System Requirements Specification

Index

For

SimpliLearn Application

Version 1.0

TABLE OF CONTENTS

SIMPLILEARN APPLICATION

System Requirements Specification

BACKEND-SPRING BOOT RESTFUL APPLICATION

1 PROJECT ABSTRACT

The **SimpliLearn Application** is implemented using Spring Boot with a MySQL database. The SimplILearn App is designed as a comprehensive educational platform that facilitates learning and skill development through a robust course management system.

You are tasked with building a system where students can seamlessly browse, enroll, and manage their courses. The application should provide functionalities to create, update, and delete course records, as well as manage student enrollments. Students should be able to view all their course enrollments and be dynamically managed within courses.

Following is the requirement specifications:

	SimpliLearn Application
Modules	
1	Course
2	Student
Course Module Functionalities	
1	List all courses **(must return all courses by title and that also in list)**
2	Get courses by id
3	Create course
4	Update course by id
5	Delete course by id

Student Module Functionalities	
1	Create student
2	Enroll student in a course
3	Get student enrollments by student id (**should be a custom query).**
4	Manage student enrollment (**should be a custom query).**
5	List all students enrolled in a specific course by course id

Overall Application	
1	Actuator support needs to be added in the properties file. Expose all actuator endpoints except beans.
2	In application.properties file expose a property “profile.validate.data” with value as “This is default profile”. Create application-qa.properties file (for QA profile) and expose a property “profile.validate.data” with value as “This is qa profile”.
3	Create an endpoint in CourseController with following configurations: 1. Method – GET 2. Endpoint - /profile 3. Return – String **The method for this endpoint must read the “profile.validate.data” property file and return its value based on the active profile.**

2 Assumptions, Dependencies, Risks / Constraints

2.1 COURSE CONSTRAINTS

- When updating a course by ID, if the course ID does not exist, the service method should throw a ResourceNotFoundException with the message "Course not found."
- When deleting a course by ID, if the course ID does not exist, the service method should throw a ResourceNotFoundException with the message "Course not found."
- When fetching a course by ID, if the course ID does not exist, the service method should throw a ResourceNotFoundException with the message "Course not found."

2.2 STUDENT CONSTRAINTS

- When viewing a student's enrollment, if the student ID does not exist, the service method should throw a ResourceNotFoundException with the message "Student not found.”
- When enrolling a student in course:
    1) If the student ID does not exist, the service method should throw a ResourceNotFoundException with the message "Student not found."
    2) If the course ID does not exist and an enrollment attempt is made, the service method should throw a ResourceNotFoundException with the message "Course not found."
- When managing student enrollment:
    1) If the student ID does not exist, the service method should throw a ResourceNotFoundException with the message "Student not found."
    2) If the course ID does not exist and an enrollment attempt is made, the service method should throw a ResourceNotFoundException with the message "Course not found."

- When listing students for a course, if the course ID does not exist, the service method should throw a ResourceNotFoundException with the message "Course not found."

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

Course:

- Id must be of type id.
- Title should not be blank, min 3 and max 100 characters.
- Description cannot exceed 1000 characters.

Student:

- Id must be of type id.
- Name should not be blank, min 2 and max 100 characters.
- Email should not be blank and must be of type email and unique in the system.

4 REST ENDPOINTS

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

4.1 COURSECONTROLLER

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>1./api/courses
<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter</td><td>-</td></tr><tr><td>Return</td><td>List<CourseDTO></td></tr></table></td><td>Fetches all the courses</td></tr>
<tr><td>2. /api/courses/{courseId}
<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>Long (courseId)</td></tr><tr><td>Return</td><td>CourseDTO</td></tr></table></td><td>Get existing course by id</td></tr>
</table>

3. /api/courses		
Http Method	POST **The course data to be created must be received in the controller using @RequestBody.**	Create a new course
Parameter 1	-	
Return	CourseDTO	

4. /api/courses/{courseId}		
Http Method	PUT **The course data to be updated must be received in the controller using @RequestBody.**	Updates an existing course by id
Parameter	Long (courseId)	
Return	CourseDTO	

5. /api/courses/{courseId}		
Http Method	DELETE	Delete a course by id
Parameter 1	Long (courseId)	
Return	-	

6. /api/courses/profile		
Http Method	GET	Fetches the profile
Parameter 1	-	
Return	String	

4.2 StudentController

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>1. /api/students/courses/{courseId}
<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter</td><td>Long (courseId)</td></tr><tr><td>Return</td><td>List<StudentDTO></td></tr></table></td><td>List all students enrolled in a specific course</td></tr>
<tr><td>2. /api/students/enroll
<table><tr><td>Http Method</td><td>POST</td></tr><tr><td>Parameter 1</td><td>Long (studentId)</td></tr><tr><td>Parameter 2</td><td>Long (courseId)</td></tr><tr><td>Return</td><td>StudentDTO</td></tr></table></td><td>Enroll a student in a specified course</td></tr>
<tr><td>3. /api/students
<table><tr><td>Http Method</td><td>POST
The student data to be updated must be received in the controller using @RequestBody.</td></tr><tr><td>Parameter</td><td>-</td></tr><tr><td>Return</td><td>StudentDTO</td></tr></table></td><td>Creates a new student</td></tr>
<tr><td>4. /api/students/{studentId}/courses
<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter</td><td>Long (studentId)</td></tr><tr><td>Return</td><td>List<CourseDTO></td></tr></table></td><td>Retrieve all courses that a student is enrolled in</td></tr>
<tr><td>5. /api/students/{studentId}/courses/{courseId}
<table><tr><td>Http Method</td><td>POST</td></tr><tr><td>Parameter 1</td><td>Long (studentId)</td></tr><tr><td>Parameter 2</td><td>Long (courseId)</td></tr><tr><td>Parameter 3</td><td>Boolean(enroll)</td></tr><tr><td>Return</td><td>StudentDTO</td></tr></table></td><td>Manage enrollment of a student in a course</td></tr>
</table>

5 TEMPLATE CODE STRUCTURE

5.1 PACKAGE: COM.SIMPLILEARN

Resources

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

5.2 PACKAGE: COM.SIMPLILEARN.REPOSITORY

Resources

Class/Interface	Description	Status
CourseRepository (interface)	• Repository interface exposing CRUD functionality for Course Entity. • You can go ahead and add any custom methods as per requirements. • It must contain the method for: ○ Finding and Listing courses by their title.	Partially implemented.
StudentRepository (interface)	• Repository interface exposing CRUD functionality for Student Entity. • You can go ahead and add any custom methods as per requirements.	Already implemented.

5.3 PACKAGE: COM.SIMPLILEARN.SERVICE

Resources

Class/Interface	Description	Status
CourseService (interface)	• Interface to expose method signatures for course related functionality. • Do not modify, add or delete any method.	Already implemented.

StudentService (interface)	• Interface to expose method signatures for student related functionality. • Do not modify, add or delete any method.	Already implemented.

5.4 PACKAGE: COM.SIMPLILEARN.SERVICE.IMPL

Class/Interface	Description	Status
CourseServiceImpl (class)	• Implements CourseService. • Contains template method implementation. • Need to provide implementation for course related functionalities. • Do not modify, add or delete any method signature.	To be implemented.
StudentServiceImpl (class)	• Implements StudentService. • Contains template method implementation. • Need to provide implementation for student related functionalities. • Do not modify, add or delete any method signature.	To be implemented.

5.5 PACKAGE: COM.SIMPLILEARN.CONTROLLER

Resources

Class/Interface	Description	Status
CourseController (Class)	• Controller class to expose all rest-endpoints for course 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

5.6 PACKAGE: COM.SIMPLILEARN.DTO

Resources

Class/Interface	Description	Status
CourseDTO (Class)	Use appropriate annotations for validating attributes/fields of this class.	Partially implemented.
StudentDTO (Class)	Use appropriate annotations for validating attributes/fields of this class.	Partially implemented.

5.7 PACKAGE: COM.SIMPLILEARN.ENTITY

Resources

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

5.8 PACKAGE: COM.COM.SIMPLILEARN.EXCEPTION

Resources

Class/Interface	Description	Status
ResourceNotFoundException (Class)	• Custom Exception to be thrown when trying to fetch, update or delete the entity info which does not exist. • Need to create Exception Handler for same wherever needed (local or global)	Already implemented.
ErrorResponse (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	Already implemented.

RestExceptionHandler (Class)	• RestControllerAdvice Class for defining rest exception handlers. • Contains Exception Handler for **ResourceNotFoundException** class. • Use this as a reference for creating exception handler for other custom exception classes	Already implemented.

5.9 PROPERTIES FILES

Resources

Class/Interface	Description	Status
application.properties	• This file is treated as the default properties file for this application. • You need to write properties to add actuator support. • You need to write property to expose all endpoints. • You need to write property to exclude /beans endpoint. • Add "profile.validate.data" property with value as "This is default profile".	Partially implemented.
application-qa.properties	• This file is treated as the qa properties file for this application. • You need to write properties to add actuator support. • You need to write property to expose all endpoints. • You need to write property to	To be implemented.

	exclude /beans endpoint. • Add "profile.validate.data" property with value as "This is qa profile".	

6 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. **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.Please use 127.0.0.1 instead of localhost to test rest endpoints.**
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

12. 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'

13. Mandatory: Before final submission run the following command:

       mvn test

14. 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.