Project Report
Modern Application Development - II

Kabilan (22f2000945)

August 10, 2024

1 Project Overview

This project is a comprehensive web-based library management system that aims to streamline and digitize the interaction between users and the library. The backend of the system is developed using Flask, a lightweight and versatile Python web framework, while the frontend is implemented using Vue.js, a progressive JavaScript framework known for its ease of integration and reactivity. The system provides functionalities such as user registration, book search, book requests, and real-time tracking of issued and returned books.

The application is built with a clear separation of concerns, where the Flask backend handles the core logic, including database interactions, authentication, and API endpoint management. The backend is responsible for processing requests from the frontend, querying the database for relevant information, and sending appropriate responses back to the client. Additionally, the backend supports various scheduled tasks, such as sending daily reminders to users and generating monthly activity reports for the librarian.

On the frontend, Vue.js is used to create a dynamic and user-friendly interface. This frontend framework interacts with the Flask backend via RESTful APIs to display data such as book listings, user profiles, and administrative dashboards. Vue.js components are used to ensure a modular and maintainable codebase, allowing for easier updates and scalability of the system.

To support administrative functions, the system includes an admin panel where librarians can manage book sections, add or update book details, and monitor user activity. The application also generates visual reports, such as pie charts and bar charts, to provide insights into book distribution and user engagement within different sections of the library. These charts are generated using Matplotlib and Seaborn and are cached to improve performance.

Overall, this project demonstrates a robust and scalable architecture, integrating Flask's backend capabilities with Vue.js's frontend dynamism. The system enhances the library experience for both users and administrators by providing an efficient and modernized platform for managing library resources and activities.

2 Frameworks and Libraries Used

2.1 Flask

Flask is a lightweight WSGI web application framework in Python, designed to make getting started quick and easy, with the ability to scale up to complex applications. In this project, Flask serves as the backend framework, handling HTTP requests, routing, and database interactions.

2.2 Vue.js

Vue.js is a progressive JavaScript framework used for building user interfaces. Vue.js is used in this project to create a dynamic and responsive frontend, allowing for a smooth user experience. The framework's reactive data binding and component-based architecture enable developers to build reusable UI components, which enhances the maintainability and scalability of the application.

2.3 Bootstrap

Bootstrap is a popular open-source CSS framework directed at responsive, mobile-first front-end web development. It contains CSS- and JavaScript-based design templates for typography, forms, buttons, navigation, and other interface components. In this project, Bootstrap is utilized to ensure a consistent and visually appealing design across different pages of the application. The framework's pre-built components and utilities help in rapidly developing a professional-looking interface with minimal custom CSS.

2.4 Flask-JWT-Extended

Flask-JWT-Extended is an extension for Flask that adds support for using JSON Web Tokens (JWT) to handle authentication. JWT is a compact and self-contained way to securely transmit information between parties as a JSON object. This project uses Flask-JWT-Extended to manage user authentication, including token generation, storage, and validation. By integrating JWT, the system ensures that only authenticated users can access certain routes and perform actions, enhancing the security of the application.

2.5 Celery

Celery is an asynchronous task queue/job queue based on distributed message passing. It is focused on real-time operation but supports scheduling tasks as well. In this project, Celery is employed to handle background tasks such as sending daily reminders to users and generating monthly activity reports. The integration of Celery allows the application to offload time-consuming tasks from the main request/response cycle, improving performance and user experience.

2.6 Redis

Redis is an in-memory data structure store used as a database, cache, and message broker. It is commonly used for caching and storing session data in web applications. In this project, Redis is used in conjunction with Celery to manage task queues efficiently and handle caching operations, thus speeding up response times and reducing database load.

2.7 Flask-Caching

Flask-Caching is a Flask extension that adds caching support to Flask applications. It supports various caching backends, including Redis. In this project, Flask-Caching is used to store frequently accessed data such as user sessions, API responses, and database query results. By reducing the need to repeatedly fetch the same data, the caching mechanism significantly enhances the application's performance.

2.8 Flask-CORS

Flask-CORS is a Flask extension that handles Cross-Origin Resource Sharing (CORS), making it easy to enable CORS in a Flask application. CORS is crucial when the frontend (Vue.js) and backend (Flask) are hosted on different domains. This project uses Flask-CORS to ensure that the frontend can securely communicate with the backend, allowing seamless integration of services.

2.9 Flask-SQLAlchemy

Flask-SQLAlchemy is an extension for Flask that adds support for SQLAlchemy, a SQL toolkit and Object-Relational Mapping (ORM) library for Python. SQLAlchemy allows developers to interact with the database using Python classes and objects instead of raw SQL queries. In this project, Flask-SQLAlchemy is used to manage database models, queries, and transactions, providing a more Pythonic way to work with relational databases.

2.10 Matplotlib and Seaborn

Matplotlib and Seaborn are powerful Python libraries used for data visualization. Matplotlib provides a broad range of plotting functions to create static, animated, and interactive visualizations, while Seaborn is built on top of Matplotlib and offers a higher-level interface for drawing attractive and informative statistical graphics. These libraries are used in this project to generate charts and graphs that display insights into library usage, such as the distribution of books across sections and user engagement metrics. These visualizations aid the librarian in making data-driven decisions.

2.11 Smtplib

Smtplib is a built-in Python module that defines an SMTP client session object used to send mail to any Internet machine with an SMTP or ESMTP listener daemon. In this project, smtplib is used to send automated emails, including daily reminders to users about visiting or returning books, and monthly reports to the librarian. By automating email communication, the system ensures timely updates and notifications, enhancing user engagement and administrative efficiency.

3 Entity-Relationship Diagram

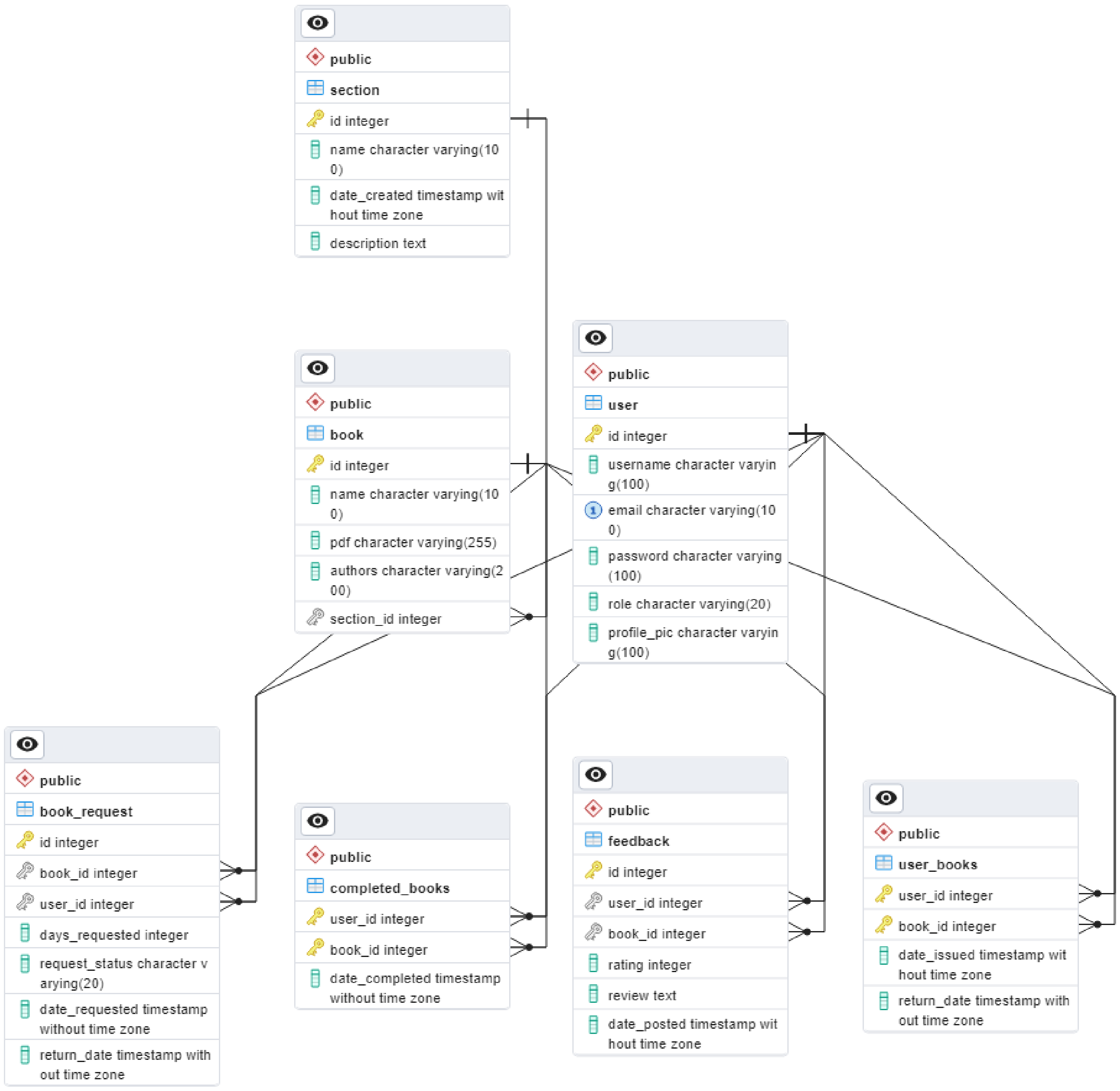
public
section
id integer
name character varying(100)
date_created timestamp without time zone
description text
public
book
id integer
name character varying(100)
pdf character varying(255)
authors character varying(200)
section_id integer
public
user
id integer
username character varying(100)
email character varying(100)
password character varying(100)
role character varying(20)
profile_pic character varying(100)
public
book_request
id integer
book_id integer
user_id integer
days_requested integer
request_status character varying(20)
date_requested timestamp without time zone
return_date timestamp without time zone
public
completed_books
user_id integer
book_id integer
date_completed timestamp without time zone
public
feedback
id integer
user_id integer
book_id integer
rating integer
review text
date_posted timestamp without time zone
public
user_books
user_id integer
book_id integer
date_issued timestamp without time zone
return_date timestamp without time zone

Figure 1: Entity-Relationship Diagram of the Library Management System

4 API Resource Endpoints

This section outlines the key API endpoints provided by the library management system. These endpoints are built using Flask and Flask-RESTful and are designed to handle CRUD operations for sections and books, as well as providing access to generated graphs.

4.1 Section Resource Endpoints

- **GET /api/section/{section_id}**: Retrieves the details of a specific section by its ID.
- **POST /api/section**: Creates a new section. Requires `section_name` and `section_description` as input parameters.
- **PUT /api/section/{section_id}**: Updates an existing section by its ID. Accepts `section_name` and `section_description` as parameters.
- **DELETE /api/section/{section_id}**: Deletes a section by its ID.

4.2 Book Resource Endpoints

- **GET /api/book/{book_id}**: Retrieves the details of a specific book by its ID.
- **PUT /api/book/{book_id}**: Updates an existing book by its ID. Accepts `book_name`, `book_authors`, and `section_id` as parameters.
- **DELETE /api/book/{book_id}**: Deletes a book by its ID.

4.3 Graph Resource Endpoints

- **GET /api/graph/bar_chart**: Retrieves the bar chart visualization generated for administrative insights.
- **GET /api/graph/pie_chart**: Retrieves the pie chart visualization generated for administrative insights.

5 Presentation Video

The presentation video for this project can be accessed via the following link: Video.