

In this tutorial, we will learn how to build REST APIs for a Library Management System Project using Spring Boot, Spring Data JPA (Hibernate), and MySQL database.

We'll create a simple version of the Library Management System with basic functionalities:

- Create a new User
- Fetch all the Users
- Add new book
- Fetch all the books
- Fetch specific book
- Delete a book
- Borrow a book
- Return a book

1. Set up a Spring Boot project

Let's launch Spring Initializr and fill up the following project details:

Project: Maven Project (or Gradle)

Language: Java

Packaging: Jar

Java version: 17

Dependencies: Spring Web, Spring Data JPA, MySQL Driver, and Lombok

Download, extract the project, and import to your favorite IDE.

2. Configure the MySQL database

Let's open the `src/main/resources/application.properties` file and add the MySQL configuration properties:

```
spring.datasource.url=jdbc:mysql://localhost:3306/banking_app
spring.datasource.username=root
spring.datasource.password=Mysql@123
spring.jpa.hibernate.ddl-auto=update
```

Make sure that you change the MySQL database username and password as per your MySQL installation on your machine.

The `spring.jpa.hibernate.ddl-auto=update` line ensures that tables and columns get automatically created or updated based on your JPA entities.

3. Create JPA Entities

User

Let's create a User JPA entity and add the following code to it:

```

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
@Entity
public class User {

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

```

Book

Let's create a Book JPA entity and add the following code to it:

```

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
@Entity
public class Book {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String title;
    private String author;
    private boolean borrowed;
    @ManyToOne
    @JoinColumn(name = "user_id")
    private User borrowedBy;
}

```

We added a relationship between the Book and User entities to track which user has borrowed a book.

4. Create Spring Data JPA Repositories

Next, let's create Spring Data JPA repositories to get CRUD methods to perform CRUD database operations on Book and User entities.

BookRepository

```

import net.javaguides.bankingapp.entity.Book;

```

```
import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Long> {
}
```

UserRepository

```
import net.javaguides.bankingapp.entity.User;
import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {
}
```

6. Create a Service Layer

BookService

```
import net.javaguides.bankingapp.entity.Book;
import net.javaguides.bankingapp.entity.User;
import net.javaguides.bankingapp.repository.BookRepository;
import net.javaguides.bankingapp.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
```

```
import java.util.List;
```

```
@Service
public class BookService {

    @Autowired
    private BookRepository bookRepository;

    @Autowired
    private UserRepository userRepository;

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

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

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

```

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

public Book borrowBook(Long bookId, Long userId) {
    Book book = findById(bookId);
    User user = userRepository.findById(userId).orElse(null);

    if (book != null && !book.isBorrowed() && user != null) {
        book.setBorrowedBy(user);
        book.setBorrowed(true);
        return save(book);
    }
    // Handle errors (e.g., book not found, book already borrowed, user not found)
    return null;
}

public Book returnBook(Long bookId) {
    Book book = findById(bookId);
    if (book != null && book.isBorrowed()) {
        book.setBorrowedBy(null);
        book.setBorrowed(false);
        return save(book);
    }
    // Handle errors (e.g., book not found, book not borrowed)
    return null;
}
}

```

UserService

```

import net.javaguides.bankingapp.entity.User;
import net.javaguides.bankingapp.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class UserService {

    @Autowired
    private UserRepository userRepository;

    public List<User> findAll() {

```

```

        return userRepository.findAll();
    }

    public User save(User user) {
        return userRepository.save(user);
    }
}

```

7. Controller Layer

UserController

Let's create a UserController class and expose a couple of REST APIs to save and fetch all users.

```

import net.javaguides.bankingapp.entity.User;
import net.javaguides.bankingapp.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

```

```

import java.util.List;

```

```

@RestController
@RequestMapping("/api/users")
public class UserController {
    @Autowired
    private UserService userService;

    @GetMapping
    public List<User> getAllUsers() {
        return userService.findAll();
    }

    @PostMapping
    public User addUser(@RequestBody User user) {
        return userService.save(user);
    }
}

```

BookController

Let's create BookController class and expose REST APIs to save, fetch, fetch all, delete, borrow, and return books.

```

import net.javaguides.bankingapp.entity.Book;
import net.javaguides.bankingapp.service.BookService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

```

```

import java.util.List;

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

    @Autowired
    private BookService bookService;

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

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

    @PostMapping
    public Book addBook(@RequestBody Book book) {
        return bookService.save(book);
    }

    @PutMapping("/{id}")
    public Book updateBook(@PathVariable Long id, @RequestBody Book book) {
        // Additional logic to ensure you're updating the correct book
        return bookService.save(book);
    }

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

    @PostMapping("/{bookId}/borrow/{userId}")
    public ResponseEntity<Book> borrowBook(@PathVariable Long bookId, @PathVariable
Long userId) {
        Book borrowedBook = bookService.borrowBook(bookId, userId);
        if (borrowedBook != null) {
            return ResponseEntity.ok(borrowedBook);
        } else {
            return ResponseEntity.badRequest().build(); // or a more descriptive error response
        }
    }
}

```

```

@PostMapping("/{bookId}/return")
public ResponseEntity<Book> returnBook(@PathVariable Long bookId) {
    Book returnedBook = bookService.returnBook(bookId);
    if (returnedBook != null) {
        return ResponseEntity.ok(returnedBook);
    } else {
        return ResponseEntity.badRequest().build(); // or a more descriptive error response
    }
}
}
}

```

Run and Test the Spring Boot Application

Navigate to the main application class (with `@SpringBootApplication` annotation) and run it as a Java application.

Add a new user:

The screenshot shows a REST client interface. At the top, the method is set to **POST** and the URL is `http://localhost:8080/api/users`. A **Send** button is visible. Below the URL bar, there are tabs for Params, Authorization, Headers (8), **Body**, Pre-request Script, Tests, and Settings. Under the **Body** tab, the format is set to **JSON**. The request body is a JSON object: `{ "name": "Ramesh Fadataare" }`. Below the request, the response is shown in **JSON** format: `{ "id": 1, "name": "Ramesh Fadataare" }`. The status bar at the bottom indicates a **200 OK** response with a response time of **385 ms** and a body size of **197 B**. There are also options to **Save as Example** and a search icon.

Fetch all users:

HTTP Method: GET

GET http://localhost:8080/api/users Send

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL **JSON** Beautify

```
1 {
2   "name": "Umesh Fadatare"
3 }
4
```

Body Cookies Headers (5) Test Results 200 OK 171 ms 232 B Save as Example

Pretty Raw Preview Visualize **JSON** Copy Search

```
1 [
2   {
3     "id": 1,
4     "name": "Ramesh Fadatare"
5   },
6   {
7     "id": 2,
8     "name": "Umesh Fadatare"
9   }
10 ]
```

Add a new book:

HTTP Method: POST

URL: <http://localhost:8080/api/books>

POST http://localhost:8080/api/books Send

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL **JSON** Beautify

```
1 {
2   "title": "The Great Gatsby",
3   "author": "F. Scott Fitzgerald",
4   "borrowed": false
5 }
6
```

Body Cookies Headers (5) Test Results 200 OK 21 ms 265 B Save as Example

Pretty Raw Preview Visualize **JSON** Copy Search

```
1 {
2   "id": 1,
3   "title": "The Great Gatsby",
4   "author": "F. Scott Fitzgerald",
5   "borrowed": false,
6   "borrowedBy": null
7 }
```

Fetch all books:

HTTP Method: GET

URL: <http://localhost:8080/api/books>

GET http://localhost:8080/api/books

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

Body Cookies Headers (5) Test Results 200 OK 19 ms 358 B Save as Example

```
1 {
2   {
3     "id": 1,
4     "title": "The Great Gatsby",
5     "author": "F. Scott Fitzgerald",
6     "borrowed": false,
7     "borrowedBy": null
8   },
9   {
10    "id": 2,
11    "title": "Core Java",
12    "author": "Ramesh Fadatare",
13    "borrowed": false,
14    "borrowedBy": null
15  }
```

Fetch a specific book:

HTTP Method: GET

URL: `http://localhost:8080/api/books/{id}` (Replace {id} with the book's ID.)

GET http://localhost:8080/api/books/1

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

Body Cookies Headers (5) Test Results 200 OK 30 ms 265 B Save as Example

```
1 {
2   "id": 1,
3   "title": "The Great Gatsby",
4   "author": "F. Scott Fitzgerald",
5   "borrowed": false,
6   "borrowedBy": null
7 }
```

Update a book:

HTTP Method: PUT

URL: `http://localhost:8080/api/books/{id}` (Replace {id} with the book's ID.)

PUT ▼ http://localhost:8080/api/books/2 Send ▼

Params Authorization Headers (8) **Body** ● Pre-request Script Tests Settings Cookies

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL **JSON** ▼ Beautify

```

1  {
2  ... "title": "Core Java",
3  ... "author": "Ramesh Fadatare",
4  ... "borrowed": true
5  }
6

```

Body Cookies Headers (5) Test Results 200 OK 16 ms 253 B Save as Example ⋮

Pretty Raw Preview Visualize **JSON** ▼ ⇌

```

1  {
2  "id": 3,
3  "title": "Core Java",
4  "author": "Ramesh Fadatare",
5  "borrowed": true,
6  "borrowedBy": null
7  }

```

Delete a book:

HTTP Method: DELETE

URL: <http://localhost:8080/api/books/{id}> (Replace {id} with the book's ID.)

[HTTP](#) Blogging / **New Request** Save ▼ ✎ 💬

DELETE ▼ http://localhost:8080/api/books/1 Send ▼

Params Authorization Headers (8) **Body** ● Pre-request Script Tests Settings Cookies

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL **JSON** ▼ Beautify

```

1  {
2  ... "title": "Core Java",
3  ... "author": "Ramesh Fadatare",
4  ... "borrowed": true
5  }
6

```

Body Cookies Headers (4) Test Results 200 OK 34 ms 123 B Save as Example ⋮

Pretty Raw Preview Visualize **Text** ▼ ⇌

```

1

```

Borrow a book:

HTTP Method: POST

URL: <http://localhost:8080/api/books/{bookId}/borrow/{userId}>

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:8080/api/books/2/borrow/1
- Body:** A JSON object representing a book borrowing record:

```
1 {
2   "id": 2,
3   "title": "Core Java",
4   "author": "Ramesh Fadatare",
5   "borrowed": true,
6   "borrowedBy": {
7     "id": 1,
8     "name": "Ramesh Fadatare"
9   }
10 }
```
- Response:** 200 OK, 23 ms, 282 B

Return a book:

HTTP Method: POST

URL: http://localhost:8080/api/books/{bookId}/return (Replace {bookId} with the book's ID.)

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:8080/api/books/2/return
- Body:** A JSON object representing a book return record:

```
1 {
2   "id": 2,
3   "title": "Core Java",
4   "author": "Ramesh Fadatare",
5   "borrowed": false,
6   "borrowedBy": null
7 }
```
- Response:** 200 OK, 26 ms, 254 B