO</code> as request body (validated).</li><li>- Calls <code>departmentService.createDepartment(departmentDTO)</code>.</li><li>- Returns the created <code>ResponseEntity&lt;DepartmentDTO&gt;</code> with <code>HttpStatus.CREATED</code> response.</li></ul>
<b>updateDepartment()</b>	Updates an existing department by ID.	<ul style="list-style-type: none"><li>- Accepts <code>id</code> as path variable and <code>DepartmentDTO</code> as request body (validated).</li><li>- Calls <code>departmentService.updateDepartment(id, departmentDTO)</code>.</li><li>- Returns the updated <code>ResponseEntity&lt;DepartmentDTO&gt;</code> with <code>HttpStatus.OK</code> response.</li></ul>
<b>deleteDepartment()</b>	Deletes a department by its ID.	<ul style="list-style-type: none"><li>- Accepts <code>id</code> as a path variable.</li><li>- Calls <code>departmentService.deleteDepartment(id)</code>.</li><li>- Returns a boolean wrapped with <code>HttpStatus.ACCEPTED</code> response.</li></ul>
<b>searchDepartmentsByName()</b>	Searches departments based on name.	<ul style="list-style-type: none"><li>- Accepts <code>name</code> as request parameter.</li><li>- Calls <code>departmentService.searchDepartmentsByName(name)</code>.</li><li>- Returns a list of matching <code>ResponseEntity&lt;DepartmentDTO&gt;</code> with <code>HttpStatus.OK</code> response.</li></ul>

### B. StudentController – Method Descriptions

Method	Task	Implementation Details
<b>getAllStudents()</b>	Fetches all students from the database.	<ul style="list-style-type: none"><li>- Calls <code>studentService.getAllStudents()</code>.</li><li>- Returns a list of <code>ResponseEntity&lt;StudentDTO&gt;</code> with <code>HttpStatus.OK</code>.</li></ul>

<b>getStudentById()</b>	Fetches a student by their ID.	<ul style="list-style-type: none"> <li>- Accepts `id` as a path variable.</li> <li>- Calls `studentService.getStudentById(id)`.</li> <li>- Returns a `ResponseEntity&lt;StudentDTO&gt;` object with `HttpStatus.OK`.</li> </ul>
<b>createStudent()</b>	Creates a new student.	<ul style="list-style-type: none"> <li>- Accepts `StudentDTO` as request body (validated).</li> <li>- Calls `studentService.createStudent(studentDTO)`.</li> <li>- Returns the created `ResponseEntity&lt;StudentDTO&gt;` with `HttpStatus.CREATED`.</li> </ul>
<b>updateStudent()</b>	Updates an existing student.	<ul style="list-style-type: none"> <li>- Accepts `id` as path variable and `StudentDTO` as request body (validated).</li> <li>- Calls `studentService.updateStudent(id, studentDTO)`.</li> <li>- Returns the updated `ResponseEntity&lt;StudentDTO&gt;` with `HttpStatus.OK`.</li> </ul>
<b>deleteStudent()</b>	Deletes a student by ID.	<ul style="list-style-type: none"> <li>- Accepts `id` as path variable.</li> <li>- Calls `studentService.deleteStudent(id)`.</li> <li>- Returns `Boolean` value with `HttpStatus.FOUND`.</li> </ul>
<b>searchStudentsByName()</b>	Searches for students by name.	<ul style="list-style-type: none"> <li>- Accepts `name` as request parameter.</li> <li>- Calls `studentService.searchStudentsByName(name)`.</li> <li>- Returns a list of matching `ResponseEntity&lt;StudentDTO&gt;` objects with `HttpStatus.OK`.</li> </ul>

### C. TeacherController – Method Descriptions

Method	Task	Implementation Details
<b>getAllTeachers()</b>	Fetches all teachers from the database.	<ul style="list-style-type: none"> <li>- Calls `teacherService.getAllTeachers()`</li> <li>- Returns a list of `ResponseEntity&lt;TeacherDTO&gt;` with `HttpStatus.OK`</li> </ul>
<b>getTeacherById()</b>	Fetches a teacher by ID.	<ul style="list-style-type: none"> <li>- Accepts `id` as a path variable</li> <li>- Calls `teacherService.getTeacherById(id)`</li> <li>- Returns a `ResponseEntity&lt;TeacherDTO&gt;` with `HttpStatus.OK`</li> </ul>
<b>createTeacher()</b>	Creates a new teacher.	<ul style="list-style-type: none"> <li>- Accepts `TeacherDTO` as request body (validated)</li> <li>- Calls `teacherService.createTeacher(teacherDTO)`</li> <li>- Returns the created `ResponseEntity&lt;TeacherDTO&gt;` with `HttpStatus.CREATED`</li> </ul>

<b>updateTeacher( )</b>	Updates an existing teacher by ID.	<ul style="list-style-type: none"> <li>- Accepts `id` as a path variable and `TeacherDTO` in request body</li> <li>- Calls `teacherService.updateTeacher(id, teacherDTO)`</li> <li>- Returns the updated `ResponseEntity&lt;TeacherDTO&gt;` with `HttpStatus.OK`</li> </ul>
<b>deleteTeacher( )</b>	Deletes a teacher by ID.	<ul style="list-style-type: none"> <li>- Accepts `id` as a path variable</li> <li>- Calls `teacherService.deleteTeacher(id)`</li> <li>- Returns boolean response with `HttpStatus.FOUND`</li> </ul>
<b>searchTeachers ByName()</b>	Searches teachers by name.	<ul style="list-style-type: none"> <li>- Accepts `name` as a request parameter</li> <li>- Calls `teacherService.searchTeachersByName(name)`</li> <li>- Returns a list of `ResponseEntity&lt;TeacherDTO&gt;` with `HttpStatus.OK`</li> </ul>

## 8 CONSIDERATIONS

- A. There is no roles in this application
- B. You can perform the following 3 possible actions

Department
Student
Teacher

# FRONTEND-ANGULAR SPA

## 1 PROBLEM STATEMENT

College management is SPA (Single Page Application) for maintaining all information related to students, teachers and departments in college. It performs all CRUD operations along with searching functionality for all 3 modules.

The core modules of College management app are:

1. Home Page
2. Students Page
3. Teachers Page
4. Departments Page

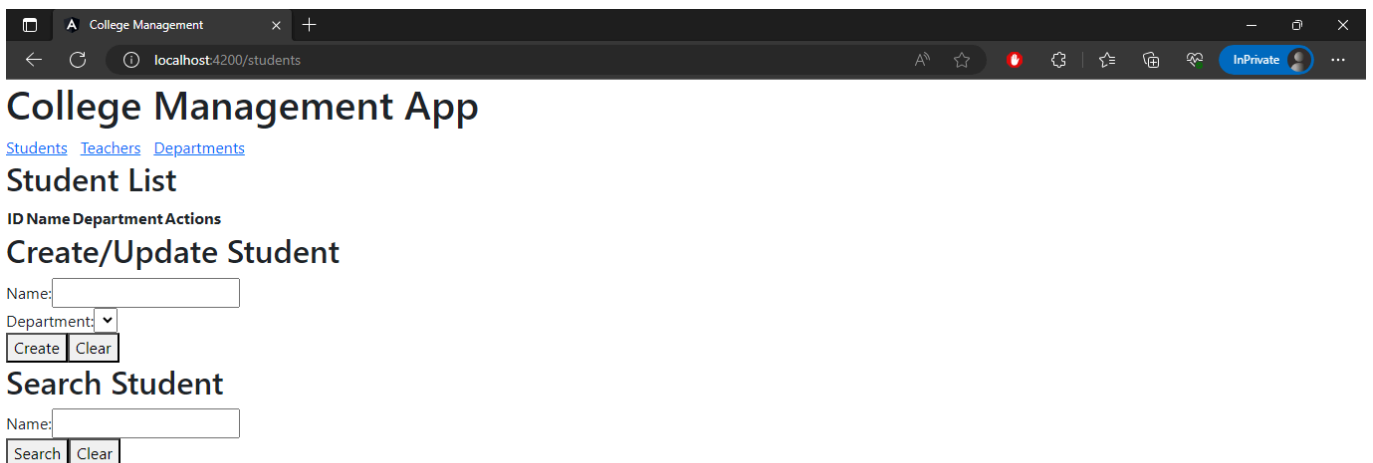
## 2 PROPOSED COLLEGE MANAGEMENT WIREFRAME

UI needs improvisation and modification as per given use case and to make test cases passed.

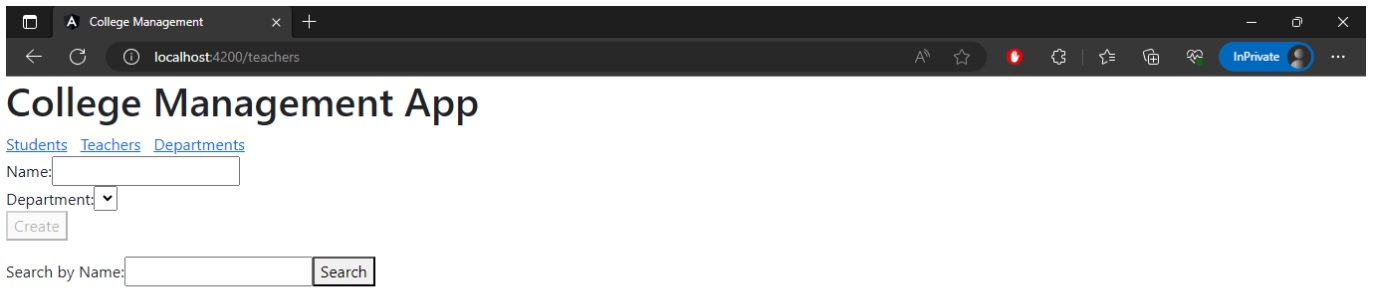
### 2.1 HOME PAGE



### 2.2 STUDENTS PAGE



## 2.3 TEACHERS PAGE



A screenshot of a web browser showing the 'College Management App' interface. The browser's address bar displays 'localhost:4200/teachers'. The page has a dark header with the app name and navigation links: 'Students', 'Teachers' (active), and 'Departments'. Below the header, there is a form with a 'Name:' label and a text input field, a 'Department:' label with a dropdown menu, and a 'Create' button. At the bottom, there is a search section with the label 'Search by Name:', a text input field, and a 'Search' button.

College Management App

[Students](#) [Teachers](#) [Departments](#)

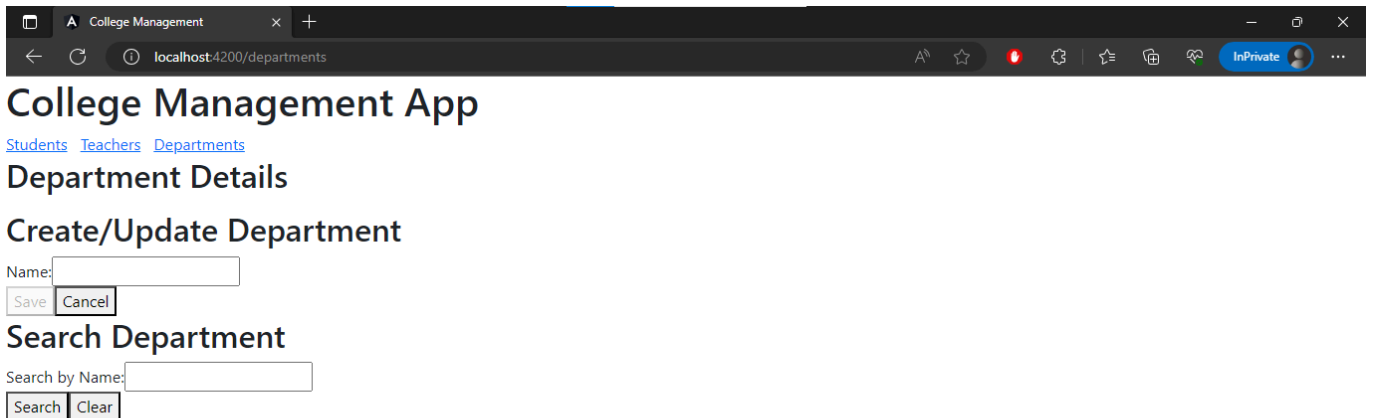
Name:

Department:

Create

Search by Name:  Search

## 2.4 DEPARTMENTS PAGE



A screenshot of a web browser showing the 'College Management App' interface. The browser's address bar displays 'localhost:4200/departments'. The page has a dark header with the app name and navigation links: 'Students', 'Teachers', and 'Departments' (active). Below the header, there is a section titled 'Department Details' followed by 'Create/Update Department'. This section contains a 'Name:' label with a text input field, and 'Save' and 'Cancel' buttons. Below this is a section titled 'Search Department' with a 'Search by Name:' label, a text input field, and 'Search' and 'Clear' buttons.

College Management App

[Students](#) [Teachers](#) [Departments](#)

Department Details

Create/Update Department

Name:

Save Cancel

Search Department

Search by Name:

Search Clear

### 3 BUSINESS-REQUIREMENT:

As an application developer, develop the College management App (Single Page App) with below guidelines:

User Story #	User Story Name	User Story
US_01	Home Page	<p>As a user I should be able to visit the welcome page as default page.</p> <p><b>Acceptance criteria:</b></p> <p><b>HTML Structure:</b></p> <ul style="list-style-type: none"><li>• <code>&lt;h1&gt;</code>: Displays the main title – "College Management App".</li><li>• <code>&lt;nav&gt;</code>: Provides navigation links using <code>routerLink</code> to switch between:<ul style="list-style-type: none"><li>◦ <code>/students</code> route (StudentComponent)</li><li>◦ <code>/teachers</code> route (TeacherComponent)</li><li>◦ <code>/departments</code> route (DepartmentComponent)</li></ul></li><li>• <code>&lt;router-outlet&gt;</code>: Acts as a placeholder where routed components will be rendered.</li></ul> <p><b>Responsibilities:</b></p> <ul style="list-style-type: none"><li>• Main entry layout of the application.</li><li>• Enables top-level navigation across major modules (Students, Teachers, Departments).</li><li>• Dynamically renders the selected route's component view inside the router outlet.</li></ul> <p><b>app-routing.module.ts:</b></p> <p><b>Routing Logic:</b></p> <ul style="list-style-type: none"><li>• Defines all available routes for the app:<ul style="list-style-type: none"><li>◦ <code>/students</code>: Loads the StudentComponent</li><li>◦ <code>/teachers</code>: Loads the TeacherComponent</li><li>◦ <code>/departments</code>: Loads the DepartmentComponent</li></ul></li><li>• Default root route redirects to the homepage ("<code>''</code>" → "<code>/</code>") but note: this will just reload app root as configured.</li></ul> <p><b>Responsibilities:</b></p> <ul style="list-style-type: none"><li>• Centralized configuration of all client-side navigation paths.</li><li>• Connects routes with the respective Angular components for rendering.</li></ul> <p><b>app.component.ts:</b></p> <ul style="list-style-type: none"><li>• Root component class associated with <code>app.component.html</code>.</li><li>• Holds app-wide properties like the title (<code>College</code></li></ul>



		<p>Management) that can be used within the component.</p> <p><b>app.module.ts:</b></p> <ul style="list-style-type: none"> <li>• Acts as the root configuration file for the Angular app.</li> <li>• Registers and wires up modules, components, and services required at the app level.</li> </ul> <p><b>** Kindly refer to the screenshots for any clarifications. **</b></p>
US_02	Students Page	<p>As a user I should be able to see Students page and perform all CRUD operations:</p> <p><b>Acceptance criteria:</b></p> <p><b>Student Component - student.component.ts</b></p> <p><b>HTML Structure</b></p> <ul style="list-style-type: none"> <li>• <b>Heading:</b> The page begins with an <b>h2</b> tag to label the section as <i>Student List</i>.</li> <li>• <b>Student Table:</b> Loops over <b>students</b> using <b>*ngFor</b> to display: <ul style="list-style-type: none"> <li>○ A table is rendered listing all students.</li> <li>○ Columns include: ID, Name, Department, and Actions.</li> <li>○ Each row has action buttons to <i>Edit</i> and <i>Delete</i> a student.</li> </ul> </li> <li>• <b>Student Form:</b> <ul style="list-style-type: none"> <li>○ A form appears below the table to <i>Add</i> or <i>Update</i> student data.</li> <li>○ Includes fields for student name and department (dropdown list).</li> <li>○ A button dynamically changes between “Create” and “Update” depending on context.</li> </ul> </li> <li>• <b>Search Form:</b> <ul style="list-style-type: none"> <li>○ An additional form enables searching students by name.</li> <li>○ Includes <i>Search</i> and <i>Clear</i> buttons.</li> </ul> </li> </ul> <p><b>Component Behavior - student.component.ts</b></p> <p><b>On Component Initialization (ngOnInit)</b></p> <ul style="list-style-type: none"> <li>• Fetches and displays the list of all students.</li> <li>• Fetches and loads available departments for the form.</li> </ul>

### getStudents()

- Fetches all students from the backend via `StudentService`.
- Resets the search form after loading.
- Invoked on component load, after add/update/delete, and after clearing search.

### getDepartments()

- Retrieves all department options using `DepartmentService`.
- Used to populate the department `<select>` dropdown.
- Called during component initialization.

### createStudent()

- When form is submitted in "create" mode:
  - Validates form fields.
  - Constructs a new student object and sends it to the backend.
  - Clears the form and refreshes the student list.

### updateStudent()

- When editing, the form fields are pre-filled based on the selected student.
- On submission in "update" mode:
  - Sends updated data to the backend.
  - Clears form and refreshes list.

### getStudentById(id)

- Fetches the details of a specific student by ID.
- Prefills the form with the fetched data for editing.

### deleteStudent(student)

- When *Delete* is clicked for a student:
  - The respective record is deleted.
  - List of students is refreshed afterward.

### searchStudents()

- Filters student list by name using the search form.
- If the input is cleared, the full list is reloaded.

### clearForm()

- Clicking the *Clear* button resets the form and deselects any student.

		<p><b>Services Utilized – student.service.ts:</b></p> <ul style="list-style-type: none"><li>● <b>StudentService:</b> Handles API operations for student records including create, update, delete, get all, get by ID, and search.</li><li>● <b>DepartmentService:</b> Fetches department data for populating the dropdown in the form.</li></ul> <p><b>Functions &amp; Responsibilities</b></p> <p><b>getAllStudents():</b></p> <ul style="list-style-type: none"><li>● Sends a <b>GET request</b> to retrieve all students.</li><li>● Returns an array of student objects.</li><li>● Used in the component to populate the student list when the UI loads.</li><li>● <b>Returns:</b> The HTTP response containing a list of all student objects.</li></ul> <p><b>getStudentById(id):</b></p> <ul style="list-style-type: none"><li>● Sends a <b>GET request</b> to fetch a single student by its unique ID.</li><li>● Used when you want to edit or view a student’s complete details.</li><li>● <b>Returns:</b> The HTTP response containing the student’s data matching the ID.</li></ul> <p><b>createStudent(student):</b></p> <ul style="list-style-type: none"><li>● Sends a <b>POST request</b> to create a new student with provided data.</li><li>● Prepares a payload including student’s name and departmentId.</li><li>● Called when the user submits the “Create Student” form.</li><li>● <b>Returns:</b> The HTTP response containing the newly created student object.</li></ul> <p><b>updateStudent(student):</b></p> <ul style="list-style-type: none"><li>● Sends a <b>PUT request</b> to update an existing student’s details using their ID.</li><li>● Prepares a payload including updated name and departmentId.</li><li>● Called when the user submits the “Update Student” form.</li><li>● <b>Returns:</b> The HTTP response containing the updated student object.</li></ul> <p><b>deleteStudent(id):</b></p> <ul style="list-style-type: none"><li>● Sends a <b>DELETE request</b> to remove a student by its unique ID.</li><li>● Called when the user clicks the “Delete” button.</li><li>● Removes the student from the backend and refreshes the UI.</li><li>● <b>Returns:</b> The HTTP response confirming successful deletion</li></ul>
--	--	---

		<p>(void).</p> <p><b>searchStudents(name):</b></p> <ul style="list-style-type: none"> <li>• Sends a <b>GET request</b> with query parameters to search students by name.</li> <li>• Useful for quickly finding students without listing all records.</li> <li>• <b>Returns:</b> The HTTP response containing an array of student objects matching the search name.</li> </ul> <p><b>Dynamic Form:</b></p> <ul style="list-style-type: none"> <li>• The form dynamically switches between <i>Create</i> and <i>Update</i> based on whether a student is currently selected.</li> <li>• UI labels and submit actions change accordingly without needing separate forms.</li> </ul> <p><b>** Kindly refer to the screenshots for any clarifications. **</b></p>
US_03	Teachers Page	<p>As a user I should be able to see teachers page and perform all CRUD operations:</p> <p><b>Acceptance criteria:</b></p> <p><b>HTML Structure</b></p> <p><b>1. Create Teacher Form</b></p> <ul style="list-style-type: none"> <li>• <b>Visible only when no teacher is selected.</b></li> <li>• Includes: <ul style="list-style-type: none"> <li>○ A text input field labeled "<b>Name</b>" (validated for <b>required</b> and <b>minlength = 3</b>).</li> <li>○ A dropdown (<b>&lt;select&gt;</b>) labeled "<b>Department</b>", populated with department options.</li> <li>○ A <b>Create</b> button that remains disabled until the form is valid.</li> </ul> </li> <li>• <b>Error messages</b> shown: <ul style="list-style-type: none"> <li>○ "Name is required."</li> <li>○ "Name must be at least 3 characters long."</li> <li>○ "Department is required."</li> </ul> </li> </ul> <p><b>2. Update Teacher Form</b></p> <ul style="list-style-type: none"> <li>• <b>Visible only when a teacher is selected.</b></li> <li>• Contains: <ul style="list-style-type: none"> <li>○ A prefilled input for the selected teacher's <b>name</b>, with same validation and errors as the create form.</li> <li>○ A dropdown to select/change <b>department</b>, also</li> </ul> </li> </ul>

		<p>prefilled and validated.</p> <ul style="list-style-type: none"><li>○ A <b>Update</b> button, disabled until form is valid.</li></ul> <h3>3. Teacher List</h3> <ul style="list-style-type: none"><li>● A list of all teachers, displaying:<ul style="list-style-type: none"><li>○ <b>Teacher name</b></li><li>○ <b>Department name</b></li></ul></li><li>● For each teacher, includes:<ul style="list-style-type: none"><li>○ An <b>Edit</b> button to load the teacher into the update form.</li><li>○ A <b>Delete</b> button to remove the teacher.</li></ul></li></ul> <h3>4. Search Form</h3> <ul style="list-style-type: none"><li>● Consists of:<ul style="list-style-type: none"><li>○ An input field labeled "<b>Search by Name</b>"</li><li>○ A <b>Search</b> button to trigger filtering</li></ul></li></ul> <p><b>Component Behavior - <code>teacher.component.ts</code></b></p> <p><b>Lifecycle Behavior</b></p> <p><b><code>ngOnInit()</code></b></p> <ul style="list-style-type: none"><li>● Invoked once on component initialization.</li><li>● Triggers:<ul style="list-style-type: none"><li>○ <b><code>getTeachers()</code></b> to fetch all teacher records.</li><li>○ <b><code>getDepartments()</code></b> to populate department dropdowns.</li><li>○ <b><code>initializeForm()</code></b> to set up form structure with validations.</li></ul></li></ul> <p><b>Core Functionalities &amp; Methods:</b></p> <p><b><code>initializeForm():</code></b></p> <ul style="list-style-type: none"><li>● Sets up the <b>Reactive Form</b> using <b><code>FormBuilder</code></b>.</li><li>● Contains two fields:<ul style="list-style-type: none"><li>○ <b><code>name</code></b>: Required</li><li>○ <b><code>department</code></b>: Required</li></ul></li></ul> <p><b><code>getTeachers():</code></b></p> <ul style="list-style-type: none"><li>● Calls the teacher service to fetch <b>all teachers</b> from the backend.</li><li>● Populates:<ul style="list-style-type: none"><li>○ <b><code>teachers</code></b>: main list for rendering.</li><li>○ <b><code>filteredTeachers</code></b>: for future use like</li></ul></li></ul>
--	--	---

		<p>filtering/searching.</p> <p><b>getDepartments():</b></p> <ul style="list-style-type: none"><li>• Calls department service to get <b>all available departments</b>.</li><li>• Used to populate dropdowns in both create and update forms.</li></ul> <p><b>createTeacher():</b></p> <ul style="list-style-type: none"><li>• Sends the <b>new teacher object</b> (<b>newTeacher</b>) to the backend via POST.</li><li>• On success:<ul style="list-style-type: none"><li>○ Clears the form by resetting <b>newTeacher</b>.</li><li>○ Refreshes the teacher list via <b>getTeachers()</b>.</li></ul></li></ul> <p><b>editTeacher(teacher):</b></p> <ul style="list-style-type: none"><li>• Accepts a selected teacher from the UI list.</li><li>• Assigns it to <b>selectedTeacher</b> for editing in the update form.</li></ul> <p><b>updateTeacher():</b></p> <ul style="list-style-type: none"><li>• Triggered on submitting the update form.</li><li>• Checks if <b>selectedTeacher</b> is valid.</li><li>• Sends PUT request to update teacher details.</li><li>• On success:<ul style="list-style-type: none"><li>○ Clears the <b>selectedTeacher</b>.</li><li>○ Refreshes the teacher list.</li></ul></li></ul> <p><b>deleteTeacher(id):</b></p> <ul style="list-style-type: none"><li>• Sends a DELETE request using the teacher ID.</li><li>• On success:<ul style="list-style-type: none"><li>○ Removes the teacher from the list locally (without reloading everything).</li></ul></li></ul> <p><b>searchTeacher():</b></p> <ul style="list-style-type: none"><li>• Filters teacher list based on the <b>name entered</b> in the search field.</li><li>• Calls the teacher service to fetch matching records.</li></ul> <p><b>teacher.service.ts:</b></p> <p><b>Purpose</b></p> <ul style="list-style-type: none"><li>• <b>Encapsulates</b> API logic related to teachers.</li><li>• <b>Separates</b> data access logic from component logic</li><li>• <b>Provides</b> reusable methods for Create, Read, Update, Delete (CRUD), and Search functionalities.</li></ul>
--	--	--

## Functions & Responsibilities

### getAllTeachers():

- Sends a **GET request** to retrieve all teachers.
- Returns an array of teacher objects.
- Used in the component to populate the teacher list on UI load.
- **Returns:** The HTTP response containing a list of all teacher objects.

### getTeacherById(id):

- Sends a **GET request** to fetch a single teacher by its unique ID.
- Used when you want to edit or view a teacher's complete details.
- **Returns:** The HTTP response containing the matching teacher's data.

### createTeacher(teacher):

- Sends a **POST request** to add a new teacher.
- The payload includes:
  - **name** of the teacher.
  - **departmentId**.
- Used in the create form submission from the component.
- **Returns:** The HTTP response with the newly created teacher object.

### updateTeacher(id, teacher):

- Sends a **PUT request** to update an existing teacher by ID.
- Payload contains:
  - Updated name.
  - Updated department ID.
- Used during form submission in edit mode.
- **Returns:** The HTTP response with the updated teacher object.

### deleteTeacher(id):

- Sends a **DELETE request** to remove a teacher by ID.
- Invoked when the user clicks on a "Delete" button next to a teacher.
- **Returns:** The HTTP response with no content (void) if deletion is successful.

### searchTeachersByName(name):

- Sends a **GET request** with a query parameter (**name**) to search for teachers whose names match the input.
- Helps in filtering teachers based on search input from the UI.
- **Returns:** The HTTP response containing the list of matched

		<p>teacher objects.</p> <h2>Dynamic Behavior &amp; API Routing</h2> <ul style="list-style-type: none"> <li>• <b>Base URL:</b> Points to the backend's <code>teachers</code> endpoint (<code>http://localhost:8081/collegemanagement/teachers</code>).</li> <li>• <b>Dynamic paths &amp; query params:</b> <ul style="list-style-type: none"> <li>○ <code>/teachers/:id</code> for specific teacher.</li> <li>○ <code>/teachers/search?name=XYZ</code> for name-based filtering.</li> </ul> </li> </ul> <p><b>** Kindly refer to the screenshots for any clarifications. **</b></p>
US_04	Departments Page	<p>As a user I should be able to see department page and perform all CRUD operations:</p> <h2>Acceptance criteria:</h2> <h2>HTML Structure: <code>department.component.html</code></h2> <ul style="list-style-type: none"> <li>• <b>Heading:</b> The page begins with an <b>h2 tag</b> labeled <i>Department Details</i>.</li> <li>• <b>Selected Department Section:</b> A block that only appears if a department is selected (<code>*ngIf="selectedDepartment"</code>). <ul style="list-style-type: none"> <li>○ Displays the selected department's <b>name</b> in an h3 heading.</li> <li>○ Includes two subsections: <ul style="list-style-type: none"> <li>■ <b>Students Section:</b> <ul style="list-style-type: none"> <li>■ Heading in <b>h4 tag</b> labeled <i>Students</i>.</li> <li>■ A <b>button</b> triggers <code>listStudents()</code>.</li> <li>■ A list (<code>ul</code>) dynamically shows student names using <code>*ngFor</code>.</li> </ul> </li> <li>■ <b>Teachers Section:</b> <ul style="list-style-type: none"> <li>■ Heading in <b>h4 tag</b> labeled <i>Teachers</i>.</li> <li>■ A <b>button</b> triggers <code>listTeachers()</code>.</li> <li>■ A list (<code>ul</code>) dynamically shows teacher names using <code>*ngFor</code>.</li> </ul> </li> </ul> </li> </ul> </li> <li>• <b>Departments List Section:</b> Uses <code>*ngFor</code> to display all departments as a list (<code>ul &gt; li</code>). <ul style="list-style-type: none"> <li>○ Each department shows its <b>name</b>.</li> <li>○ Includes action buttons: <ul style="list-style-type: none"> <li>■ <b>View Details</b> → Calls <code>viewDepartment(department)</code></li> </ul> </li> </ul> </li> </ul>



- **Edit** → Calls

`editDepartment(department)`

- **Delete** → Calls

`deleteDepartment(department.id)`

- **Department Form Section:**

A **form** allows adding or updating a department.

- Heading in **h2 tag** labeled *Create/Update Department*.
- Contains:

- **Input field** for department name (`ngModel` bound to `newDepartment.name`).

- **Validation messages:**

- "Name is required." appears if input is touched/non-correct and invalid.

- **Buttons:**

- **Save** (disabled if form is invalid, submits via `saveDepartment()`).
- **Cancel** (resets via `clearForm()`).

- **Search Form Section:**

A form to search departments by name.

- Heading in **h2 tag** labeled *Search Department*.
- Contains:

- **Input field** bound to `searchName`.

- **Buttons:**

- **Search** (submits via `searchDepartment()`).
- **Clear** (resets via `clearSearch()`).

## Department Component —

`department.component.ts`

## Lifecycle

- `ngOnInit()`

- Loads department data (`getDepartments()`).
- Builds the reactive form (`editForm` with `id`, `name` + required validator).
- Calls `loadDepartments()` (duplicate fetch with error logging).

## Functions & Responsibilities

1. **loadDepartments()**
  - Fetches all departments and assigns to `departments`.
  - On failure: logs error message — “Error occurred while loading departments: ...”.
2. **getDepartments()**
  - Fetches all departments and assigns to `departments`.
  - (No explicit error handling in this call.)
3. **viewDepartment(department)**
  - Sets `selectedDepartment` for the details panel.
  - Clears `students` and `teachers` lists to avoid stale data.
4. **listStudents()**
  - If a department is selected, fetches **all** students and assigns to `students`.
5. **listTeachers()**
  - If a department is selected, fetches **all** teachers, filters by `teacher.department.id === selectedDepartment.id`, then assigns to `teachers`.
6. **saveDepartment()**
  - **Create** path: when `newDepartment.id === 0`
    - Sends create request; on success, pushes the returned department into `departments` and clears the form.
  - **Update** path: when `newDepartment.id !== 0`
    - Sends update request; on success, replaces the matching item in `departments` and clears the form.
    - Logs “Department updated:” along with the returned object.
7. **editDepartment(department)**
  - Copies the chosen department into `newDepartment` to prefill the form for editing.
8. **deleteDepartment(id)**
  - Sends delete request; on success, removes the matching item from `departments`.
9. **searchDepartment()**
  - If `searchName` has a value, calls search API and replaces `departments` with the results.
10. **clearForm()**
  - Resets `newDepartment` to default (id `0`, empty name).

#### 11. `clearSearch()`

- Clears `searchName` and reloads full department list via `getDepartments()`.

#### 12. `getDepartmentById(departmentId)`

- Fetches a single department by id and sets it as `selectedDepartment`.

### Dynamic Behavior

- **Initial Load:** Department list is fetched (twice, via both `getDepartments()` and `loadDepartments()`).
- **Details Panel:** Appears only when `selectedDepartment` is set; buttons can load related students/teachers.
- **Create vs Update:** The same form drives both; mode depends on `newDepartment.id` (0 = create, non-zero = update).
- **After Create/Update:** Local list is updated and form is cleared.
- **Delete:** Removes item from the local array after successful backend call.
- **Search:** Replaces list with search results; "Clear" restores full list.
- **Error Logging:** Only `loadDepartments()` prints an error message to the console on failure.

### Department Service — `department.service.ts`

#### Class Members

- **baseUrl:** Base API endpoint for department-related requests.  
Example:  
`http://localhost:8081/collegemanagement/departments`
- **http:** Angular's `HttpClient` injected for making HTTP requests.

## Functions & Responsibilities

### `getAllDepartments():`

- Sends a **GET request** to retrieve all departments.
- Returns an array of department objects.
- Used in the component to populate the department list on UI load.
- **Returns:** The HTTP response containing a list of all department objects.

### `getDepartmentById(id):`

- Sends a **GET request** to fetch a single department by its unique ID.
- Used when you want to edit or view a department's complete details.
- **Returns:** The HTTP response containing the matching department's data.

### `createDepartment(department):`

- Sends a **POST request** to create a new department with provided data.
- Called when the user submits the "Add Department" form.
- On success, the department list is refreshed with the new entry.
- **Returns:** The HTTP response containing the newly created department object.

### `updateDepartment(id, department):`

- Sends a **PUT request** to update an existing department identified by its ID.
- Used when the user modifies department details and clicks "Update."
- Ensures backend data is updated and UI reflects the changes.
- **Returns:** The HTTP response containing the updated department object.

### `deleteDepartment(id):`

- Sends a **DELETE request** to remove a department by its unique ID.
- Used when the user clicks the "Delete" button on the UI.
- Refreshes the UI by removing the deleted department from the list.
- **Returns:** The HTTP response indicating successful deletion (void).

		<p><b>searchDepartmentsByName(name):</b></p> <ul style="list-style-type: none"> <li>• Sends a <b>GET request</b> with query parameters to search departments by name.</li> <li>• Useful for quickly filtering departments without loading all data.</li> <li>• <b>Returns:</b> The HTTP response containing an array of departments matching the search name.</li> </ul> <p><b>Dynamic Behavior</b></p> <ul style="list-style-type: none"> <li>• Each method directly interacts with the backend.</li> <li>• Consumed by <b>DepartmentComponent</b> to display lists, handle creation/update, perform searches, and delete records.</li> <li>• Uses <b>HttpParams</b> in <b>searchDepartmentsByName()</b> to pass the query string.</li> </ul> <p><b>** Kindly refer to the screenshots for any clarifications. **</b></p>
--	--	--

## 4 CONSTRAINTS

1. You should be able to press the "TAB" key and "SHIFT + TAB" to navigate from top field to bottom field and vice-versa.

### NOTE:

There are **47 test cases** on the frontend and **77 test cases** on the backend.  
Ensure that your solution passes **all of the test cases** for a **100% successful submission**.

## 9 EXECUTION STEPS TO FOLLOW FOR BACKEND

1. All actions like build, compile, running application, running test cases will be through Command Terminal.
2. To open the command terminal the test takers need to go to the Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal.
3. cd into your backend project folder
4. To build your project use command:  
**mvn clean package -Dmaven.test.skip**
5. To launch your application, move into the target folder (**cd target**). Run the following command to run the application:  
**java -jar <your application jar file name>**
6. This editor Auto Saves the code.
7. These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.
8. To test any Restful application, the last option on the left panel of IDE, you can find ThunderClient, which is the lightweight equivalent of POSTMAN.
9. To test any UI based application the second last option on the left panel of IDE, you can find Browser Preview, where you can launch the application.
10. Default credentials for MySQL:
  - a. Username: **root**
  - b. Password: **pass@word1**
11. To login to mysql instance: Open new terminal and use following command:
  - a. **sudo systemctl enable mysql**
  - b. **sudo systemctl start mysql**

**NOTE:** After typing any of the above commands you might encounter any warnings.

**>> Please note that this warning is expected and can be disregarded. Proceed to the next step.**

- c. **mysql -u root -p**  
**The last command will ask for password which is 'pass@word1'**

12. Mandatory: Before final submission run the following command:  
**mvn test**

# 10 EXECUTION STEPS TO FOLLOW FOR FRONTEND

1. All actions like build, compile, running application, running test cases will be through Command Terminal.
2. To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal.
3. This is a web-based application, to run the application on a browser, use the internal browser in the environment.
4. You can follow series of command to setup Angular environment once you are in your project-name folder:
  - a. npm install -> Will install all dependencies -> takes 10 to 15 min
  - b. npm run start -> To compile and deploy the project in browser. You can press <Ctrl> key while clicking on localhost:4200 to open project in browser -> takes 2 to 3 min
  - c. npm run test -> to run all test cases. **It is mandatory to run this command before submission of workspace -> takes 5 to 6 min**