System Requirements Specification

Index

For

College Management System

Version 1.0

TABLE OF CONTENTS

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 custom exception.
- While fetching the department details by id, if departmentId does not exist then the operation should throw a custom exception.

2.2 Student Constraints

- While creating the student, if departmentId does not exist then the operation should throw a custom exception.
- While updating a student, if studentId does not exist then the operation should throw a custom exception.
- While fetching the student details by id, if studentId does not exist then the operation should throw a custom exception.
- 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 a custom exception.
- While creating the teacher, if teacherId does not exist then the operation should throw a custom exception.
- While updating a teacher, if teacherId does not exist then the operation should throw a custom exception.
- The teacher Details are connected through a field department name – applying integrity constraint

2.4 VISITORS CAN PERFORM THE FOLLOWING ACTIONS

- Allows to add a new department/student/teacher details
- Allows to delete an existing department/student/teacher
- Allows to search the department/student/teacher on the basis of name
- Allows to display all department/student/teacher

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 onlyDo 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 Entity

- Department name is not null, min 3 and max 100 characters.

3.2 Student Entity

- Student name is not null, min 3 and max 100 characters.
- Department id is not null.

3.3 Teacher Entity

- Teacher name is not null, min 3 and max 100 characters.
- Department id is not null.

4 REST ENDPOINTS

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

4.1 DEPARTMENTCONTROLLER

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

4.2 STUDENTCONTROLLER

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

4.3 TEACHERCONTROLLER

URL Exposed	Purpose
1. /teachers/ Http Method: POST Parameter 1: Teacher Return: Teacher	Create a Teacher
2. /teachers/{id} Http Method: PUT Parameter 1: Long (id) Parameter 2: Teacher Return: Teacher	Update the Teacher

3. /teachers/{id}		Fetches the details of Teacher by Id
Http Method	GET	
Parameter 1	Long(id)	
Return	Teacher	
4. /teachers/{id}		Delete the Teacher
Http Method	DELETE	
Parameter 1	Long (id)	
Return	Boolean	
5. /teachers/		Fetch all registered Teachers
Http Method	GET	
Parameter 1	-	
Return	List<Teachers>	

6. /teachers/search?name={name}		Fetches the teachers with the given name
Http Method	GET	
Parameter 1	String (name)	
Return	List<Teachers>	

5 TEMPLATE CODE STRUCTURE

5.1 PACKAGE: COM.YAKSHA.ASSESSMENTS.COLLEGEMANAGEMENT

Resources

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

5.2 Package: com.yaksha.assessments.collegemanagement.entity

Resources

Class/Interface	Description	Status
Department (Class)	• This class is partially implemented. • Annotate this class with proper annotation to declare it as an entity class with **departmentId** as primary key. • Map this class with a **department table**. • Generate the **departmentId** using the IDENTITY strategy.	Partially implemented.
Student (Class)	• This class is partially implemented. • Annotate this class with proper annotation to declare it as an entity class with **studentId** as primary key. • Map this class with a **student table**. • Generate the **studentId** using the IDENTITY strategy.	Partially implemented.

Teacher (Class)	• This class is partially implemented. • Annotate this class with proper annotation to declare it as an entity class with **teacherId** as primary key. • Map this class with a **teacher table**. • Generate the **teacherId** using the IDENTITY strategy.	Partially implemented.

5.3 PACKAGE: COM.YAKSHA.ASSESSMENTS.COLLEGEMANAGEMENT.DTO

Resources

Class/Interface	Description	Status
DepartmentDTO (class)	Use appropriate annotations from the **Java Bean Validation API** for validating attributes of this class. (Refer **Business Validation** section for validation rules).	Partially implemented.
StudentDTO (class)	Use appropriate annotations from the **Java Bean Validation API** for validating attributes of this class. (Refer **Business Validation** section for validation rules).	Partially implemented.
TeacherDTO (class)	Use appropriate annotations from the **Java Bean Validation API** for validating attributes of this class. (Refer **Business Validation** section for validation rules).	Partially implemented.

	(Refer **Business Validation** section for validation rules).	

5.4 Package: com.yaksha.assessments.collegemanagement.repository

Resources

Class/Interface	Description	Status
DepartmentRepository (interface)	1. Repository interface exposing CRUD functionality for **Department** Entity. 2. You can go ahead and add any custom methods as per requirements.	Partially implemented
StudentRepository (interface)	1. Repository interface exposing CRUD functionality for **Student** Entity. 2. You can go ahead and add any custom methods as per requirements.	Partially implemented
TeacherRepository (interface)	1. Repository interface exposing CRUD functionality for **Teacher** Entity. 2. You can go ahead and add any custom methods as per requirements.	Partially implemented

5.5 PACKAGE: COM.YAKSHA.ASSESSMENTS.COLLEGEMANAGEMENT.SERVICE

Resources

Class/Interface	Description	Status
DepartmentService(class)	• Need to provide implementation for Department related functionalities. • Add required repository dependency. • Do not modify, add or delete any method signature.	To be implemented.
StudentService (class)	• Need to provide implementation for Student related functionalities • Add required repository dependency • Do not modify, add or delete any method signature.	To be implemented.
TeacherService (class)	• Need to provide implementation for Teacher related functionalities • Add required repository dependency • Do not modify, add or delete any method signature	To be implemented.

5.6 PACKAGE: COM.YAKSHA.ASSESSMENTS.COLLEGEMANAGEMENT.EXCEPTION

Resources

Class/Interface	Description	Status
GlobalExceptionHandler (class)	• RestControllerAdvice Class for defining global exception handlers. • Contains Exception Handler for **InvalidDataException** class. • Use this as a reference for creating exception handler for other custom exception classes.	Partially implemented.
ResourceNotFound Exception (Class)	• Custom Exception to be thrown when trying to fetch or delete the department info which does not exist. • Need to create Exception Handler for same wherever needed (local or global).	Already created.
DepartmentNotFound Exception (Class)	• Custom Exception to be thrown when trying to fetch or delete a department info which does not exist. • Need to create Exception Handler for same wherever needed (local or global)	Already created.

StudentNotFoundException (Class)	• Custom Exception to be thrown when trying to fetch or delete a student info which does not exist. • Need to create Exception Handler for same wherever needed (local or global)	Already created.
TeacherNotFoundException (Class)	• Custom Exception to be thrown when trying to fetch or delete a teacher info which does not exist. • Need to create Exception Handler for same wherever needed (local or global)	Already created.
ErrorResponse (Class)	• Object of this class is supposed to be returned in case of exception through exception handlers	Already created.

5.7 PACKAGE: COM.YAKSHA.ASSESSMENTS.COLLEGEMANAGEMENT.CONTROLLER

Resources

Class/Interface	Description	Status
DepartmentController (Class)	• Controller class to expose all rest-endpoints for Department related activities. • May also contain local exception handler methods.	To be implemented
StudentController (Class)	• Controller class to expose all rest-endpoints for Student related activities. • May also contain local exception handler methods.	To be implemented
TeacherController (Class)	• Controller class to expose all rest-endpoints for Teacher related activities. • May also contain local exception handler methods.	To be implemented

6 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

Donation management is SPA (Single Page Application) for registering different NGO under some Donation campaign along with Donor information. It performs all CRUD operations for all 3 modules.

The core modules of Donation management app are:

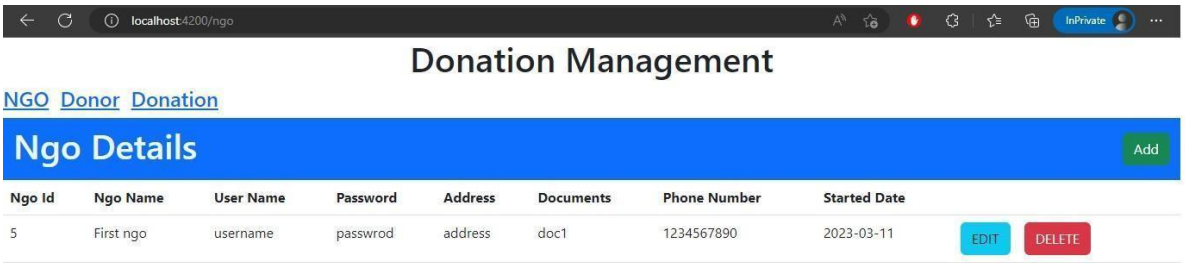
1. Welcome Page
2. Ngo homepage
3. Donor homepage
4. Donation homepage

2 PROPOSED DONATION MANAGEMENT WIREFRAME

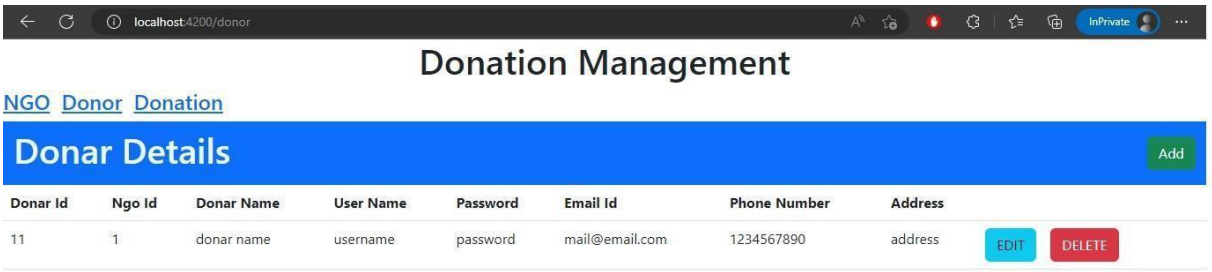
UI needs improvisation and modification as per given use case and to make test cases passed.

2.1 WELCOME PAGE

2.2 NGO HOMEPAGE



2.3 DONOR HOMEPAGE



2.4 DONATION HOMEPAGE

3 BUSINESS-REQUIREMENT:

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

User Story #	User Story Name	User Story
US_01	Welcome Page	As a user I should be able to visit the welcome page as default page. Acceptance criteria: 1. User can click any links shown at homepage.
US_02	Ngo Homepage	As a user I should be able to see Ngo page and perform all CRUD operations: Acceptance criteria: 1. As a user I should be able to furnish following details at the time of creating an ngo. 1.1 Ngo Name 1.2 User Name 1.3 Password 1.4 Address 1.5 Documents 1.6 Phone Number 1.7 Started In 2. Save button should be disabled until all fields are validated. 3. All details fields must be mandatory. If any constraint is not satisfied, a validation message must be shown.
US_03	Donor Homepage	As a user I should be able to see donor page and perform all CRUD operations: Acceptance criteria: 1. As a user I should be able to furnish following details at the time of creating an donor. 1.2 Ngo Id 1.3 Donor Name 1.4 User Name

		1.5 Password 1.6 Email ID 1.7 Phone Number 1.8 Address 2. Save button should be disabled until all fields are validated. 3. All details fields must be mandatory. If any constraint is not satisfied, a validation message must be shown.
US_04	Donation Homepage	As a user I should be able to see donation page and perform all CRUD operations: Acceptance criteria: 1. As a user I should be able to furnish following details at the time of creating an donation. 1.9 Donor Id 1.10 Ngo ID 1.11 Donation Type 1.12 Amount 1.13 Donation Date Save button should be disabled until all fields are validated. 1. All details fields must be mandatory. If any constraint is not satisfied, a validation message must be shown

4 CONSTRAINTS

1. On the page load, input focus must come to the first name input field.
2. You should be able to press the “TAB” key and “SHIFT + TAB” to navigate from top field to bottom field and vice-versa.

7 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 and run test cases use command: mvn clean package**
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. **If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) then you need to use CTRL+Shift+B-command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.**
8. **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.**
9. **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.**
10. **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.**
11. **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**
    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

13. **You need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.**

8 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**
5. **You need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.**