System Requirements Specification Index

For

E-Library System

Version 1.0

IIHT Pvt. Ltd.
fullstack@iiht.com

Table of Contents

E-Library System APPLICATION
System Requirements Specification

1 Project Abstract

E-Library System Application is Spring boot RESTful application with MySQL, where it allows to manage the books, students and students can issue the books from the library.
Following is the requirement specifications:

	E-Library System
Modules	
1	Books
2	Student
Books Module Functionalities	
1	Add a Book
2	List all Issued Books
3	List all books by stream
4	List all issued books with fine
Student Module Functionalities	
1	Student can register itself
2	Student can list all the books with the stream in which student enrolled
3	Student can issue a book
4	List all issued books
5	Return a book

2 Assumptions, Dependencies, Risks / Constraints

2.1 Books Constraints:

- While fetching the books by stream, if stream does not exist then the operation should throw a custom exception.
- By default, the book issue status should be false, after issuing the book the status should be updated to true.

- Books streams can be considered only form the given `["Science", "Commerce", "Arts", "Management", "Media"]`

2.2 Student Constraints

- If Student wants to fetch the books by stream, if stream does not exist then operation should throw custom exception.
- While issuing a book, if the book is already issued then the operation should throw a custom exception.
- While issuing a book, if the book id or student id does not exist then the operation should throw a custom exception.
- While returning a book, if the book is not issued then the operation should throw a custom exception.
- While returning a book, if the book details id does not exist then the operation should throw a custom exception.
- The Expected return date of the book should be exactly after 7 days of the issue date including holidays.
- While returning a book, if the book return date is after the expected return date, then charge a fine of Rs 5 every day.

2.3 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

- Book name is not null, min 3 and max 100 characters.
- Book ISBN is not null, min 10, max 10 characters
- Book author is not null, min 3 and max 100 characters.
- Book publisher number is not null, min 10 and max 10 digits only
- Book published Year not null, min 2020 and max 2040 in range.
- Book edition is not null, min 3 and max 100 characters.
- Book stream is not null, min 3 and max 100 characters.
- Book issue status is not null.
- Student name is not null, min 3 and max 100 characters
- Student email is not null, min 3 and max 100 characters and should be in email format
- Student Stream is not null, min 3 and max 100 characters
- Student phone number is not null, min 10 and max 10 numbers
- Student date of birth is not null, should be past date and in “yyyy-MM-dd” format
- Student address is not null, min 3 and max 100 Characters
- BookIssueDetails Issue date is not null, should be current date and in “yyyy-MM-dd” format
- BookIssueDetails expected return date is not null, should be current or future date and in “yyyy-MM-dd” format
- BookIssueDetails actual date is not null, should be current or future date and in “yyyy-MM-dd” format
- BookIssueDetails find is not null, min is Zero
- BookIssueDetails 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 BooksController

URL Exposed	Purpose
1. /books/add Http Method: POST Parameter 1: BooksDto Return: BooksDto	Add a book to the library
/books/issued Http Method: GET Parameter 1: - Return: List<BooksDto>	Fetches all the Issued books
/books/{stream} Http Method: GET Parameter 1: String(stream) Return: List<BooksDto>	Fetches the list of books for a particular stream
/books/fined Http Method: GET Parameter 1: - Return: List<BooksDto>	Fetches all the books with fine with greater than Zero

4.2 StudentController

URL Exposed	Purpose
/students/register Http Method: POST Parameter 1: StudentDto Return: StudentDto	Register a Student
/students/books/{stream} Http Method: GET Parameter 1: String(stream) Return: List<BooksDto>	Fetches all the books of the Stream
/students/issue/{studentId}/{bookId} Http Method: GET Parameter 1: Long(studentId,bookId) Return: List<BookIssueDetailsDto>	Issue a Book to the Student

/students/issued/{studentId} Http Method: GET Parameter 1: Long (studentId) Return: List<BookIssueDetailsDto>	List all the books issued to a student
/students/books/by/{id } Http Method: GET Parameter 1: Long (id) Return: List<BooksDto >	Fetches all the books of a stream in which a student is enroller
/students/return/{id} Http Method: GET Parameter 1: Long (id) Return: BookIssueDetailsDto	Return a book

5 Template Code Structure

5.1 Package: com.iiht.training.elibrary

Resources

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

5.2 Package: com.iiht.training.elibrary.entity

Resources

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

	declare it as an entity class with **id** as primary key. o Map this class with **student** table. o Generate the **id** using the **IDENTITY** strategy	
BookIssueDetails(class)	o This class is partially implemented. o Annotate this class with proper annotation to declare it as an entity class with **id** as primary key. o Map this class with **book_issue_details** table. o Generate the **id** using the **IDENTITY** strategy	Partially implemented.

5.3 PACKAGE: COM.IIHT.TRAINING.ELIBRARY.DTO

Resources

Class/Interface	Description	Status
BooksDto (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.
BookIssueDetailsDto (class)	Use appropriate annotations from the **Java Bean Validation API** for validating attributes of this class.	Partially implemented.

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

5.4 Package: com.iiht.training.elibrary.model.exception

Resources

Class/Interface	Description	Status
ExceptionResponse (class)	Object of this class is supposed to be returned in case of exception through exception handlers	Already implemented.

5.5 Package: com.iiht.training.elibrary.repository

Resources

Class/Interface	Description	Status
BooksRepository (interface)	1. Repository interface exposing CRUD functionality for **Books** 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

BookIssueDetailsRepository (interface)	1. Repository interface exposing CRUD functionality for **BookIssueDetails** Entity. 2. You can go ahead and add any custom methods as per requirements	Partially implemented

5.6 PACKAGE: COM.IIHT.TRAINING.ELIBRARY.SERVICE

Resources

Class/Interface	Description	Status
BooksService (interface)	Interface to expose method signatures for political party related functionality. Do not modify, add or delete any method	Already implemented.
StudentService (interface)	Interface to expose method signatures for political leader related functionality. Do not modify, add or delete any method	Already implemented.
BookIssueDetailsService (interface)	Interface to expose method signatures for Developments related functionality. Do not modify, add or delete any method	Already implemented.

5.7 PACKAGE: COM.IIHT.TRAINING.ELIBRARY.SERVICE.IMPL

Resources

Class/Interface	Description	Status
BooksServiceImpl (class)	• Implements **BooksService**. Contains template method implementation. • Need to provide implementation for book related functionalities • Add required repository dependency • 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 • Add required repository dependency • Do not modify, add or delete any method signature	To be implemented.
BookIssueDetailsServiceImpl (class)	• Implements **BookIssueDetailsService**. Contains template method implementation. • Need to provide implementation for BookIssueDetails related functionalities	To be implemented.

	• Add required repository dependency • Do not modify, add or delete any method signature	

5.8 PACKAGE: COM.IIHT.TRAINING.ELIBRARY.EXCEPTION

Resources

Class/Interface	Description	Status
GlobalHandler (class)	• RestControllerAdvice Class for defining global exception handlers. • Contains Exception Handler for **InvalidBookDetailsException** class. • Use this as a reference for creating exception handler for other custom exception classes	Partially implemented.

Class/Interface	Description	Status
InvalidStudentDetailsException (Class)	• Custom Exception to be thrown when trying to register the student • Need to create Exception Handler for same wherever needed (local or global)	Already created.
BookNotFoundException (Class)	• Custom Exception to be thrown when trying to	Already created.

	fetch or delete Book info which does not exist. ● Need to create Exception Handler for same wherever needed (local or global)	
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.
BookIssueDetailsNotFoundException (Class)	● Custom Exception to be thrown when trying to fetch or delete a Book Issue Details info which does not exist. ● Need to create Exception Handler for same wherever needed (local or global)	Already created.
AlreadyIssuedException (Class)	● Custom Exception to be thrown when trying to issue a book which is already issued to the student. ● Need to create Exception Handler for	Already created.

	same wherever needed (local or global)	
InvalidStreamException (Class)	• Custom Exception to be thrown when trying to add or fetch a stream which is not from the list of `["Science", "Commerce", "Arts", "Management", "Media"]` • Need to create Exception Handler for same wherever needed (local or global)	Already created.
NotIssuedException (Class)	• Custom Exception to be thrown when trying to return a book which is not issued to the student. • Need to create Exception Handler for same wherever needed (local or global)	Already created.

5.9 Package: com.iiht.training.elibrary.controller

Resources

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

6 Considerations

A. There are 2 roles in this application

Librarian(For performing books related activities)
Student

B. You can perform the following 2 possible actions

Books Actions
Student Actions

7 Execution Steps to Follow

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. To build your project use command:
   mvn clean package -Dmaven.test.skip
4. To launch your application, move into the target folder (cd target). Run the following command to run the application:
   java -jar e-library-0.0.1-SNAPSHOT.jar
5. This editor Auto Saves the code.
6. 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.
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. This is a web-based application, to run the application on a browser, use the internal browser in the workspace. Click on the second last option on the left panel of IDE, you can find Browser Preview, where you can launch the application.
   Note: The application will not run in the local browser
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
    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.**