**Library Management System Design Document**

1. Created a Spring Boot project using Spring Initializr.
2. In the package com.jio.librarymanagementsystem1, I created 3 further packages Controller, Model, and service.
3. For this project, we have entities like Book, User, and LibraryTransaction so I put them in model directory.
4. As controllers are implemented to handle the incoming HTTP requests like GET and POST and mapping these controllers to RESTful API endpoints so I added 3 classes for the 3 entities in that.
5. As services is where we implement the logic so this is where we can perform operations like borrowing, returning, renewing books, etc.

**Detailed Design:**

* + - **LibraryBookController**:
    - API Endpoints: It will be just the “/id” while for some services like getAllBooks only the request mapping is enough which is “/books” here.
    - Interaction with BookService: So just like in our Jio services code, Service folder will have interfaces with the abstract methods like createBook, getBookById and their corresponding implementation classes for all 3 entities.

We will just create bookService object as we will construct instance of the LibraryBookController with a bookService as a parameter.

* + - **Data Structures used in bookServiceImpl class:** Since we are using a collection of books which can go up and down in number depending on whether a user has issued a book, returned a book or new book has been created or deleted so we will need a dynamic array “ArrayList” data structure and we will makes the books arrayList of type List which is possible as arrayList implements List. We do this so that we can use the stream method to easily search books by id.
    - **LibraryTransactionController**:
    - API Endpoints: It will be “/id”, “/borrow”, /return/{transactionId}, /renew/{transactionId} and the request mapping will obviously be "/transactions".
* Interaction with LibraryTransactionService: So the LibraryTransactionService interface will contain abstract methods like borrowBook, returnBook, getAllTransactions and will create a LibraryTransactionService object as we will construct instance of the LibraryTransactionController with a LibraryTransactionService as a parameter.

• **Data Structures used in LibraryTransactionServiceImpl class:** Exactly same logic as the BookController above.

* + - **LibraryUserController**:
    - API Endpoints: It will just be “/id” for GET requests like getUserById or updateUser, while for some like getAllUsers or POST request like createUser just request mapping is there which will be “/users”.
* Interaction with UserService: So the UserService interface will contain abstract methods like getAllUsers, getUserByid, deleteUser and for the LibraryUserControllerClass we will create a UserService object as we will construct instance of the LibraryUserController with a userService as a parameter.
* **Data Structures used in UserServiceImpl class:** Exactly same logic as the BookController above.

**Detailed Design for Testing:**

* For the service classes testing I used the Assert method of junit class. It is a method useful in determining Pass or Fail status of a test case, The assert methods are provided from the package org.junit.jupiter.api.Assertions by the class org.junit.Assert which extends java.lang.Object class.

There are various types of assertions like Null, Equals etc.