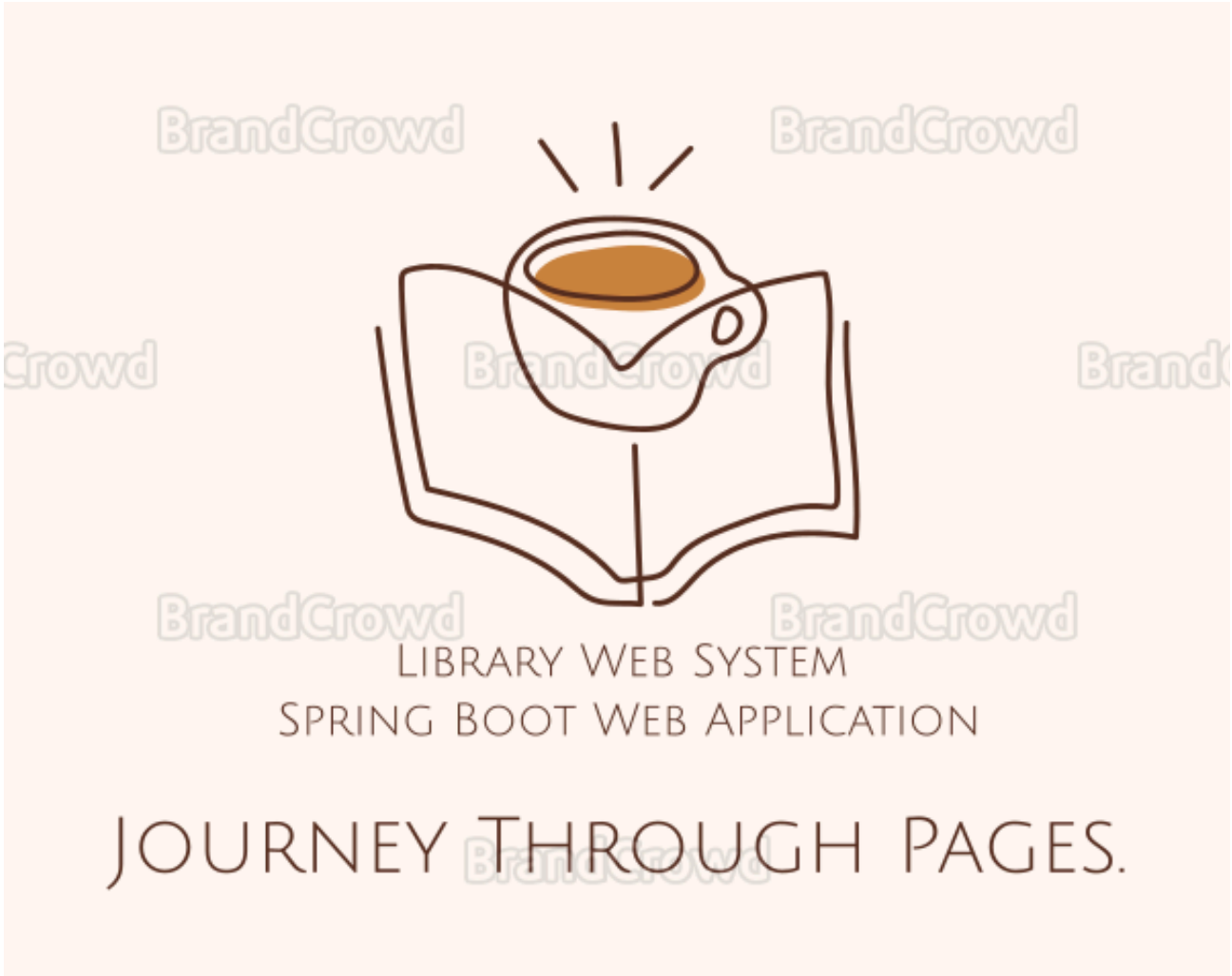
# Library Web System
# Spring Boot Web-App

Course Name & Code: Web Programming - SEN3004

Authors:

- Atena Jafari Parsa
- Baraah Ammar Mohammed Al-shiaani




Logo Retrieved from: https://www.brandcrowd.com/

## ● Project Aim

The Library Web System is a Spring Boot web application that performs CRUD operations on books.
The desired books can be selected and borrowed by the user until a specified date.
Two books have been borrowed by default and the due date is set to be one month after the date that the book was borrowed.

If the user borrows a book, the default due date would be two weeks after the day the book was borrowed. However, the due date can be adjusted by the user to any date after the borrow date.

The application has two main sections:

1. The Admin Section

The home page will ask for the role of the end-user. If Admin is selected, the admin will be able to view and search among all existing books, add a new book and edit or delete the existing books. All the changes applied will be applied to the database simultaneously.

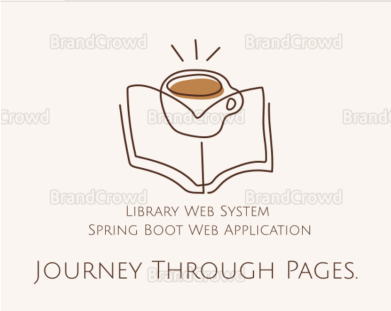
2. The User Section

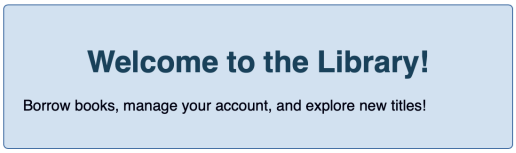
The user will be able to view and search among all the books, except the ones that have been deleted by the admin. The user will be able to borrow a book that has not already been borrowed and by default, the due date would be 14 days after the borrow date but the date can be modified by the user to any date after the borrow date. The user will also have the option to view the details all the books that have been borrowed apart from the borrow and due date.

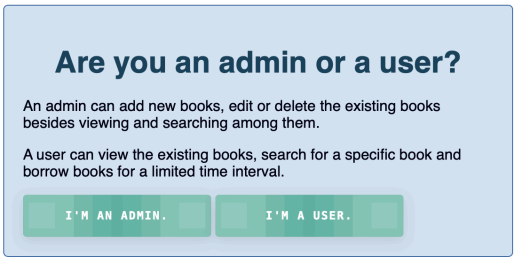
## ● General Walkthrough of the Project

**Index Page :**

- The index page, often the main landing page of the website, could serve as a starting point for users:
    - Introduction to the library system and its functionalities.
    - Information about browsing and searching for books.
    - Links to navigate to specific sections like the admin page and user page.



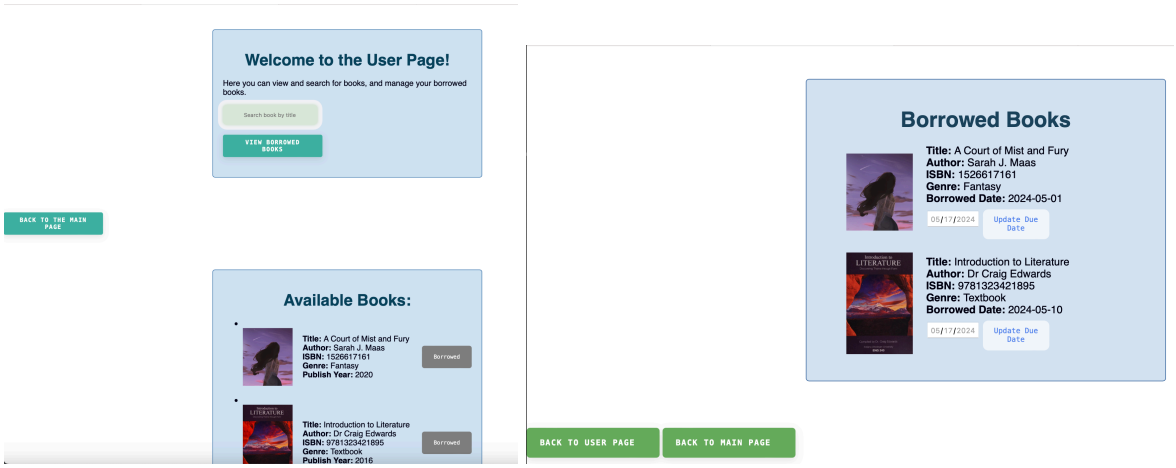




In the home page, the end-user is asked to specify his/her role. If the admin button is chosen, the user would be redirected to the admin page, if the user button is chosen the user would be redirected to the user page (The procedure will be explained in Functionalities.)

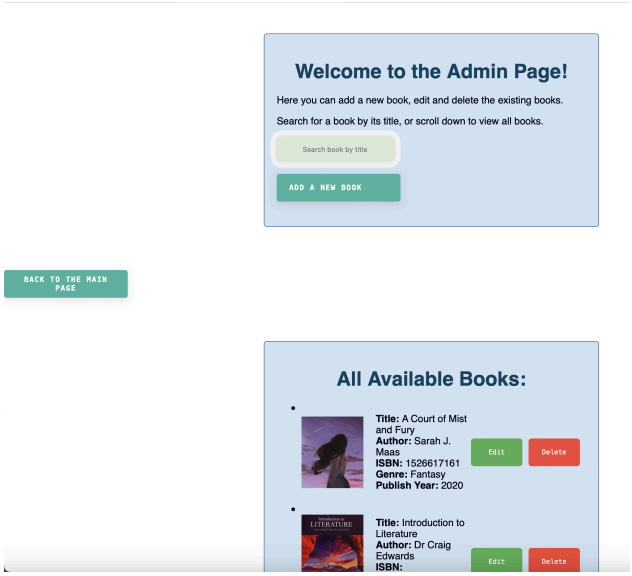
**User Page :**

- Viewing a list of available books (potentially with search and filter options).
- Searching for books by title using a search bar.
- Borrowing books: Users will be able to select books and initiate a borrowing process.
- Viewing their borrowing history: This shows a list of borrowed books and their due dates.



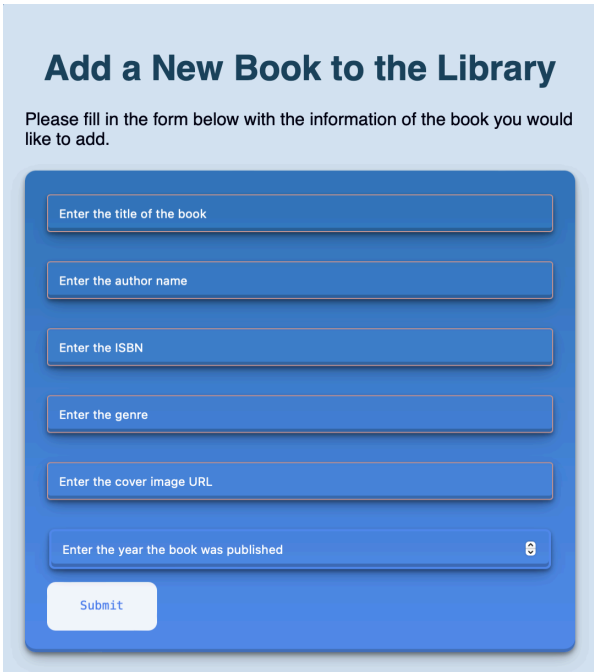

**Admin Page:**

- The admin page allows for managing the book collection:
    - Viewing a list of all books with details like title, author, ISBN, genre, and cover image.
    - Searching for books by title using a search bar.
    - Editing existing books by clicking an "Edit" button, which redirects to an edit page with the book ID.
    - Deleting books with confirmation prompts.
    - Adding new books through a dedicated form.




## Adding Books:

- A separate "Add Book" page provides a form for entering new book details, including title, author, ISBN, genre, publish year and cover image URL.
- Upon form submission, the system sends a POST request to the server to add the new book.
- According to the code, the fields title, author and isbn are required fields. The publish year is an integer so only numerical values will be accepted. Also, it must be in range [1900, 2024].



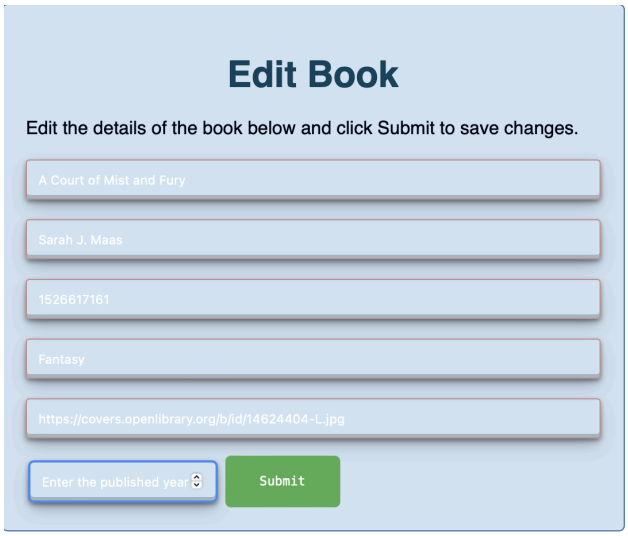

```
<div class="container">
    <h1>Add a New Book to the Library</h1>
    <p class="description">Please fill in the form below with the information of the book you would like to add.</p>
    <form class="form">
        <input placeholder="Enter the title of the book" class="input" type="text" required>
        <input placeholder="Enter the author name" class="input" type="text" required>
        <input placeholder="Enter the ISBN" class="input" type="text" required>
        <input placeholder="Enter the genre" class="input" type="text">
        <input placeholder="Enter the cover image URL" class="input" type="text">
        <input placeholder="Enter the year the book was published" class="input" type="number" min="1900" max="2024" required>

        <button>Submit</button>
    </form>
</div>
```

The code above shows the constraints applied to the input when adding a new book.

**Editing Book Details:**

- An edit option for each book exists for the admin which provides a form for editing the existing book details, including title, author, ISBN, genre, publish year and cover image URL.
- The system sends a PUT request to the server to edit the selected book.
- The same constraints as adding a new book are applied when editing an existing book.

```
<div class="container">
    <h1>Edit Book</h1>
    <p class="description">Edit the details of the book below and click Submit to save changes.</p>
    <form id="editBookForm">
        <input id="title" placeholder="Enter the title of the book" class="input" type="text">
        <input id="author" placeholder="Enter the author name" class="input" type="text">
        <input id="isbn" placeholder="Enter the ISBN" class="input" type="text">
        <input id="genre" placeholder="Enter the genre" class="input" type="text">
        <input id="coverImageUrl" placeholder="Enter the cover image URL" class="input" type="text">
        <input id="publish_year" placeholder="Enter the published year" class="input" type="number">
        <button type="submit">Submit</button>
    </form>
</div>
```




**Book Management:**

- The BookController provides a REST API for managing books:
    - Retrieving a list of all books (/api/books).
    - Searching books by title (/api/books/search).
    - Retrieving details of a specific book (/api/books/{id}).
    - Adding a new book (/api/books/add).
    - Updating an existing book (/api/books/edit/{id}).
    - Deleting a book (/api/books/delete/{id}).

**Borrowing System:**

- The LoanController manages book borrowing through a dedicated API:
    - Borrowing a book (/api/loans/borrow) requires sending a request with book ID, borrowed date, and due date.
    - Retrieving a list of borrowed book IDs (/api/loans/borrowed).

**How This Connects to Other Parts(repository classes):**

- The BookController likely interacts with this BookRepository to implement functionalities like:
    - Retrieving all books (findAll() method of JpaRepository).
    - Searching books by title (findByTitleContainingIgnoreCase custom method).

    - Saving new or updated books (save() method of JpaRepository).

- The LoanController (and potentially the BookController) interact with this LoanRepository to implement functionalities like:
    - Checking if a book is available for borrowing before allowing a borrow request (using isBookBorrowed).

**getAllBooks:** This method retrieves a list of all books from the BookRepository using the findAll() method (provided by Spring Data JPA). It returns this list of Book objects to the caller (the BookController).

**getBookById:** This method takes a Long parameter representing the book ID. It retrieves the book with the matching ID from the BookRepository using the findById(id) method. If a book is found, it returns the Book object. Otherwise, it returns null.

**saveBook:** This method takes a Book object as a parameter. It likely performs some validation or processing before delegating the actual saving task to the BookRepository using the save(book) method. This method persists the book data in the database and returns the saved Book object.

**deleteBook:** This method takes a Long parameter representing the book ID. It delegates the task of deleting the book with the matching ID to the BookRepository using the deleteById(id) method.

**searchBooksByTitle:** This method takes a String parameter representing the book title (or a part of it). It utilizes the custom method findByTitleContainingIgnoreCase provided by the BookRepository to search for books whose titles contain the search term (ignoring case sensitivity). It returns a list of Book objects matching the search criteria.

**Loan service**

- These methods (getAllLoans, getLoanById, saveLoan, deleteLoan) provide basic CRUD functionalities for loan data but might not be directly used by the controllers you shared earlier.

**getOverdueLoans:**

- This method retrieves a list of all loans (findAll from LoanRepository).
- It uses Java Streams to filter this list, keeping only loans where the isOverdue method of the Loan class returns true (likely indicating loans that have passed their due date).
- Finally, it collects the filtered loans into a new list using Collectors.toList().

**borrowBook:**

- This method takes three parameters:
  - bookId: ID of the book to be borrowed.
  - borrowedDate: Date the book is borrowed.
  - dueDate: Date the book is due for return.
- It first checks if the book exists using bookRepository.findById(bookId).
- If the book is not found, it throws an IllegalArgumentException with a message indicating the issue.
- If the book is found, it creates a new Loan object with the provided book, borrowed date, and due date.
- Finally, it saves the new loan entity using loanRepository.save(loan) and returns the saved Loan object.

**getBorrowedBookIds:**

- This method retrieves all loans (findAll from LoanRepository).
- It uses Java Streams to:
  - Map each loan to its corresponding book's ID using loan.getBook().getId().
  - Remove duplicates using distinct().
  - Collect the list of unique borrowed book IDs using Collectors.toList().

## Database Schema Diagram

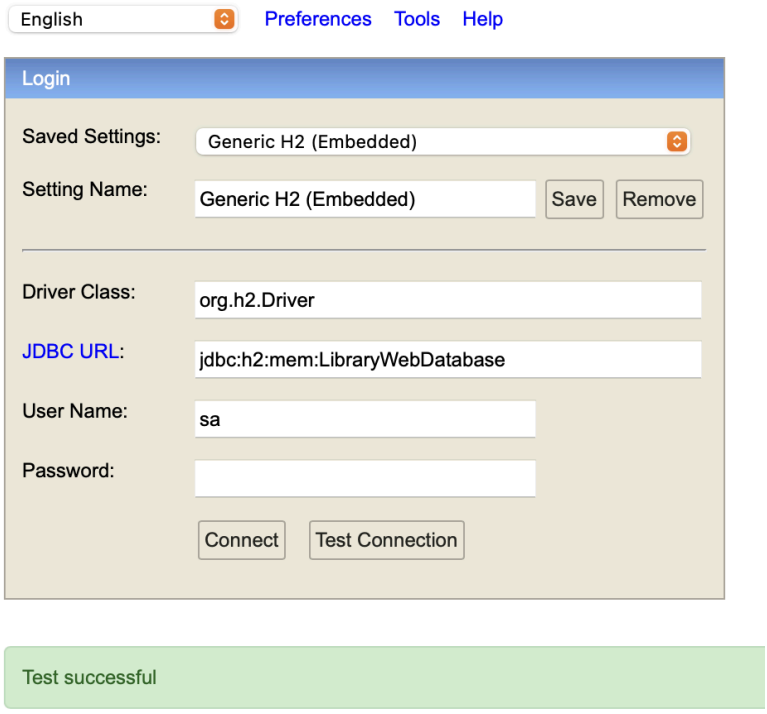
- To add the in-memory h2 database, *application.properties* is modified accordingly:

```
spring.application.name=Library-Web-System
spring.h2.console.enabled=true
spring.datasource.url=jdbc:h2:mem:LibraryWebDatabase
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
```

7. spring.jpa.show-sql=true
8. spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect
9. spring.jpa.hibernate.ddl-auto=update

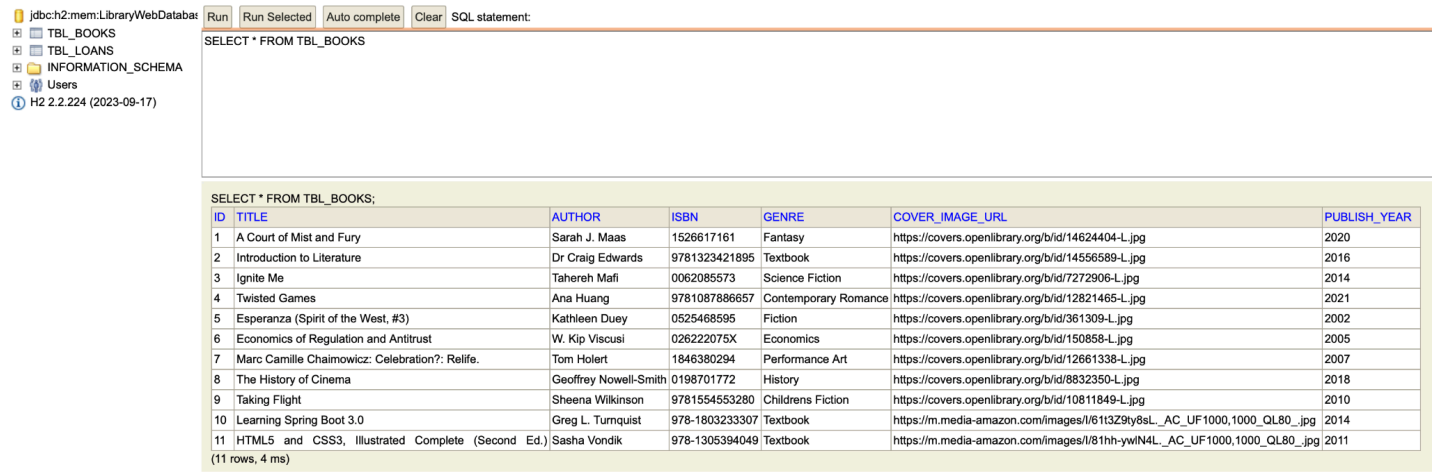
In line 3, the database *LibraryWebDatabase* is created. In line 6, the password is set to default (i.e., empty). According to line 9, tables will be generated automatically when new data is added.

- After running *LibraryWebSystem1Application.java* as a Spring Boot App, we can use the url http://localhost:8080/h2-console to access the H2 database. After testing the connection, "Test Successful" should be displayed as shown below.

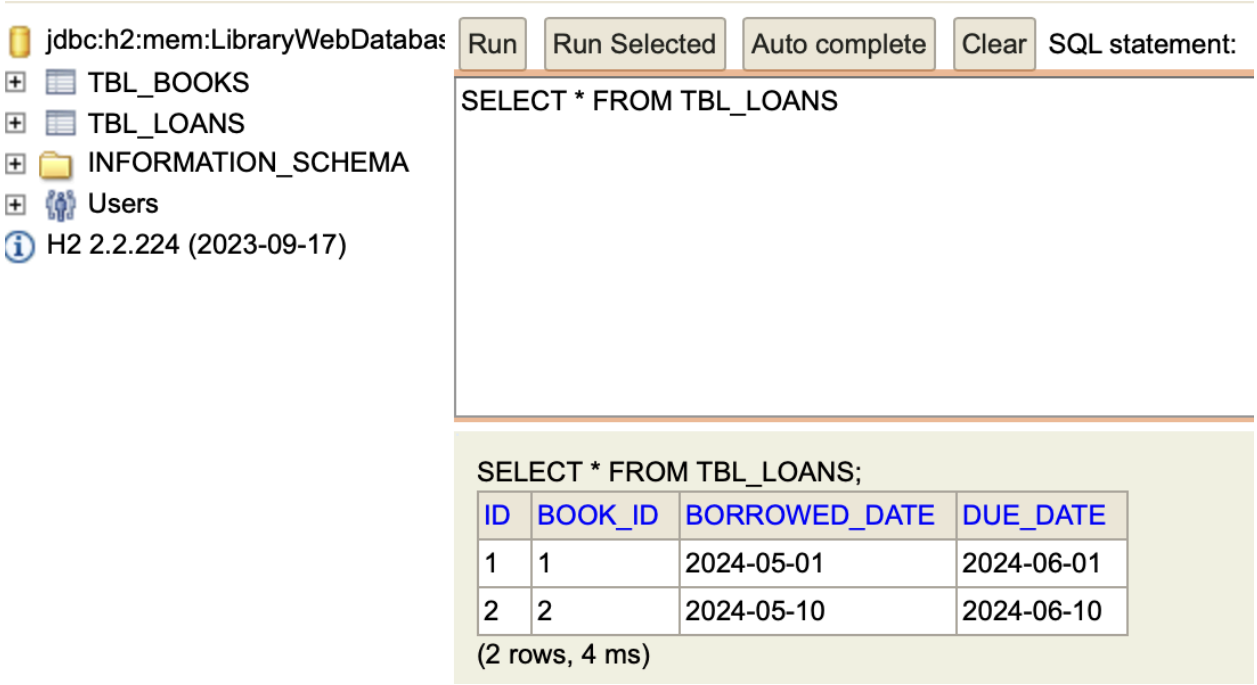



Database name is LibraryWebDatabase. After the successful connection, the database will be connected.

- The data is stored in two tables, TBL_BOOKS & TBL_LOANS.

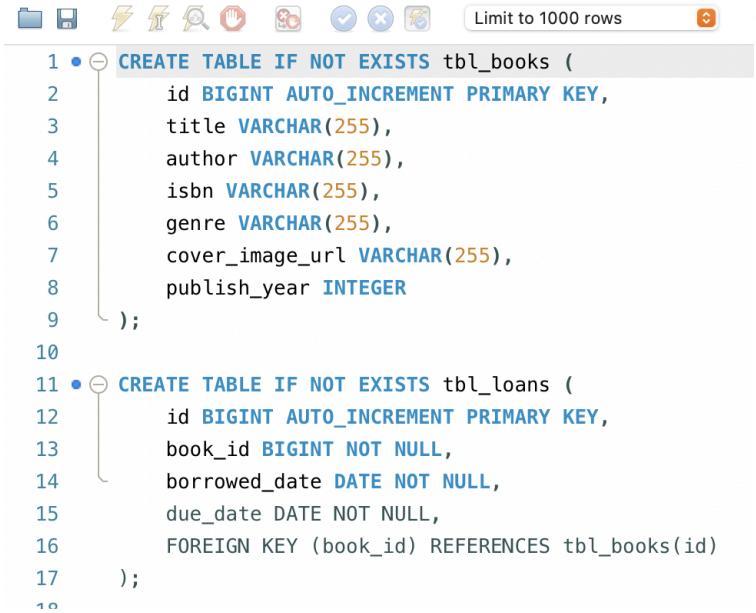



As shown above, the TBL_BOOKS table stores the book details.



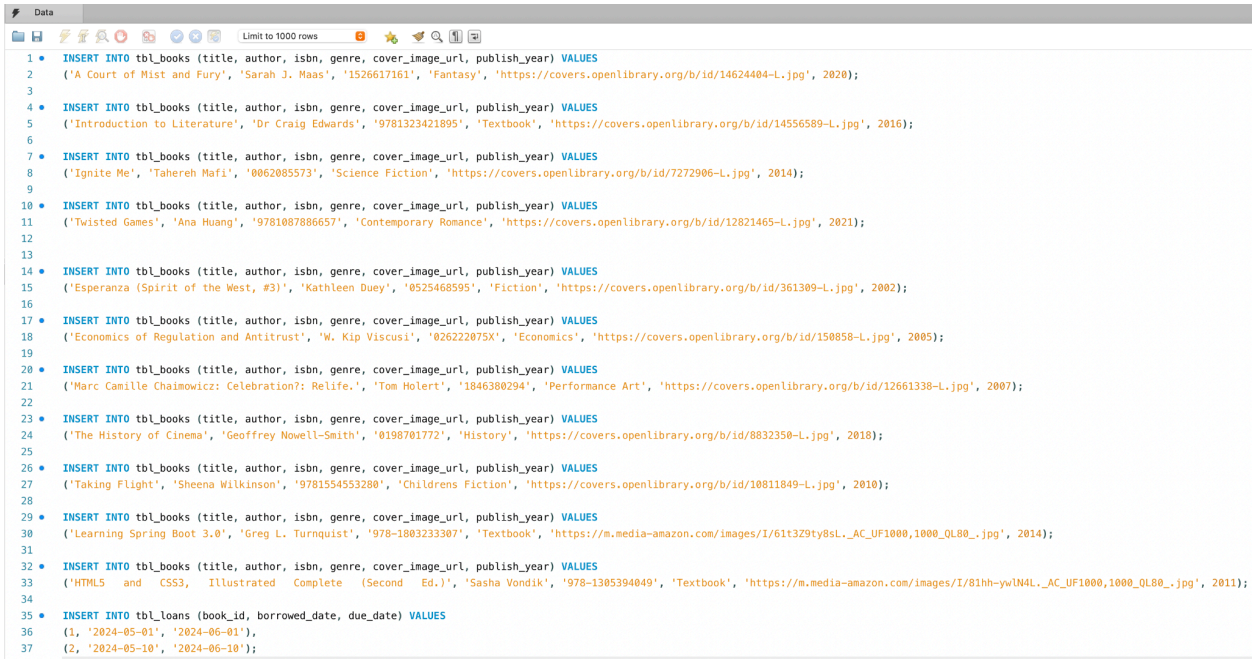

The TBL_LOANS table stores the book id of the borrowed book and the date the book was borrowed and the due date.

The tables are joined on BOOK_ID as shown in the figure below in Schema.sql.



```sql
CREATE TABLE IF NOT EXISTS tbl_books (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255),
    author VARCHAR(255),
    isbn VARCHAR(255),
    genre VARCHAR(255),
    cover_image_url VARCHAR(255),
    publish_year INTEGER
);

CREATE TABLE IF NOT EXISTS tbl_loans (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    book_id BIGINT NOT NULL,
    borrowed_date DATE NOT NULL,
    due_date DATE NOT NULL,
    FOREIGN KEY (book_id) REFERENCES tbl_books(id)
);
```

The sample data was stored in the database in Data.sql as shown below:



```sql
INSERT INTO tbl_books (title, author, isbn, genre, cover_image_url, publish_year) VALUES
('A Court of Mist and Fury', 'Sarah J. Maas', '1526617161', 'Fantasy', 'https://covers.openlibrary.org/b/id/14624404-L.jpg', 2020);

INSERT INTO tbl_books (title, author, isbn, genre, cover_image_url, publish_year) VALUES
('Introduction to Literature', 'Dr Craig Edwards', '9781323421895', 'Textbook', 'https://covers.openlibrary.org/b/id/14556589-L.jpg', 2016);

INSERT INTO tbl_books (title, author, isbn, genre, cover_image_url, publish_year) VALUES
('Ignite Me', 'Tahereh Mafi', '0062085573', 'Science Fiction', 'https://covers.openlibrary.org/b/id/7272906-L.jpg', 2014);

INSERT INTO tbl_books (title, author, isbn, genre, cover_image_url, publish_year) VALUES
('Twisted Games', 'Ana Huang', '9781087886657', 'Contemporary Romance', 'https://covers.openlibrary.org/b/id/12821465-L.jpg', 2021);


INSERT INTO tbl_books (title, author, isbn, genre, cover_image_url, publish_year) VALUES
('Esperanza (Spirit of the West, #3)', 'Kathleen Duey', '0525468595', 'Fiction', 'https://covers.openlibrary.org/b/id/361309-L.jpg', 2002);

INSERT INTO tbl_books (title, author, isbn, genre, cover_image_url, publish_year) VALUES
('Economics of Regulation and Antitrust', 'W. Kip Viscusi', '026222075X', 'Economics', 'https://covers.openlibrary.org/b/id/150858-L.jpg', 2005);

INSERT INTO tbl_books (title, author, isbn, genre, cover_image_url, publish_year) VALUES
('Marc Camille Chaimowicz: Celebration?: Relife.', 'Tom Holert', '1846380294', 'Performance Art', 'https://covers.openlibrary.org/b/id/12661338-L.jpg', 2007);

INSERT INTO tbl_books (title, author, isbn, genre, cover_image_url, publish_year) VALUES
('The History of Cinema', 'Geoffrey Nowell-Smith', '0198701772', 'History', 'https://covers.openlibrary.org/b/id/8832350-L.jpg', 2018);

INSERT INTO tbl_books (title, author, isbn, genre, cover_image_url, publish_year) VALUES
('Taking Flight', 'Sheena Wilkinson', '9781554553280', 'Childrens Fiction', 'https://covers.openlibrary.org/b/id/10811849-L.jpg', 2010);

INSERT INTO tbl_books (title, author, isbn, genre, cover_image_url, publish_year) VALUES
('Learning Spring Boot 3.0', 'Greg L. Turnquist', '978-1803233307', 'Textbook', 'https://m.media-amazon.com/images/I/61t3Z9ty8sL._AC_UF1000,1000_QL80_.jpg', 2014);

INSERT INTO tbl_books (title, author, isbn, genre, cover_image_url, publish_year) VALUES
('HTML5 and CSS3, Illustrated Complete (Second Ed.)', 'Sasha Vondik', '978-1305394049', 'Textbook', 'https://m.media-amazon.com/images/I/81hh-ywlN4L._AC_UF1000,1000_QL80_.jpg', 2011);

INSERT INTO tbl_loans (book_id, borrowed_date, due_date) VALUES
(1, '2024-05-01', '2024-06-01'),
(2, '2024-05-10', '2024-06-10');
```

All the sample data was retrieved from [https://openlibrary.org/](https://openlibrary.org/)

- Functionalities
- **Book Controller**

The functions are retrieved from Book Service and the CRUD operations can be performed on the database according to the functions below. The additional search function is also added.

```java
package com.example.demo.Controllers;

import org.springframework.beans.factory.annotation.Autowired;


@RestController
@RequestMapping("/api/books")
public class BookController {

    @Autowired
    private BookService bookService;

    @GetMapping
    public List<Book> getAllBooks() {
        return bookService.getAllBooks();
    }

    @GetMapping("/{id}")
    public Book getBookById(@PathVariable Long id) {
        return bookService.getBookById(id);
    }


    @PostMapping("/add")
    public ResponseEntity<Book> createBook(@RequestBody Book book) {
        Book savedBook = bookService.saveBook(book);
        if (savedBook != null) {
            return ResponseEntity.ok(savedBook);
        } else {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
        }
    }


    @PutMapping("/edit/{id}")
    public Book updateBook(@PathVariable Long id, @RequestBody Book book) {
        book.setId(id);
        return bookService.saveBook(book);
    }

    @DeleteMapping("/delete/{id}")
    public void deleteBook(@PathVariable Long id) {
        bookService.deleteBook(id);
    }

    @GetMapping("/search")
    public List<Book> searchBooks(@RequestParam String title) {
        return bookService.searchBooksByTitle(title);
    }
}
```

- **Book Entity**

Here we have the constructor for book object and again, we map the two tables via book id and we implement getters and setters. isbn was changed to string as integer variable did not have as much capacity.

```java
@Entity
@Table(name = "tbl_books")
public class Book {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;


    private String title;
    private String author;
    private String isbn;  // Changed to String
    private String genre;
    private int publish_year;

    @OneToMany(mappedBy = "book", cascade = CascadeType.ALL)
    private List<Loan> loans;

    @Column(name = "cover_image_url")
    private String coverImageUrl;

    // Getters and setters
```

- **DTO.java**

The Data Transfer object is used to fetch and store the data from both tables (tbl_books & tbl_loans) and store them all in the same object in order to form a collection. The purpose of this is explained below in Home Controller.

- **Home Controller**

The Home Controller manages the necessary redirections to the proper html file. It also adds the DTO collection as an attribute to the model and redirects user to borrow.html in case the user wants to view the borrowed books.

```
@Controller
public class HomeController {

    @Autowired
    private LoanService loanService;

    @GetMapping("/")
    public String mainPage() {
        return "index"; //Redirects to index.html (home page)
    }

    @GetMapping("/user")
    public String userPage() {
        return "user"; //Redirects to user.html
    }

    @GetMapping("/admin")
    public String adminPage() {
        return "admin"; //Redirects to admin.html
    }

    @GetMapping("/add")
    public String addPage() {
        return "add";  //Redirects to add.html
    }


    @GetMapping("/edit")
    public String editPage() {
        return "edit"; //Redirects to edit.html
    }

    @GetMapping("/showBorrow") //Adds the DTO collection as an attribute to the model and redirects to borrow.html
    public String borrowPage(Model model) {
        List<DTO> borrowedBooks = loanService.getBorrowedBooks();
        model.addAttribute("borrowedBooks", borrowedBooks);
        return "borrow";
    }
```

- Book Service

Book Service implements the necessary functions to interact with the database.

```
@Service
public class BookService {

    @Autowired
    private BookRepository bookRepository;

    public List<Book> getAllBooks() {
        return bookRepository.findAll();
    }

    public Book getBookById(Long id) {
        return bookRepository.findById(id).orElse(null);
    }

    public Book saveBook(Book book) {
        return bookRepository.save(book);
    }

    public void deleteBook(Long id) {
        bookRepository.deleteById(id);
    }

    public List<Book> searchBooksByTitle(String title) {
        return bookRepository.findByTitleContainingIgnoreCase(title);
    }
}
```