Library Application Preview

RavenClaw

January 2024

Contents

1 Introduction

This document provides a preview of the Library Application, a web-based platform designed to give users access to their personal library. By logging in, users can manage their book lists.

2 Prerequisites

Before you start working with the library application, make sure you have the following tools and technologies installed:

- **IntelliJ IDEA:** The integrated development environment for Java.
- **Visual Studio Code (VSC):** A lightweight code editor.
- **Angular CLI:** Command-line interface for Angular development.
- **Spring Framework:** A comprehensive framework for Java development.
- **PostgreSQL:** A powerful, open-source relational database system.

The specified version for the Spring Boot Starter Parent is 3.2.1, and the Java version is 17.

It is organized into categories, so its structure is like this:

```
Library
├── src
│   ├── main
│   │   ├── java
│   │   │   └── com.generation.italy.library
│   │   │       ├── api.restcontroller
│   │   │       ├── configuration
│   │   │       ├── dtos
│   │   │       ├── model
│   │   │       └── LibraryApplication.java
│   │   └── resources
│   │       ├── application.properties
│   │       └── library.sql
│   └── test
└── pom.xml
```

2.1 Backend

For the backend development, we opted for Java due to its powerful and versatile nature. Java's extensive libraries, platform independence, and strong community support make it an ideal choice for building robust and scalable applications.

2.1.1 Programming Language: Java

Java's object-oriented programming paradigm allows us to design modular and maintainable code. The language's strict typing system enhances code reliability, reducing the chances of runtime errors. Additionally, Java's wide adoption in enterprise-level applications ensures a vast pool of resources and community expertise.

2.1.2 Framework: Spring

We chose the Spring framework to expedite the development process and ensure a well-organized codebase. Spring provides a comprehensive ecosystem, offering features such as dependency injection, aspect-oriented programming, and robust support for building enterprise-level applications.

The use of Spring interfaces facilitates seamless integration with the SQL side of our application. Leveraging Spring's data access abstractions, we can easily connect and interact with databases, ensuring efficient data management and retrieval.

2.1.3 Design Pattern: MVC (Model-View-Controller)

To enhance the maintainability and scalability of our backend architecture, we adopted the Model-View-Controller (MVC) design pattern. This separation of concerns allows us to isolate the application logic (Model), user interface (View), and user input handling (Controller).

- **Model:** The model represents the data and business logic of the application. It encapsulates the interaction with the database and provides a structured way to manage and manipulate data.

- **View:** The view is responsible for presenting data to the user and receiving user input. It remains decoupled from the underlying business logic, ensuring flexibility in adapting to different presentation requirements.

- **Controller:** Acting as an intermediary between the model and view, the controller handles user input, processes requests, and updates the model accordingly. This separation of concerns simplifies code maintenance and promotes code reusability.

By adhering to the MVC pattern, our backend architecture promotes code organization, maintainability, and scalability, laying a solid foundation for the development of a robust and efficient application.

2.2 Frontend

As said before, the Frontend side of this application is structured in Angular, it's structure contains:

```
/
├── .angular/
├── .vscode/
├── node_modules/
├── src/
│   ├── app/
│   │   ├── components/
│   │   │   ├── author/
│   │   │   ├── author-details/
│   │   │   ├── books/
│   │   │   ├── books.component.
│   │   │   ├── books-card/
│   │   │   ├── books-detail/
│   │   │   ├── footer/
│   │   │   ├── home/
│   │   │   ├── login/
│   │   │   ├── navbar/
│   │   │   ├── search-bar/
│   │   │   └── user/
│   │   ├── app-routing.module.ts
│   │   ├── app.component.css
│   │   ├── app.component.html
│   │   ├── app component.spec.ts
│   │   ├── app.component.ts
│   │   └── app.module.ts
│   ├── model
│   │   └── dtos
│   └── services
└── package.json
```

3 Components

3.1 Navbar Component

The Navbar Component is the header of the Library Web App. It includes a user login button, and a 2 button menu for the user to navigate to different sections of the app, the books and the home button.

3.1.1 Navbar Component Template

The Navbar Component template contains the following elements:

- A login button for users to log in to the app

- A menu for users to navigate to different sections of the app

3.2 Home Component

The Home Component is the landing page of the Library Web App. It displays some quotes and a roadmap of the project.

3.2.1 Home Component Template

The Home Component template contains the following elements:

- A section for displaying quotes
- A roadmap of the project

3.3 Books Component

The Books Component is the page for displaying a list of books in the Library Web App.

3.3.1 Books Component Template

The Books Component template contains the following elements:

- A section for displaying a list of books
- A search bar for searching books

3.4 Books Detail Component

The Books Detail Component is the page for displaying the details of a book in the Library Web App.

3.4.1 Books Detail Component Template

The Books Detail Component template contains the following elements:

- the Book's various Details
- A button for adding the book to the user's library
- A section with the recommended books filtered by genre

3.5 SQL

We created our database using PostgreSQL, and for database management, we utilized PgAdmin4.
As of now, our SQL file is stored in the resources folder within the backend code. This file encompasses five tables: the first for Books, the second for Authors, the third for Genres, the fourth for Clients' Stored Books, and the last for the Login and Authentication service.

4 Contribution

This project was made by:
Marco Suriano: Github: Marcolin97
Fabio Manna: Github: Fabio0crownguard
Karim El Gharabawy: Github
Francesco Moxedano: Github

5 Code Examples

5.1 RestController

Inside the ‘api.restcontroller‘ package, there are various REST controllers that make HTTP calls. For example:
This ‘GET‘ method allows us to retrieve the list of all authors in our database.

```
@GetMapping("/")
public List<Author> getAllAuthors() {
    return libraryService.getAllAuthors();
}
```

5.2 DTOS

The DTOS package contains Data Transfer Objects (DTOs) essential for transmitting object information to the Angular frontend. For example:
The RegisterDto class is a Data Transfer Object (DTO) used to encapsulate information related to user registration.

```
package com.generation.italy.library.dtos;

@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class RegisterDto {
    private String firstname;
    private String lastname;
    private String email;
    private String password;
    private Role role;
}
```