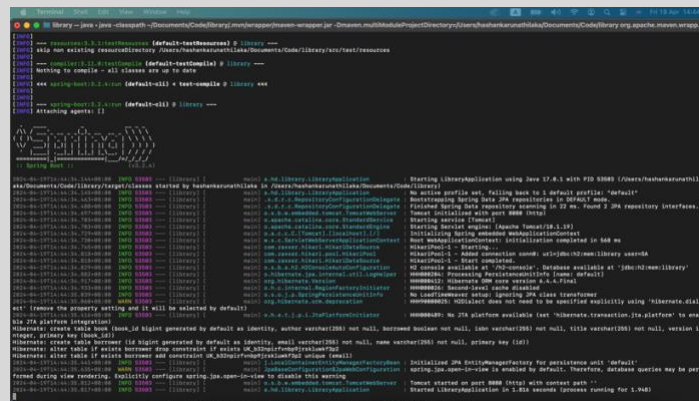
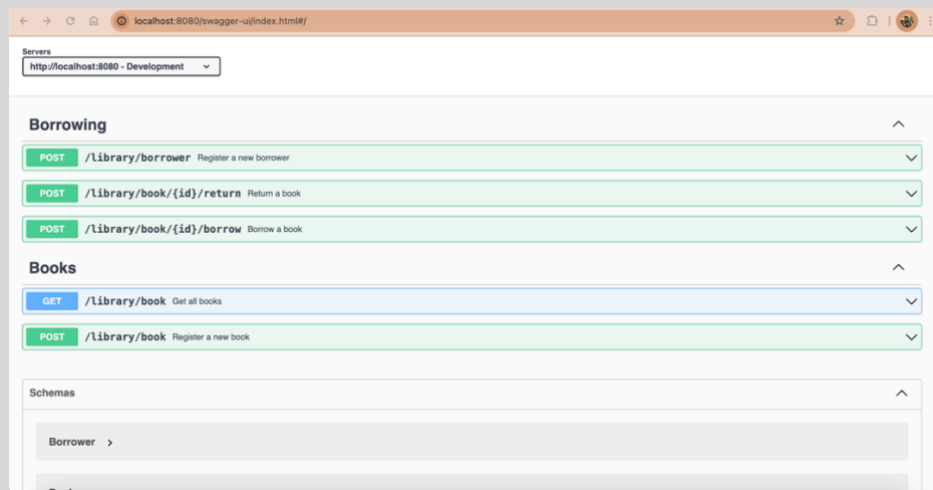


How to run the app:

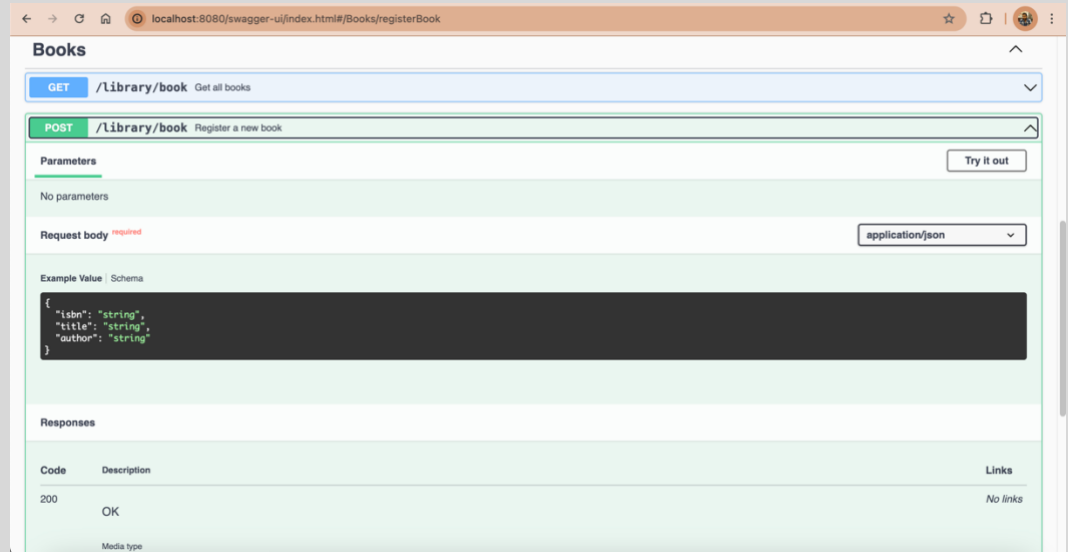
1. Make sure java 17 is selected in the terminal. JAVA_HOME environment variable should be set to the path of jdk17 installation.
2. In the project root folder, run the command: `./mvnw spring-boot:run`
3. It will start the app. Please go to this swagger url to interact with the app. <http://localhost:8080/swagger-ui/index.html#/>



4. <http://localhost:8080/swagger-ui/index.html#/> has the swagger UI with all endpoints and guidance to how to interact with the APIs. There are 5 endpoints provided.
 - a. Register a new borrower.
 - b. Get all books.
 - c. Register a new book.
 - d. Borrow a book of a provided book id.
 - e. Return a book of a provided book id.
5. Sample steps:
 - a. Open swagger url once the app is running after step 2 above.



- b. Expand **POST /library/book** and click “Try it out”

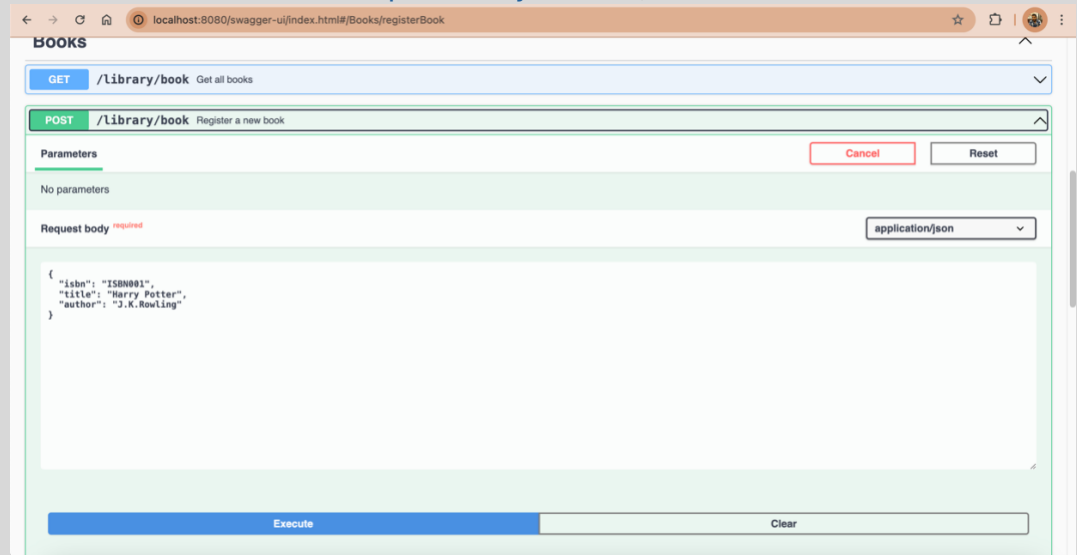


The screenshot shows the Swagger UI for the **POST /library/book** endpoint. The interface includes a 'Try it out' button in the top right corner of the endpoint section. Below the endpoint name, there are tabs for 'Parameters' and 'Request body'. The 'Request body' tab is selected, showing a required field with a dropdown menu set to 'application/json'. An example value is provided in a dark box:

```
{  "isbn": "string",  "title": "string",  "author": "string"}
```

. The 'Responses' section at the bottom shows a 200 status code with the description 'OK' and 'No links'.

- c. Add book details in the **request body** section, and click **Execute** button.

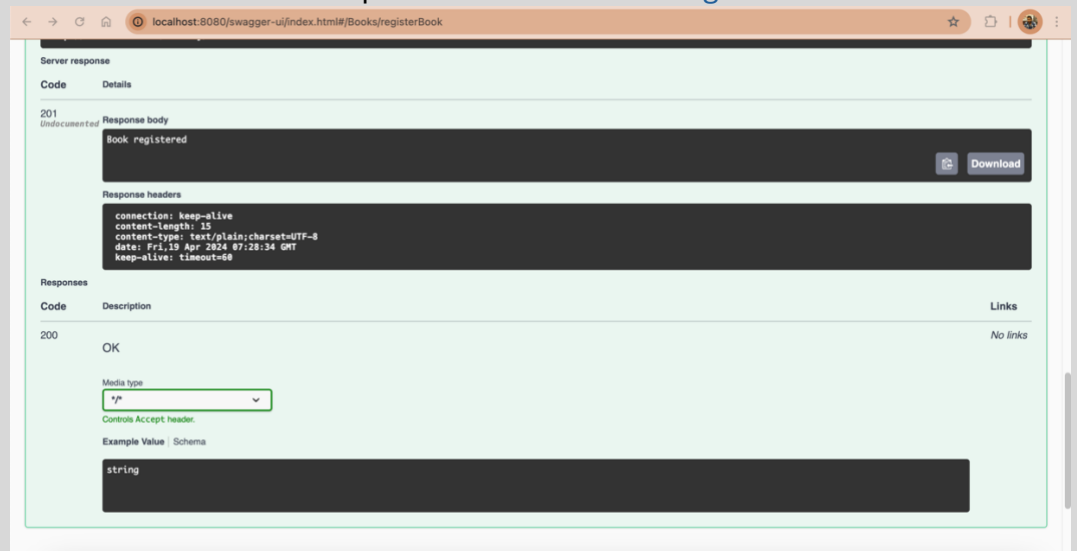


The screenshot shows the Swagger UI for the **POST /library/book** endpoint. The 'Request body' tab is selected, and the example value has been replaced with the following JSON:

```
{  "isbn": "ISBN9801",  "title": "Harry Potter",  "author": "J.K.Rowling"}
```

. The 'Execute' button is visible at the bottom of the interface, next to a 'Clear' button.

- d. Scroll down to see the response. Here it is “Book registered”



The screenshot shows the Swagger UI for the **POST /library/book** endpoint. The 'Server response' section is expanded, showing a 201 status code with the description 'Book registered'. The response body is displayed in a dark box:

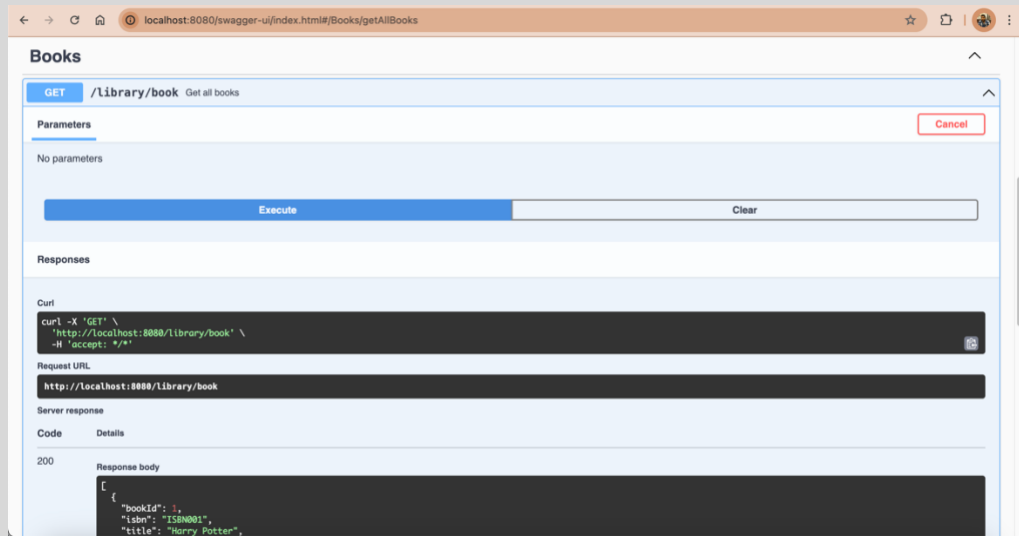
```
Book registered
```

. The response headers are also visible:

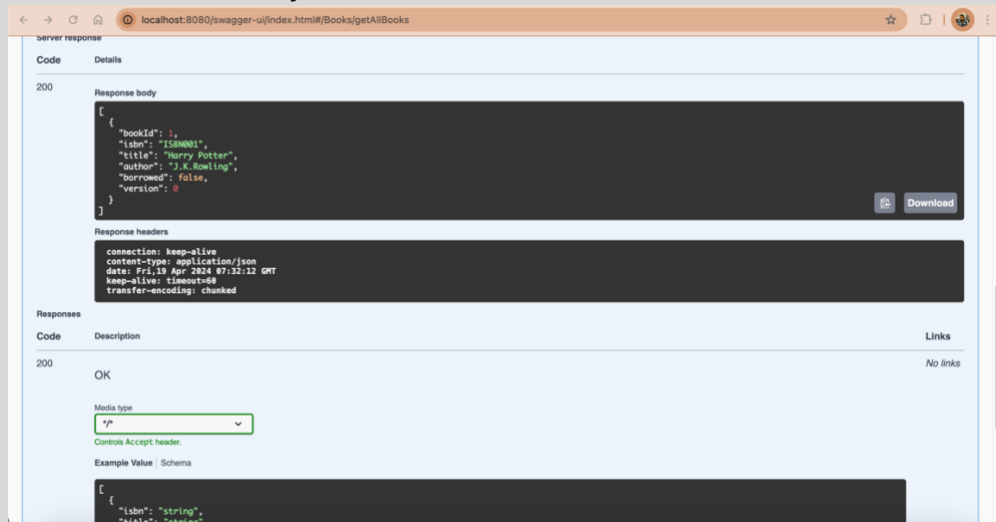
```
connection: keep-alive
content-length: 15
content-type: text/plain; charset=UTF-8
date: Fri, 19 Apr 2024 07:28:34 GMT
keep-alive: timeout=60
```

. The 'Responses' section at the bottom shows a 200 status code with the description 'OK' and 'No links'.

- e. Now go to [GET /library/book](#) section and click “Try it out”. Then click execute. It will get all books in the system. Then scroll down to see the result.



- f. We can see the newly added book.



- g. Please follow similar way to test other endpoints.

Design decisions:

- In memory H2 database was selected as the database of this app to be able to run easily on other machines. However, the database can be easily switched by adding different drivers, dialect, and connection details.
- There are no data of books or borrowers preloaded into the database. For the testing, first have to add new books via book register endpoint.
- Since there were no mention about keeping data of which borrower borrowed a particular book, it is not implemented for the simplicity.

- The APIs will do the following validations to keep the data consistency in the database.
 - To borrow, a book should be available. (not borrowed already)
 - To return, a book should be already borrowed.
 - Optimistic locking is implemented to prevent multiple users requesting to borrow the same book concurrently.
 - When registering a new book, ISBN of previous books is checked to ensure, books with same ISBN have the same name and author.