

LIBRARY PROJECT

Library Management System

Description: A basic app to manage books in a library, where users can add, view, update, and delete book records (e.g., title, author, ISBN, and publication year).

Learning Objectives:

- **Spring Boot:** Build REST endpoints and use Spring Data JPA for database operations.
- **PostgreSQL:** Store book data in a single table or extend to include authors in a separate table.
- **React:** Create a UI with a book list, search functionality, and forms for adding/editing books.
- **Key Concepts:** Query methods in Spring Data JPA (e.g., `findByTitleContaining`), React routing, and filtering data on the frontend.

Implementation Steps:

- **Backend:**
 - Create a `Book` entity with fields like `id`, `title`, `author`, `isbn`, and `publicationYear`.
 - Set up a `BookRepository` with custom query methods (e.g., `findByTitleContaining`).
 - Create a `BookController` with endpoints like `/api/books` and `/api/books/search`.
- **Frontend:**
 - Use React Router for navigation between book list and add/edit pages.
 - Implement a search bar to filter books by title or author using API calls.
 - Use Axios to handle CRUD operations.
- **Database:**
 - Create a `librarydb` database in PostgreSQL.
 - Let Hibernate manage the `books` table.

Why It's Suitable: This project is ideal for teaching students how to implement search functionality and work with query methods. It's simple but can be extended with features like categories or borrowing status.

START

Install PgAdmin

Install PostgreSQL

Create Database librarydb

Learning Points:

LIBRARY PROJECT

- Students learn to set up a PostgreSQL database and understand basic schema design.
- Using Hibernate to manage tables introduces them to ORM (Object-Relational Mapping).

```
CREATE TABLE books (  
    id SERIAL PRIMARY KEY,  
    title VARCHAR(255) NOT NULL,  
    author VARCHAR(255) NOT NULL,  
    isbn VARCHAR(13) UNIQUE NOT NULL,  
    publication_year INTEGER  
);
```

Application.properties

```
spring.datasource.url=jdbc:postgresql://localhost:5432/librarydb  
spring.datasource.username=postgres  
spring.datasource.password=root  
spring.jpa.hibernate.ddl-auto=validate  
spring.jpa.show-sql=true  
spring.jpa.properties.hib
```

BOOK JPA Entity

```
package com.tca.library.jpa;
```

```
import java.io.Serializable;  
import java.math.BigInteger;
```

```
import jakarta.persistence.Entity;  
import jakarta.persistence.GeneratedValue;  
import jakarta.persistence.GenerationType;  
import jakarta.persistence.Id;  
import lombok.Data;
```

```
@Entity
```

```
@Data
```

```
public class Books implements Serializable {
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Integer id;
```

LIBRARY PROJECT

```
        private String title;
        private String author;
        private String isbn;
        private Integer publicationYear;

        public void setId(Integer id2) {
            // TODO Auto-generated method stub

        }
    }
```

BOOK REPOSITORY

```
package com.tca.library.repository;
import com.tca.library.jpa.Books;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.List;
public interface BookRepository extends JpaRepository<Books, Integer> {
    List<Books> findByTitleContainingIgnoreCase(String title);
}
```

CONTROLLER

```
package com.tca.library.controller;
import com.tca.library.jpa.Books;
import com.tca.library.repository.BookRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.util.List;
import java.util.Optional;
@RestController
@RequestMapping("/api/books")
public class BookController {
    @Autowired
    private BookRepository bookRepository;
    // Get all books
    @GetMapping
    public List<Books> getAllBooks() {
        return bookRepository.findAll();
    }
    // Get book by ID
    @GetMapping("/{id}")
    public ResponseEntity<Books> getBookById(@PathVariable Integer id) {
```

LIBRARY PROJECT

```
Optional<Books> book = bookRepository.findById(id);
return book.map(ResponseEntity::ok)
    .orElseGet(() -> ResponseEntity.notFound().build());
}
// Create a new book
@PostMapping
public Books createBook(@RequestBody Books book) {
    return bookRepository.save(book);
}
// Update a book
@PutMapping("/{id}")
public ResponseEntity<Books> updateBook(@PathVariable Integer id, @RequestBody Books
updatedBook) {
    Optional<Books> book = bookRepository.findById(id);
    if (book.isPresent()) {
        updatedBook.setId(id);
        return ResponseEntity.ok(bookRepository.save(updatedBook));
    }
    return ResponseEntity.notFound().build();
}
// Delete a book
@DeleteMapping("/{id}")
public ResponseEntity<Void> deleteBook(@PathVariable Integer id) {
    if (bookRepository.existsById(id)) {
        bookRepository.deleteById(id);
        return ResponseEntity.ok().build();
    }
    return ResponseEntity.notFound().build();
}
// Search books by title
@GetMapping("/search")
public List<Books> searchBooks(@RequestParam String title) {
    return bookRepository.findByTitleContainingIgnoreCase(title);
}
}
```

Test the Backend with Postman:

- Run the Spring Boot application in STS (right-click project > **Run As > Spring Boot App**).
- Open Postman and test the following endpoints:
 - **GET** `http://localhost:8080/api/books`: Retrieve all books.
 - **POST** `http://localhost:8080/api/books`:
 - json

```
{
  "title": "The Great Gatsby",
```

LIBRARY PROJECT

```
"author": "F. Scott Fitzgerald",  
"isbn": "9780743273565",  
"publicationYear": 1925
```

```
}
```

- **GET** `http://localhost:8080/api/books/1`: Get book with ID 1.
- **PUT** `http://localhost:8080/api/books/1`:
- `json`

```
{  
  "title": "The Great Gatsby (Updated)",  
  "author": "F. Scott Fitzgerald",  
  "isbn": "9780743273565",  
  "publicationYear": 1925
```

```
}
```

- **DELETE** `http://localhost:8080/api/books/1`: Delete book with ID 1.
- **GET** `http://localhost:8080/api/books/search?title=Gatsby`: Search for books with "Gatsby" in the title.
- Verify database changes in PostgreSQL:
 - `sql`
 - `SELECT * FROM books;`

BACKEND

Let's build the **Library Management System** step-by-step, focusing on a beginner-friendly approach for a boot camp. We'll start with the database setup in PostgreSQL, move to the backend with Spring Boot (using Spring Tool Suite), and then create the React frontend. We'll use GitHub for version control and Postman to test API endpoints. Each section includes detailed steps, code snippets, and explanations to ensure students learn key concepts like database operations, REST APIs, and frontend-backend integration.

LIBRARY PROJECT

Step 1: Database Creation (PostgreSQL)

Objective: Set up a PostgreSQL database and create a `librarydb` database to store book records.

Steps

- **Install PostgreSQL:**
 - Download and install PostgreSQL (<https://www.postgresql.org/download/>) if not already installed.
 - Use a GUI tool like pgAdmin or DBeaver for easier management (optional but helpful for beginners).
 - Alternatively, use Docker:
 - `bash`
 - `docker run -d -p 5432:5432 --name postgres -e POSTGRES_PASSWORD=yourpassword postgres`
- **Create the Database:**
 - Connect to PostgreSQL using `psql` or a GUI tool.
 - Run the following command to create the database:
 - `sql`
 - `CREATE DATABASE librarydb;`
 - Verify the database exists:
 - `sql`
 - `\l`
- **Plan the Schema:**
 - For simplicity, we'll use a single `books` table with fields: `id`, `title`, `author`, `isbn`, and `publication_year`.
 - We'll let Spring Boot's Hibernate create the table automatically based on the entity, but for reference, the equivalent SQL would be:
 - `sql`

```
CREATE TABLE books (  
  id SERIAL PRIMARY KEY,  
  title VARCHAR(255) NOT NULL,  
  author VARCHAR(255) NOT NULL,  
  isbn VARCHAR(13) UNIQUE NOT NULL,  
  publication_year INTEGER
```

- `);`

Learning Points:

LIBRARY PROJECT

- Students learn to set up a PostgreSQL database and understand basic schema design.
 - Using Hibernate to manage tables introduces them to ORM (Object-Relational Mapping).
-

Step 2: Backend with Spring Boot (Spring Tool Suite)

Objective: Create a Spring Boot application to manage books with REST endpoints, using Spring Data JPA to interact with PostgreSQL.

Prerequisites

- **Spring Tool Suite (STS):** Install from <https://spring.io/tools>.
- **Java:** Ensure JDK 17 or later is installed.
- **Maven:** STS includes Maven, but ensure it's configured.
- **Postman:** Install from <https://www.postman.com/> for testing APIs.

Steps

- **Create a Spring Boot Project:**
 - Open STS and select **File > New > Spring Starter Project**.
 - Configure:
 - **Name:** `library-management-system`
 - **Group:** `com.example`
 - **Artifact:** `library-management`
 - **Package:** `com.example.library`
 - **Dependencies:** Add **Spring Web**, **Spring Data JPA**, **PostgreSQL Driver**, and **Lombok**.
 - Click **Finish** to generate the project.
- **Configure PostgreSQL:**
 - Open
`src/main/resources/application.properties`
and add:
 - `properties`

```
spring.datasource.url=jdbc:postgresql://localhost:5432/librarydb
spring.datasource.username=postgres
spring.datasource.password=yourpassword
```

LIBRARY PROJECT

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect

- Replace `yourpassword` with your PostgreSQL password.
- `ddl-auto=update` lets Hibernate create/update the `books` table based on the entity.
- **Create the Book Entity:**
 - Create a package `com.example.library.entity`.
 - Create a `Book.java` class:

```
package com.tca.library.jpa;
import java.io.Serializable;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import lombok.Data;
@Entity
@Data
public class Books implements Serializable {

    /**
     *
     */
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    private String title;
    private String author;
    private String isbn;
    private Integer publicationYear;

    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getAuthor() {
```


LIBRARY PROJECT

```
        return author;
    }
    public void setAuthor(String author) {
        this.author = author;
    }
    public String getIsbn() {
        return isbn;
    }
    public void setIsbn(String isbn) {
        this.isbn = isbn;
    }
    public Integer getPublicationYear() {
        return publicationYear;
    }
    public void setPublicationYear(Integer publicationYear) {
        this.publicationYear = publicationYear;
    }
}
```

- **Explanation:**
 - `@Entity` marks this as a JPA entity.
 - `@Id` and `@GeneratedValue` define the primary key with auto-increment.
 - `@Data` (Lombok) generates getters, setters, and other utilities.
- **Create the Book Repository:**
 - Create a package `com.example.library.repository`.
 - Create a `BookRepository.java` interface:

```
package com.tca.library.repository;
import com.tca.library.jpa.Books;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import java.util.List;
@Repository
public interface BookRepository extends JpaRepository<Books, Integer> {
    List<Books> findByTitleContainingIgnoreCase(String title);
}
```

- **Explanation:**
 - `JpaRepository` provides CRUD methods (e.g., `save`, `findAll`, `deleteById`).
 - `findByTitleContainingIgnoreCase` is a custom query method for searching books by title (case-insensitive).

LIBRARY PROJECT

- **Create the Book Service**
 - Create a package `com.example.library.service`.
 - Create a `BookService.java` class:

```
package com.tca.library.service;
import com.tca.library.jpa.Books;
import com.tca.library.repository.BookRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import java.util.List;
import java.util.Optional;
@Service
@Transactional
public class BookService {
    private final BookRepository bookRepository;
    @Autowired
    public BookService(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }
    @Transactional(readOnly = true)
    public List<Books> getAllBooks() {
        return bookRepository.findAll();
    }
    @Transactional(readOnly = true)
    public Optional<Books> getBookById(Integer id) {
        return bookRepository.findById(id);
    }
    public Books createBook(Books book) {
        validateBook(book);
        return bookRepository.save(book);
    }
    public Optional<Books> updateBook(Integer id, Books updatedBook) {
        return bookRepository.findById(id)
            .map(existingBook -> {
                updatedBook.setId(id);
                validateBook(updatedBook);
                return bookRepository.save(updatedBook);
            });
    }
    public boolean deleteBook(Integer id) {
        if (bookRepository.existsById(id)) {
            bookRepository.deleteById(id);
            return true;
        }
        return false;
    }
}
```

LIBRARY PROJECT

```
@Transactional(readOnly = true)
public List<Books> searchBooksByTitle(String title) {
    return bookRepository.findByTitleContainingIgnoreCase(title);
}

@Transactional(readOnly = true)
public boolean existsById(Integer id) {
    return bookRepository.existsById(id);
}

private void validateBook(Books book) {
    if (book.getTitle() == null || book.getTitle().trim().isEmpty()) {
        throw new IllegalArgumentException("Book title cannot be null or empty");
    }
    if (book.getAuthor() == null || book.getAuthor().trim().isEmpty()) {
        throw new IllegalArgumentException("Book author cannot be null or empty");
    }
}
}
```

- **Create the Book Controller:**

- Create a package `com.example.library.controller`.
- Create a `BookController.java` class:

```
package com.tca.library.controller;
import com.tca.library.jpa.Books;
import com.tca.library.service.BookService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@CrossOrigin(exposedHeaders = "Authorization")
@RestController
@RequestMapping("/api/books")
public class BookController {
    private final BookService bookService;

    @Autowired
    public BookController(BookService bookService) {
```

LIBRARY PROJECT

```
this.bookService = bookService;
}
// Get all books
@GetMapping
public ResponseEntity<List<Books>> getAllBooks() {
    List<Books> books = bookService.getAllBooks();
    return ResponseEntity.ok(books);
}
// Get book by ID
@GetMapping("/{id}")
public ResponseEntity<Books> getBookById(@PathVariable Integer id) {
    return bookService.getBookById(id)
        .map(book -> ResponseEntity.ok(book))
        .orElse(ResponseEntity.notFound().build());
}
// Create a new book
@PostMapping
public ResponseEntity<Books> createBook(@RequestBody Books book) {
    try {
        Books createdBook = bookService.createBook(book);
        return ResponseEntity.status(HttpStatus.CREATED).body(createdBook);
    } catch (IllegalArgumentException e) {
        return ResponseEntity.badRequest().build();
    }
}
// Update a book
@PutMapping("/{id}")
public ResponseEntity<Books> updateBook(@PathVariable Integer id, @RequestBody Books
updatedBook) {
    try {
        return bookService.updateBook(id, updatedBook)
            .map(book -> ResponseEntity.ok(book))
            .orElse(ResponseEntity.notFound().build());
    } catch (IllegalArgumentException e) {
        return ResponseEntity.badRequest().build();
    }
}
// Delete a book
@DeleteMapping("/{id}")
public ResponseEntity<Void> deleteBook(@PathVariable Integer id) {
    if (bookService.deleteBook(id)) {
        return ResponseEntity.ok().build();
    }
    return ResponseEntity.notFound().build();
}
// Search books by title
```

LIBRARY PROJECT

```
@GetMapping("/search")
public ResponseEntity<List<Books>> searchBooks(@RequestParam String title) {
    if (title == null || title.trim().isEmpty()) {
        return ResponseEntity.badRequest().build();
    }
    List<Books> books = bookService.searchBooksByTitle(title);
    return ResponseEntity.ok(books);
}
```

- **Explanation:**
 - @RestController and @RequestMapping define this as a REST controller with a base path /api/books.
 - Endpoints handle CRUD operations and title-based search.
 - ResponseEntity is used for proper HTTP responses (e.g., 200 OK, 404 Not Found).
- **Test the Backend with Postman:**
 - Run the Spring Boot application in STS (right-click project > **Run As > Spring Boot App**).
 - Open Postman and test the following endpoints:
 - **GET** http://localhost:8080/api/books: Retrieve all books.
 - **POST** http://localhost:8080/api/books:
 - json

```
{
  "title": "The Great Gatsby",
  "author": "F. Scott Fitzgerald",
  "isbn": "9780743273565",
  "publicationYear": 1925
```

- }
- **GET** http://localhost:8080/api/books/1: Get book with ID 1.
- **PUT** http://localhost:8080/api/books/1:
- json

```
{
  "title": "The Great Gatsby (Updated)",
  "author": "F. Scott Fitzgerald",
  "isbn": "9780743273565",
  "publicationYear": 1925
```

LIBRARY PROJECT

- }
- **DELETE** `http://localhost:8080/api/books/1`: Delete book with ID 1.
- **GET** `http://localhost:8080/api/books/search?title=Gatsby`: Search for books with "Gatsby" in the title.
- Verify database changes in PostgreSQL:
 - `sql`
 - `SELECT * FROM books;`
- **Set Up GitHub Repository:**
 - Create a new repository on GitHub (e.g., `library-management-system`).
 - Initialize Git in the project folder:
 - `bash`

`git init`

`git add .`

`git commit -m "Initial commit: Spring Boot backend"`

`git remote add origin https://github.com/yourusername/library-management-system.git`

- `git push -u origin main`
- Encourage students to commit changes frequently (e.g., after each major step).

Learning Points:

- Students learn to create entities, repositories, and REST controllers.
- They practice JPA query methods and RESTful API design.
- Testing with Postman reinforces understanding of HTTP methods and JSON payloads.
- Git usage introduces version control basics.

FRONTEND

This frontend will interact with the existing Spring Boot backend (running at `http://localhost:8080/api/books`) to manage books (view, add, edit, search, delete) without login functionality. We'll use **React Router** for navigation, **Axios** for API calls,

LIBRARY PROJECT

GitHub for version control, and **Postman** to test endpoints. The setup is tailored for a boot camp, ensuring simplicity and clear learning outcomes.

Project Overview

- **Functionality:**
 - Display a list of books with title, author, ISBN, and publication year.
 - Search books by title.
 - Add new books via a form.
 - Edit existing books.
 - Delete books.
- **Tools:**
 - Vite for the React project setup.
 - Tailwind CSS for styling.
 - Axios for HTTP requests.
 - React Router for navigation.
 - Git for version control.
- **Assumptions:**
 - The Spring Boot backend with endpoints (`/api/books`, `/api/books/{id}`, `/api/books/search`) is running at `http://localhost:8080`.
 - The PostgreSQL database (`librarydb`) is set up.
 - CORS is configured in the backend to allow requests from `http://localhost:5173` (Vite's default port):

```
java
```

@Configuration

```
public class CorsConfig implements WebMvcConfigurer {  
    @Override  
    public void addCorsMappings(CorsRegistry registry) {  
        registry.addMapping("/**")  
            .allowedOrigins("http://localhost:5173")  
            .allowedMethods("GET", "POST", "PUT", "DELETE")  
            .allowCredentials(true);  
    }  
}
```

LIBRARY PROJECT

}

- }

Step 1: Set Up the Vite React Project

Objective: Create a React project with Vite and configure Tailwind CSS.

Prerequisites

- **Node.js:** Install from <https://nodejs.org/> (version 18 or later recommended).
- **Code Editor:** VS Code or any preferred editor.
- **Git:** Installed for version control.
- **Postman:** For API testing (already set up from backend).

Steps

- **Create a Vite Project:**
 - Open a terminal and run:
 - `bash`

```
npm create vite@latest library-frontend -- --template react
cd library-frontend
```

- `npm install`
 - This sets up a React project with Vite, which is faster and more modern than Create React App.
- **Install Dependencies:**
 - Install Axios for API calls, React Router for navigation, and Tailwind CSS:
 - `bash`
 - `npm install axios react-router-dom tailwindcss postcss autoprefixer`
 - Or this `npm install -D @tailwindcss/postcss`
 - Initialize Tailwind CSS:

LIBRARY PROJECT

- `bash`
- `npx tailwindcss init -p`
- **Configure Tailwind CSS:**
 - Update `tailwind.config.js` to include all project files:
 - `Js`
 - Tailwind config should be `jsx` not `js`

```
/** @type {import('tailwindcss').Config} */
export default {
  content: [
    './index.html',
    './src/**/*.{js,ts,jsx,tsx}',
  ],
  theme: {
    extend: {},
  },
  plugins: [],
```

- `}`
- Replace `src/index.css` with:
- `css`

```
@tailwind base;
@tailwind components;
```

- `@tailwind utilities;`
- This sets up Tailwind's directives for base styles, components, and utilities.
- **Test the Setup:**
 - Run the development server:
 - `bash`
 - `npm run dev`
 - Open `http://localhost:5173` (Vite's default port) to verify the default Vite + React page.
- **Initialize Git:**

LIBRARY PROJECT

- Initialize a Git repository and connect to GitHub:
- `bash`

`git init`

`git add .`

- `git commit -m "Initial commit: Vite React project with Tailwind"`
- Create a new GitHub repository (e.g., `library-frontend`) and link it:
- `bash`

`git remote add origin https://github.com/yourusername/library-frontend.git`

- `git push -u origin main`

Learning Points:

- Students learn to set up a modern React project with Vite.
 - Configuring Tailwind introduces utility-first CSS.
 - Git initialization reinforces version control basics.
-

Step 2: Create the Main App Component

Objective: Set up `App.jsx` with React Router for navigation between the book list and add/edit forms.

Steps

- **Update** `src/App.jsx`:
 - Replace the default `App.jsx` with:
 - `jsx`

LIBRARY PROJECT

```
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import BookList from './components/BookList';
import BookForm from './components/BookForm';
```

```
function App() {
  return (
    <Router>
      <div className="max-w-4xl mx-auto p-4">
        <h1 className="text-3xl font-bold mb-6 text-center">
          Library Management System
        </h1>
        <Routes>
          <Route path="/" element={<BookList />} />
          <Route path="/add" element={<BookForm />} />
          <Route path="/edit/:id" element={<BookForm />} />
        </Routes>
      </div>
    </Router>
  );
}
```

- export default App;
- **Explanation:**
 - Uses `BrowserRouter` and `Routes` to define three routes: `/` (book list), `/add` (add book form), and `/edit/:id` (edit book form).
 - Tailwind classes (`max-w-4xl`, `mx-auto`, `p-4`, etc.) center the content and style the header.
- **Commit Changes:**
 - Save and commit:
 - `bash`

```
git add src/App.jsx
```

```
git commit -m "Add App component with routing"
```

- `git push origin main`

LIBRARY PROJECT

Learning Points:

- Students learn React Router for SPA navigation.
 - They practice Tailwind CSS for layout and typography.
-

Step 3: Create the BookList Component

Objective: Build a component to display books, include a search bar, and provide options to add, edit, or delete books.

Steps

- **Create** `src/components/BookList.jsx`:
 - Create a `components` folder under `src` and add `BookList.jsx`:

BOOKLIST COMPONENT

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { Link } from 'react-router-dom';

const BookList = () => {
  // ----- State -----
  const [books, setBooks] = useState([]);
  const [search, setSearch] = useState('');
  const [showModal, setShowModal] = useState(false);
  const [bookToDelete, setBookToDelete] = useState(null);

  // ----- Fetch Books on Mount -----
  useEffect(() => {
    fetchBooks();
  }, []);

  // ----- Fetch All Books -----
  const fetchBooks = async () => {
    try {
```

LIBRARY PROJECT

```
const response = await
axios.get('http://localhost:8080/api/books');
setBooks(response.data);
} catch (error) {
console.error('Error fetching books:', error);
}
};

// ----- Handle Search -----
const handleSearch = async (e) => {
e.preventDefault();
try {
if (search.trim()) {
const response = await axios.get(
`http://localhost:8080/api/books/search?title=${search}`
);
setBooks(response.data);
} else {
fetchBooks();
}
} catch (error) {
console.error('Error searching books:', error);
}
};

// ----- Delete Book (with Modal) -----
const confirmDelete = (id) => {
setBookToDelete(id);
setShowModal(true);
};

const handleDelete = async () => {
try {
await
axios.delete(`http://localhost:8080/api/books/${bookToDelete}`);
setShowModal(false);
setBookToDelete(null);
fetchBooks();
} catch (error) {
console.error('Error deleting book:', error);
setShowModal(false);
setBookToDelete(null);
}
}
```

LIBRARY PROJECT

```
};

const handleCancel = () => {
  setShowModal(false);
  setBookToDelete(null);
};

// ----- Render UI -----
return (
  <div className="max-w-4xl mx-auto mt-14 bg-white/90 shadow-2xl
rounded-2xl p-8 border border-gray-200 backdrop-blur-md">
    {/* Book List Heading */}
    <h2 className="text-3xl font-bold mb-8 text-gray-800 text-center
tracking-tight flex items-center justify-center gap-2">
      <span role="img" aria-label="books"><img alt="books icon" data-bbox="585 360 605 375"/></span>
      Book List
    </h2>
    {/* Search Form */}
    <form onSubmit={handleSearch} className="mb-8">
      <div className="flex gap-2">
        <input
          type="text"
          className="flex-1 p-3 border border-gray-300 rounded-lg
focus:outline-none focus:ring-2 focus:ring-indigo-400 bg-white
placeholder:text-gray-400 transition"
          placeholder="Search by title"
          value={search}
          onChange={(e) => setSearch(e.target.value)}
        />
        <button
          type="submit"
          className="bg-gradient-to-r from-indigo-500 to-blue-600
text-white px-6 py-2 rounded-lg font-semibold shadow
hover:from-indigo-600 hover:to-blue-700 transition-all"
        >
          Search
        </button>
      </div>
    </form>
    {/* Add Book Button */}
    <Link
      to="/add"
    />
  </div>
);
```

LIBRARY PROJECT

```
        className="inline-block bg-gradient-to-r from-green-500
to-emerald-500 text-white px-6 py-2 rounded-lg font-semibold mb-6
shadow hover:from-green-600 hover:to-emerald-600 transition-all"
    >
        + Add Book
    </Link>
    { /* Book Table */ }
    <div className="overflow-x-auto rounded-xl shadow mt-2">
        <table className="w-full border-collapse bg-white rounded-xl
overflow-hidden">
            <thead>
                <tr className="bg-gray-50">
                    <th className="border-b p-3 text-left font-semibold
text-gray-700">
                        Title
                    </th>
                    <th className="border-b p-3 text-left font-semibold
text-gray-700">
                        Author
                    </th>
                    <th className="border-b p-3 text-left font-semibold
text-gray-700">
                        ISBN
                    </th>
                    <th className="border-b p-3 text-left font-semibold
text-gray-700">
                        Year
                    </th>
                    <th className="border-b p-3 text-left font-semibold
text-gray-700">
                        Actions
                    </th>
                </tr>
            </thead>
            <tbody>
                { /* Show message if no books */ }
                { books.length === 0 ? (
                    <tr>
                        <td colspan="5" className="p-6 text-center
text-gray-400">
                            No books found
                        </td>
                    </tr>
```

LIBRARY PROJECT

```
    ) : (
      // Render each book row
      books.map((book) => (
        <tr key={book.id} className="hover:bg-indigo-50
transition">
          <td className="p-3 border-b">{book.title}</td>
          <td className="p-3 border-b">{book.author}</td>
          <td className="p-3 border-b">{book.isbn}</td>
          <td className="p-3 border-b">{book.publicationYear ||
'-'}</td>
          <td className="p-3 border-b flex gap-2">
            {/* Edit Button */}
            <Link
              to={`/edit/${book.id}`}
              className="bg-gradient-to-r from-indigo-500
to-blue-600 text-white px-4 py-1 rounded-lg font-medium shadow
hover:from-indigo-600 hover:to-blue-700 transition-all"
            >
              Edit
            </Link>
            {/* Delete Button */}
            <button
              onClick={() => confirmDelete(book.id)}
              className="bg-gradient-to-r from-red-500
to-pink-500 text-white px-4 py-1 rounded-lg font-medium shadow
hover:from-red-600 hover:to-pink-600 transition-all"
            >
              Delete
            </button>
          </td>
        </tr>
      ))
    )}
  </tbody>
</table>
</div>

{/* Delete Confirmation Modal */}
{showModal && (
  <div className="fixed inset-0 z-50 flex items-center
justify-center bg-black/30">
    <div className="bg-white rounded-xl shadow-xl p-8 max-w-sm
w-full border border-gray-200">
```


LIBRARY PROJECT

```
        <h3 className="text-xl font-semibold mb-4 text-gray-800
text-center">Confirm Delete</h3>
        <p className="mb-6 text-gray-600 text-center">Are you sure
you want to delete this book?</p>
        <div className="flex justify-center gap-4">
          <button
            onClick={handleDelete}
            className="bg-gradient-to-r from-red-500 to-pink-500
text-white px-6 py-2 rounded-lg font-semibold shadow hover:from-red-600
hover:to-pink-600 transition-all"
          >
            Yes, Delete
          </button>
          <button
            onClick={handleCancel}
            className="bg-gray-100 text-gray-700 px-6 py-2
rounded-lg font-semibold border border-gray-300 shadow
hover:bg-gray-200 transition-all"
          >
            Cancel
          </button>
        </div>
      </div>
    </div>
  )}
</div>
);
};

export default BookList;
```

- **Explanation:**

- Uses `useState` for the book list and search input.
- Uses `useEffect` to fetch books on mount via `/api/books`.
- `handleSearch` calls `/api/books/search` with the title query.
- `deleteBook` sends a DELETE request and refreshes the list.

LIBRARY PROJECT

- Tailwind classes style the table, buttons, and input (`bg-blue-500`, `hover:bg-blue-600`, `border`, **etc.**).
- The table is responsive with `overflow-x-auto` for small screens.
- **Commit Changes:**
 - Save and commit:

```
git add src/components/BookList.jsx
```

```
git commit -m "Add BookList component with list and search"
```

- `git push origin main`

Learning Points:

- Students learn to fetch and display data with Axios and React hooks.
- They practice Tailwind CSS for styling tables and forms.
- The search feature introduces query parameters and conditional rendering.

Step 4: Create the BookForm Component

Objective: Build a form for adding and editing books, handling POST and PUT requests.

Steps

- **Create** `src/components/BookForm.jsx`:
 - Add the following code:

BOOKFORM COMPONENT

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { useParams, useNavigate } from 'react-router-dom';

const BookForm = () => {
  // ----- Hooks and State -----
  const { id } = useParams();
```

LIBRARY PROJECT

```
const navigate = useNavigate();
const [book, setBook] = useState({
  title: '',
  author: '',
  isbn: '',
  publicationYear: ''
});

// ----- Fetch Book Data if Editing -----
useEffect(() => {
  if (id) {
    axios
      .get(`http://localhost:8080/api/books/${id}`)
      .then((response) => {
        setBook(response.data);
      })
      .catch((error) => {
        console.error('Error fetching book:', error);
      });
  }
}, [id]);

// ----- Handle Input Change -----
const handleChange = (e) => {
  setBook({ ...book, [e.target.name]: e.target.value });
};

// ----- Handle Form Submit -----
const handleSubmit = async (e) => {
  e.preventDefault();
  try {
    if (id) {
      await axios.put(`http://localhost:8080/api/books/${id}`, book);
    } else {
      await axios.post('http://localhost:8080/api/books', book);
    }
    navigate('/');
  } catch (error) {
    console.error('Error saving book:', error);
  }
};
```

LIBRARY PROJECT

```
// ----- Render UI -----
return (
  <div className="max-w-lg mx-auto mt-16 bg-white/90 shadow-2xl
rounded-2xl p-10 border border-gray-200 backdrop-blur-md">
    {/* Form Heading */}
    <h2 className="text-3xl font-bold mb-8 text-gray-800 text-center
tracking-tight flex items-center justify-center gap-2">
      <span role="img" aria-label="book"><img alt="book icon" data-bbox="575 225 595 240"/></span>
      {id ? 'Edit Book' : 'Add Book'}
    </h2>
    {/* Book Form */}
    <form onSubmit={handleSubmit} className="space-y-7">
      {/* Title Field */}
      <div>
        <label className="block text-base font-semibold mb-2
text-gray-700">Title</label>
        <input
          type="text"
          name="title"
          value={book.title}
          onChange={handleChange}
          className="w-full p-3 border border-gray-300 rounded-lg
focus:outline-none focus:ring-2 focus:ring-indigo-400 bg-white
placeholder:text-gray-400 transition"
          required
          placeholder="e.g. The Great Gatsby"
        />
      </div>
      {/* Author Field */}
      <div>
        <label className="block text-base font-semibold mb-2
text-gray-700">Author</label>
        <input
          type="text"
          name="author"
          value={book.author}
          onChange={handleChange}
          className="w-full p-3 border border-gray-300 rounded-lg
focus:outline-none focus:ring-2 focus:ring-indigo-400 bg-white
placeholder:text-gray-400 transition"
          required
          placeholder="e.g. F. Scott Fitzgerald"
        />
      </div>
    </form>
  </div>
)
```

LIBRARY PROJECT

```
</div>
{/* ISBN Field */}
<div>
  <label className="block text-base font-semibold mb-2
text-gray-700">ISBN</label>
  <input
    type="text"
    name="isbn"
    value={book.isbn}
    onChange={handleChange}
    className="w-full p-3 border border-gray-300 rounded-lg
focus:outline-none focus:ring-2 focus:ring-indigo-400 bg-white
placeholder:text-gray-400 transition"
    required
    placeholder="e.g. 978-3-16-148410-0"
  />
</div>
{/* Publication Year Field */}
<div>
  <label className="block text-base font-semibold mb-2
text-gray-700">Publication Year</label>
  <input
    type="number"
    name="publicationYear"
    value={book.publicationYear}
    onChange={handleChange}
    className="w-full p-3 border border-gray-300 rounded-lg
focus:outline-none focus:ring-2 focus:ring-indigo-400 bg-white
placeholder:text-gray-400 transition"
    placeholder="e.g. 1925"
  />
</div>
{/* Form Actions */}
<div className="flex gap-4 justify-end pt-4">
  <button
    type="submit"
    className="bg-gradient-to-r from-indigo-500 to-blue-600
text-white px-8 py-3 rounded-lg font-semibold shadow-lg
hover:from-indigo-600 hover:to-blue-700 transition-all duration-200
flex items-center gap-2"
  >
    <span role="img" aria-label="save"><img alt="save icon" data-bbox="618 878 638 893"/></span>
    Save
  </button>
</div>
```

LIBRARY PROJECT

```
        </button>
        <button
          type="button"
          onClick={() => navigate('/') }
          className="bg-white text-indigo-700 px-8 py-3 rounded-lg
font-semibold border border-indigo-200 shadow hover:bg-indigo-50
transition-all duration-200 flex items-center gap-2"
        >
          <span role="img" aria-label="cancel">✖</span>
          Cancel
        </button>
      </div>
    </form>
  </div>
);
};

export default BookForm;
```

- **Explanation:**

- Uses `useParams` to get the book ID for editing.
- Uses `useNavigate` for redirection after saving.
- Fetches book data on mount for editing.
- Handles form input with `useState` and submits via POST (add) or PUT (edit).
- Tailwind classes (`space-y-4`, `w-full`, `focus:ring-2`, etc.) style the form.

- **Commit Changes:**

- Save and commit:
- `bash`

```
git add src/components/BookForm.jsx
```

```
git commit -m "Add BookForm component for add/edit"
```

- `git push origin main`

LIBRARY PROJECT

Learning Points:

- Students learn form handling and controlled inputs.
 - They practice conditional rendering for add/edit modes.
 - Tailwind CSS simplifies responsive form styling.
-

Step 5: Test the Application

Objective: Verify the frontend works with the backend.

Steps

- **Run the Backend:**
 - Ensure the Spring Boot application is running at `http://localhost:8080`.
- **Run the Frontend:**
 - In the `library-frontend` directory, run:
 - `bash`
 - `npm run dev`
 - Open `http://localhost:5173` in a browser.
- **Test Functionality:**
 - **List Books:** Check if the book list loads (empty if no books).
 - **Add a Book:**
 - Click “Add Book”, enter details (e.g., Title: “1984”, Author: “George Orwell”, ISBN: “9780451524935”, Publication Year: 1949), and save.
 - Verify the book appears in the list.
 - **Search Books:**
 - Search for “1984” and confirm only matching books show.
 - Clear the search to display all books.
 - **Edit a Book:**
 - Click “Edit”, update the title (e.g., “1984 (Updated)”), and save.
 - Check the updated book in the list.
 - **Delete a Book:**
 - Click “Delete” and verify the book is removed.

LIBRARY PROJECT

- **Verify with Postman:**

- Test backend endpoints:

- **GET** `http://localhost:8080/api/books`: List all books.
 - **POST** `http://localhost:8080/api/books`:
 - `json`

```
{
```

```
  "title": "Pride and Prejudice",  
  "author": "Jane Austen",  
  "isbn": "9780141439518",  
  "publicationYear": 1813
```

- `}`
 - **GET** `http://localhost:8080/api/books/search?title=1984`:

Verify search.

- Check the database:
 - `sql`
 - `SELECT * FROM books;`

Learning Points:

- Students test full-stack integration.
- They debug issues like CORS or API errors.
- Postman reinforces API testing skills.

Step 6: Finalize GitHub Setup

Objective: Document and push the project to GitHub.

Steps

- **Create a README:**

LIBRARY PROJECT

- Add `README.md` in the project root:

Library Management System Frontend

A React frontend for managing books, built with Vite and Tailwind CSS, integrated with a Spring Boot backend and PostgreSQL.

Setup

1. Ensure the backend is running at `http://localhost:8080`.
2. Install dependencies: `npm install`.
3. Start the app: `npm run dev`.
4. Open `http://localhost:5173` in a browser.

Features

- View a list of books.
- Search books by title.
- Add, edit, and delete books.

Technologies

- React, Vite, Tailwind CSS, Axios, React Router
- Backend: Spring Boot, PostgreSQL
- Tools: Git, Postman

API Endpoints

- GET `/api/books`: List all books
 - POST `/api/books`: Create a book
 - GET `/api/books/{id}`: Get a book
 - PUT `/api/books/{id}`: Update a book
 - DELETE `/api/books/{id}`: Delete a book
- - GET `/api/books/search?title={title}`: Search books by title
 - Commit:
 - `bash`

```
git add README.md
```

```
git commit -m "Add README for frontend"
```

- `git push origin main`
- **Final Push:**
 - Commit all changes:

LIBRARY PROJECT

- `bash`

`git add .`

`git commit -m "Complete React frontend with Vite and Tailwind"`

- `git push origin main`

Learning Points:

- Students learn to write clear documentation.
 - Regular commits reinforce Git workflows.
-
-

Learning Outcomes

Students will:

- **React:** Use hooks (`useState`, `useEffect`, `useParams`, `useNavigate`) and handle forms.
- **Vite:** Experience a modern build tool with fast development.
- **Tailwind CSS:** Apply utility-first styling for responsive design.
- **API Integration:** Make CRUD requests with Axios.
- **Routing:** Implement navigation with React Router.
- **Git:** Manage code with Git and GitHub.

Resources:


- Vite: <https://vite.dev/guide/>
- Tailwind CSS: <https://tailwindcss.com/docs/installation>
- React Router: <https://reactrouter.com/en/main>
- Axios: <https://axios-http.com/docs/intro>

Next Steps:

LIBRARY PROJECT

- Run and test the app thoroughly.
- Share the GitHub repository for collaboration.

BOOKLIST

 **Book List**

Search

+ Add Book

Title	Author	ISBN	Year	Actions
The Great Gatsby	F. Scott Fitzgerald	9780743273565	1925	<button>Edit</button> <button>Delete</button>
48 Laws of Power	Robert Greene	123456789	2002	<button>Edit</button> <button>Delete</button>

LIBRARY PROJECT

Add / Edit Book Form

Add Book

Title

e.g. The Great Gatsby

Author

e.g. F. Scott Fitzgerald

ISBN

e.g. 978-3-16-148410-0

Publication Year

e.g. 1925



Save



Cancel