System Requirements Specification Index

For

E-Library Application (Collaborative)

Version 4.0

IIHT Pvt. Ltd.
IIHT Ltd, No: 15, 2nd Floor, Sri Lakshmi Complex, Off MG Road, Near SBI LHO,
Bangalore, Karnataka – 560001, India
fullstack@iiht.com

Table of Contents

1. Business-Requirement:

1.1 Problem Statement:

Library Management System Application is .net core 3.1 web API application integrated with MS SQL Database, where it allows students and faculty to apply for a book, and the same would be processed by library admin.

1.2 Following is the requirement specification:

	Library Management System
USERS	
1	Admin/librarian
2	Students
Admin /librarian Functionalities	
Librarian	Add A book
	List all issued books
	List all books
	List all issued books with fine
Student Functionalities	
Student 2	Register itself
	List all books
	Issue a book
	List all issued
	Return a book

2. Assumptions, Dependencies, Risks / Constraints

2.1 Admin/librarian Constraints

- While Adding book by admin, all fields should be filled if not filled then operation should throw a custom exception.
- While Adding book by admin, Stream field should be filled or selected as enum value if not selected then operation should throw custom exception.
- While fetching all books, if the loan application id does not exist then operation should throw a custom exception.
- While fetching book details by stream of student, if student stream and id does not exist then operation should throw custom exception.
- While fetching book details with fine bookId does not exist, then the operation should throw a custom exception.

2.2 Students Constraints

- While Register Student Students details, all are should be filled if the operation should throw a custom exception.
- While returning a book if student id and book Id do not exist, then operation should throw custom exception.
- While issuing a new book if student id and book Id do not exist, then operation should throw a custom exception.
- While issuing a new book if book issue status is true exists, then the operation should throw a custom exception.
- While returning a book if the return date crosses the expected return date then calculate fine by 5 rs per day, then the operation should throw a custom exception.

2.3 Common Constraints

- All students must be registered before issuing a book.
- 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 model 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.**
- **All business logic CRUD operations under repository class and write your business logic validation in Services class and related validation use proper user defined exceptions mentioned in above document.**
- **Controller must validate before processing any logic on the database.**

3. Business Validations

3.1 Book Entity:

- Id int is not null, with key attribute
- BookName string is not null, min 3 and max 100 characters.
- ISBN string is not null, min 3 and max 100 characters.
- Author string is not null
- Publisher string value is not null
- Published_Year string value is not null
- Tax_Indicator enum value is not null
- Edition string is not null
- Streams string is not null, min 10 and max 12 characters.
- Issued boolean is not null, valid format

3.2 Student Entity:

- Id int is not null, key attribute
- Name string is not null, min 3 and max 100 characters.
- Email string is not null, valid format
- Streams is not null and enum types
- Phone long is not null, min 10 and max 12 characters.
- DOB DateTime is not null
- Address string is not null, min 10 and max 100 characters.

3.3 Book_Issue Entity:

- Id int is not null, key attribute
- BookId int is not null, > 0
- StudentId int is not null, > 0
- Issue_Date DateTime not null
- Return_Date DateTime not null
- ActualReturn_Date DateTime not null
- Fine double not null
- Returned Boolean not null.

4. Considerations

A. For Role of application users 3 possible values must be used: -
   1. Admin/librarian
   2. Student

B. For Streams of loan following 5 possible Enum values must be used.

1. Science = 1,
2. Commerce = 2,
3. Arts = 3,
4. Management =4,
5. Media = 5

5. Rest Endpoints

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

5.1 BooksController

URL Exposed	Purpose
/issuedbook Http Method: GET Parameter 1: Return: <IEnumerable<Book>>	Fetches details of issued all book.
/bookbystream/{streams} Http Method: GET Parameter 1: Streams streams Return: <IEnumerable<Book>>	Get all book by stream.
/bookbystudentId/{studentId} Http Method: GET Parameter 1: studentId Return: <IEnumerable<Book>>	Get all book details by studentId .
/addbook Http Method: POST Parameter 1: Book model Return: HttpResponse status code	Add new book in database by admin.
/finedbook Http Method: GET Parameter 1: - Return: <IEnumerable<Book>>	Get all book details with have fine and not return within time.

5.2 StudentController

URL Exposed	Purpose
/register Http Method: POST Parameter 1: Student model Return: HttpResponse status code	Register new student to Databse.
/issuebook/{studentId}/{bookId} Http Method: PUT	Get the list of loan allocation that is only applied by

<table>
<tr><td><table>
<tr><td>Parameter 1</td><td>studentId</td></tr>
<tr><td>Parameter 2</td><td>bookId</td></tr>
<tr><td>Return</td><td>boolean</td></tr>
</table></td><td>customer and not processed by clerk</td></tr>
<tr><td>/returnbook/{studentId}/{bookId}<table>
<tr><td>Http Method</td><td>PUT</td></tr>
<tr><td>Parameter 1</td><td>studentId</td></tr>
<tr><td>Parameter 2</td><td>bookId</td></tr>
<tr><td>Return</td><td>boolean</td></tr>
</table></td><td>Return a book</td></tr>
<tr><td>/studentissuebooks/{studentId}<table>
<tr><td>Http Method</td><td>GET</td></tr>
<tr><td>Parameter 1</td><td>studentId</td></tr>
<tr><td>Return</td><td><IEnumerable<Book>></td></tr>
</table></td><td>Get book details that issued for student.</td></tr>
</table>

6. Template Code Structure

6.1 Package: e-library

Resources

Names	Resource	Remarks	Status
Package Structure			
controller	Books, Student Controller	These controllers handle all application Function, update/Edit show information and login existing user.	Partially Implemented
Startup.cs	Startup CS file	Contain all Services settings and SQL Db Configuration.	Already Implemented
Properties	launchSettings.json file	All URL Setting for API	Already Implemented
appsettings.json		Contains connection string for database	Already Implemented

6.2 Package: e-library.Bussinesslayer

Resources

Names	Resource	Remarks	Status
Package Structure			
Interface	ILibraryServices interface	Inside all these interface files contain all business validation logic functions.	Already Implemented

Service	LibraryServices CS file	Using this all class we are calling the Repository method and use it in the program and on the controller.	Partially Implemented
Repository	ILibrary, LIbrary Repository CS file and interface.	All these interfaces and class files contain all CRUD operation code for Db.	Partially Implemented
ViewModels, Enum	StudentViewModel,Streams Enum-	Contain all view Domain entities for show and bind data.	Already Implemented

6.3 Package: e-library.DATALAYER

Resources

Names	Resource	Remarks	Status
Package Structure			
DataLayer	LibraryDBContext cs file	All Database Connection and collection setting class	Already Implemented

6.4 Package: e-library.Entities

Resources

Names	Resource	Remarks	Status
Package Structure			
Entities and Enum	Book, Student, Book_Issue CS file and streams Enum file	All Entities/Domain attribute are used for pass the data in controller and check the various constants using enum	Already Implemented

6.5 Package: e-library.Test

Resources

The e-library.Test project contains all test case classes and functions for code evaluation. Don't edit or change anything inside this project.

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 the Application menu (Three horizontal lines at left top) Terminal → New Terminal.**

3. **On command prompt, cd into your project folder (cd <Your-Project-folder>).**

4. **To connect SQL server from terminal:**
   (e-library/sqlcmd -S localhost -U sa -P pass@word1)
   - **To create database from terminal -**
     1> Create Database e-library_Db
     2> Go

5. **Steps to Apply Migration(Code first approach):**
   - **Press Ctrl+C to get back to command prompt**
   - **Run following command to apply migration-**
     (e-library /dotnet-ef database update)

6. **To check whether migrations are applied from terminal:**
   (e-library /sqlcmd -S localhost -U sa -P pass@word1)

   1> Use e-library_Db
   2> Go

   1> Select * From __EFMigrationsHistory

   2> Go

7. **To build your project use command:**
   (e-library /dotnet build)

8. **To launch your application, Run the following command to run the application:**
   (e-library /dotnet run)

9. **This editor Auto Saves the code.**

10. **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.**

11. **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.**

12. **To run the test cases in CMD, Run the following command to test the application:**
    (You can run this command multiple times to identify the test case status,
    and refactor code to make maximum test cases passed before final submission)
    (e-library /dotnet test --logger "console;verbosity=detailed").

13. **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.**

14. **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.**

15. **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.**