EASTERN INTERNATIONAL UNIVERSITY

SCHOOL OF COMPUTING AND INFORMATION TECHNOLOGY

DEPARTMENT OF SOFTWARE ENGINEERING

PROJECT 1 REPORT

LIBRARY WEB APPLICATION

Student(s)

Tran Trung Hau – 2131200004

Hoang Cong Thien – 2131200053

Supervisor(s)

Nguyen Duc Tien

Binh Duong, March 2025

ABSTRACT

The Book Management System is a web platform designed to streamline book inventory management, facilitate user access to book information, and provide administrative control over book and user records. Built with React.js for the frontend, Nest JS for the backend, and MySQL for the database, the system ensures efficiency, scalability, and security.

Key features include user authentication, book searching, and an admin dashboard for book and user administration. While the core functionalities have been successfully implemented, additional features such as book rental and user rating system are still under development and planned for future releases.

This project aims to enhance usability, security, and performance while providing a user-friendly interface for both general users and administrators. The results indicate that the system meets its primary objectives, but further enhancements will continue to improve its effectiveness.

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to Mr. Nguyen Duc Tien, whose guidance and feedback have been invaluable throughout the development of this project. Their insights and expertise have played an important role in refining the system's design and implementation.

I would also like to thank Easten International University for providing the necessary resources and support, as well as my friends for their encouragement and constructive suggestions.

Lastly, I am grateful for the opportunity to work on this project, which has allowed me to expand my knowledge of web development, database management, and system architecture. The experience gained through this work will undoubtedly contribute to my future endeavors in software development.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

No.	Term	Meaning
1	UI	User Interface
2	DOM	Document Object Model
3	ESM	ES modules
4	HMR	Hot module replacement
5	JIT	just-in-time
6	OOP	Object-Oriented Programming
7	FP	Functional Programming
8	FRP	Functional Reactive Programming
9	RDBMS	relational database management system
10	JSON	JavaScript Object Notation
11	JWT	JSON Web Token

CHAPTER 1. INTRODUCTION

1.1. Project Background

A book management system is a web application designed to facilitate the organization, storage, and retrieval of books. This system helps users efficiently manage book collections by providing functionalities such as book registration, categorization, lending, and returning. Libraries, bookstores, and individual collectors often struggle with maintaining accurate records, tracking book availability, and managing user interactions. A digital platform simplifies these processes and ensures data integrity, accessibility, and easy to use. Features of the system:

- **User Management:**
  - Admins can add, update, and delete users.
  - Users can register and log in to the system.
- **Book Inventory Management**
  - Books can be added, categorized, and updated by the admins.
  - Users can search for books based on title and different criteria.
- **Borrowing and Returning**
  - Users can borrow and return books.
  - The system tracks due dates and penalties.
- **Rating System**
  - Users can rate books based on their reading experience.
  - The system provides an average rating for each book.

1.2. Problem Statement

Managing books manually is time-consuming and prone to errors. Traditional methods, such as paper-based or spreadsheet records, often lead to misplaced data, difficulty in tracking borrowed books, and searching inefficient. Additionally, without a management system, users may struggle to access real-time information about book availability and borrowing history. A web book management system addresses these issues by providing an automated, reliable, and user-friendly solution.

1.3. Project objectives

The main objectives of this project are:

- To develop a web system that simplify book management processes.
- To enable users to efficiently search, borrow, and return books.
- To enhance the accessibility of book records through a database.
- To implement an intuitive user interface that simplifies interactions.

1.4. Scope of the Project (in-scope, out-scope)

1.4.1. In-Scope:

- User Interface Development: A fully functional web interface that enables users and administrators to interact with the system efficiently.
- Book Inventory Management: Features for adding, updating, deleting, and managing books in the system.
- User Management: Functionalities for user registration, authentication, role-based access control, and profile management.
- Book Borrowing System: Users can borrow and return books, with due date tracking.
- Rating System: Users can provide ratings for books, and the system calculates an average rating for each book.
- Database Development: A structured database designed to store book details, user accounts, ratings, and book rental transactions.

1.4.2. Out-Scope:

- Advanced Recommendation System: AI on user preferences are not implemented.
- Mobile Application: The system is designed for web platforms only, mobile development is currently out of scope.

1.5. Methodology

The development of this project follows the **Agile methodology**. The project phases include:

- **Requirement Gathering** – Identifying the needs of users and defining system functionalities.
- **System Design** – Creating wireframes, database schemas, and defining technology stack.
- **Development** – Implementing core features using web technologies such as React, Tailwind, Typescript, and a backend framework (Node.js, Nest.js).
- **Deployment** – Deploying the system on a local server.
- **Maintenance and Updates** – Continuously improving the system.

1.6. Report structure

This report is structured as follows:

Chapter 1: **Introduction** – Provides an overview of the project, problem statement, objectives, scope, methodology, and report structure.

Chapter 2: **Related Works** – Reviews existing book management solutions and relevant technologies

Chapter 3: **Application Analysis, Design, and Implementation** – Details system architecture, database design, use cases, activity diagrams, and implementation details.

Chapter 4: **Results and Discussion** – Discusses project outcomes, performance evaluations, and challenges faced.

Chapter 5: **Conclusion and Future Works** – Summarizes findings and discusses potential future enhancements.

CHAPTER 2. RELATED WORKS

2.1. Well-known Library Management Systems.

Several open-source library management systems have been developed to assist libraries in organizing and managing their book inventories, user transactions, and cataloging. Here are some of the popular libraries we have explored and searched during this project:

- **Koha** is a widely used integrated library system (ILS) that offers cataloging, circulation tracking, and reporting functionalities. It is built using Perl, MySQL/MariaDB, and supports Zebra indexing for efficient search functionality.
- **Evergreen** is another scalable library system designed for public and academic libraries, featuring book transaction management, multi-library support, and advanced search capabilities. It is developed using **PostgreSQL**, **OpenSRF framework**, and **AngularJS.**
- **OpenBiblio** is a lightweight system tailored for small libraries, having basic book cataloging and user account management using **PHP** and **MySQL**.

Compared to these systems, our book management system leverages modern technologies such as **React**, **Nest JS**, and **MariaDB** to enhance performance and user experience. Additionally, our system introduces a built-in book rating feature and an improved search mechanism, differentiating it from traditional library management solutions.

2.2. Inventory Management Systems

Book inventory management systems are widely used in libraries and bookstores to keep track of book availability, stock levels, and categorization. Traditional systems often rely on barcode scanning and database management to handle book transactions efficiently. Some well-known systems:

- Evergreen: A scalable book inventory system commonly used in public and academic libraries.
- Koha: An open-source integrated library system used by many public and academic libraries worldwide.
- OpenBiblio: A free library management system that supports small libraries with book cataloging and lending features.

Comparison with Our System:

- Our book inventory management module aims to provide similar functionalities, including book registration, categorization, and real-time tracking. However, our system focuses more on user-friendly web interactions and integration with a rating system to enhance user engagement.

2.3. User Management

User management systems are essential for handling user authentication, role-based access. Existing solution such as Firebase Authentication provide users management services for applications.

Comparison with Our System:

- Our system incorporates a basic user management module where admins can add, update, and delete user accounts, while users can register and log in securely. Unlike third-party solutions, our approach prioritizes direct control over user data and customization flexibility.

2.4. Rating System

Rating systems allow users to provide feedback and evaluate books based on their reading experience. Platforms like Goodreads and Amazon Books use rating systems to influence book recommendations and popularity.

Comparison with Our System:

- Our rating system, while still in development, aims to offer users the ability to rate books and view average ratings. Unlike larger platforms, our implementation focuses on simplicity and integration within a book management environment.

2.5. Technology Review

2.5.1. React

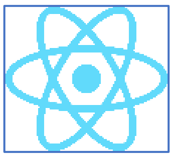


Figure 1. React Logo

React is an open-source JavaScript library for building dynamic user interfaces. Developed by Facebook, it uses a component-based architecture and a virtual DOM for efficient rendering [1]. React's declarative approach simplifies UI development by automatically updating components when data changes. It is widely used for web and mobile apps (via React Native) due to its performance and flexibility [2].

2.5.2. Vite

Figure 2. Vite Logo

Vite is a next-generation frontend build tool that significantly improves development speed. Unlike traditional bundlers like Webpack, Vite leverages native ES modules (ESM) to serve code instantly during development, enabling near-instant hot module replacement (HMR) [3]. It is optimized for modern frameworks like React, Vue, and Svelte, offering fast production builds using Rollup under the hood.

2.5.3. Tailwind CSS.

Figure 3. Tailwind CSS Logo

Tailwind CSS is a utility-first CSS framework that enables rapid UI development by providing low-level utility classes directly in HTML. Unlike traditional CSS frameworks, it avoids pre-designed components, allowing for fully customizable designs without writing custom CSS [4]. Its just-in-time (JIT) compiler optimizes performance by generating only the styles you use.

2.5.4. Font Awesome

Figure 4. Font Awesome Logo

Font Awesome is a popular icon toolkit and library that provides scalable vector icons that can be customized with CSS. It offers both free and premium icons that work in any web project, from simple websites to complex applications [5]. The icons integrate easily with frameworks like React, Vue, and Angular, and can be implemented using web fonts, SVG, or JavaScript.

2.5.5. Nest JS

Figure 5. NestJS Logo

NestJS is a progressive **Node.js framework** for building efficient, scalable server-side applications. It uses **TypeScript** by default and combines elements of **Object-Oriented Programming (OOP)**, **Functional Programming (FP)**, and **Functional Reactive**

Programming (FRP). NestJS provides an out-of-the-box application architecture that allows developers to create highly testable, maintainable, and loosely coupled applications. It leverages **Express.js** (or optionally **Fastify**) under the hood while providing a higher-level abstraction for organizing code through modules, controllers, and providers. [6]

2.5.6. MariaDB

Figure 6. MariaDB Logo

MariaDB is a high-performance, open-source relational database management system (RDBMS) that serves as a community-developed fork of MySQL. Designed for **speed, reliability, and scalability**, MariaDB maintains compatibility with MySQL while introducing **enhanced features, better performance optimizations, and additional storage engines** [7]. It is widely used in web applications, data warehousing, and enterprise-level deployments, offering robust security features and strong community support.

CHAPTER 3. APPLICATION ANALYSIS, DESIGN AND IMPLEMENTATION

3.1. Requirements

The system must fulfill the following requirements:

- **Functional Requirements**
  - Users must be able to register, log in, and manage their accounts.
  - Admins must be able to add, update, delete books and users.
  - Users must be able to borrow and return books.
  - The system support book rating functionality.
- **Non-Functional Requirements**
  - The system must be secure and protect user data.
  - The interface should be user-friendly and responsive.
  - The database should ensure data integrity and quick retrieval.

3.2. System Architecture

The system is built using the following technology stack:

- **Frontend:** React.js with Tailwind CSS for UI styling.
- **Backend:** Node.js with Nest.js for handling API requests.
- **Database:** MySQL for storing structured data.
- **Authentication:** JWT (JSON Web Token) for secure user authentication.

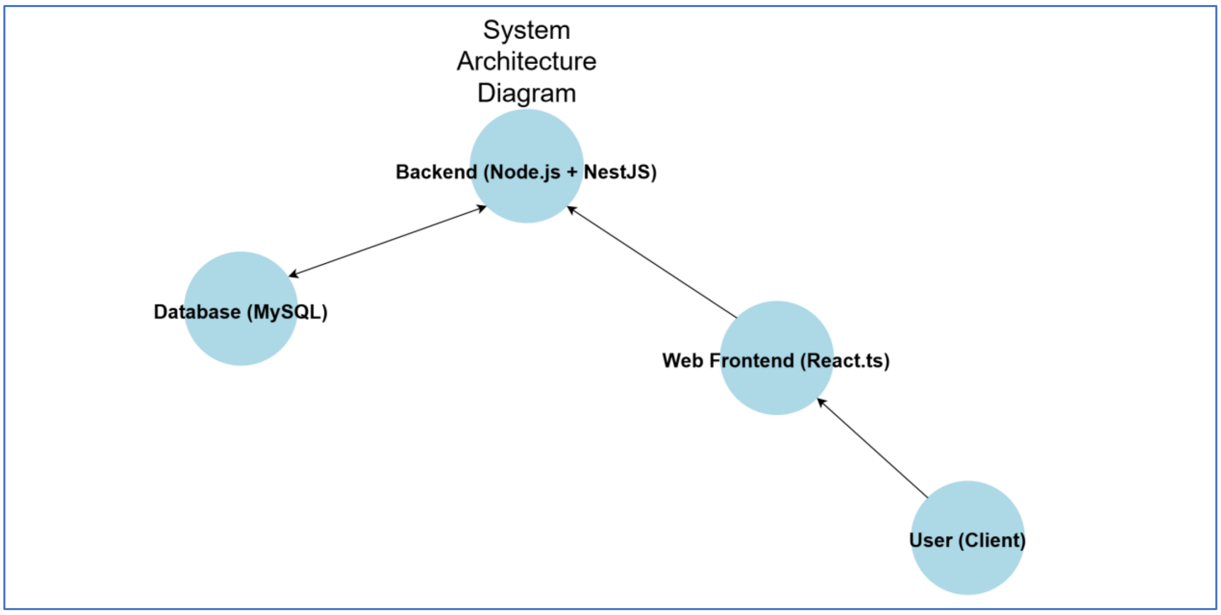


Figure 7. System Architecture Diagram

3.3. System Design

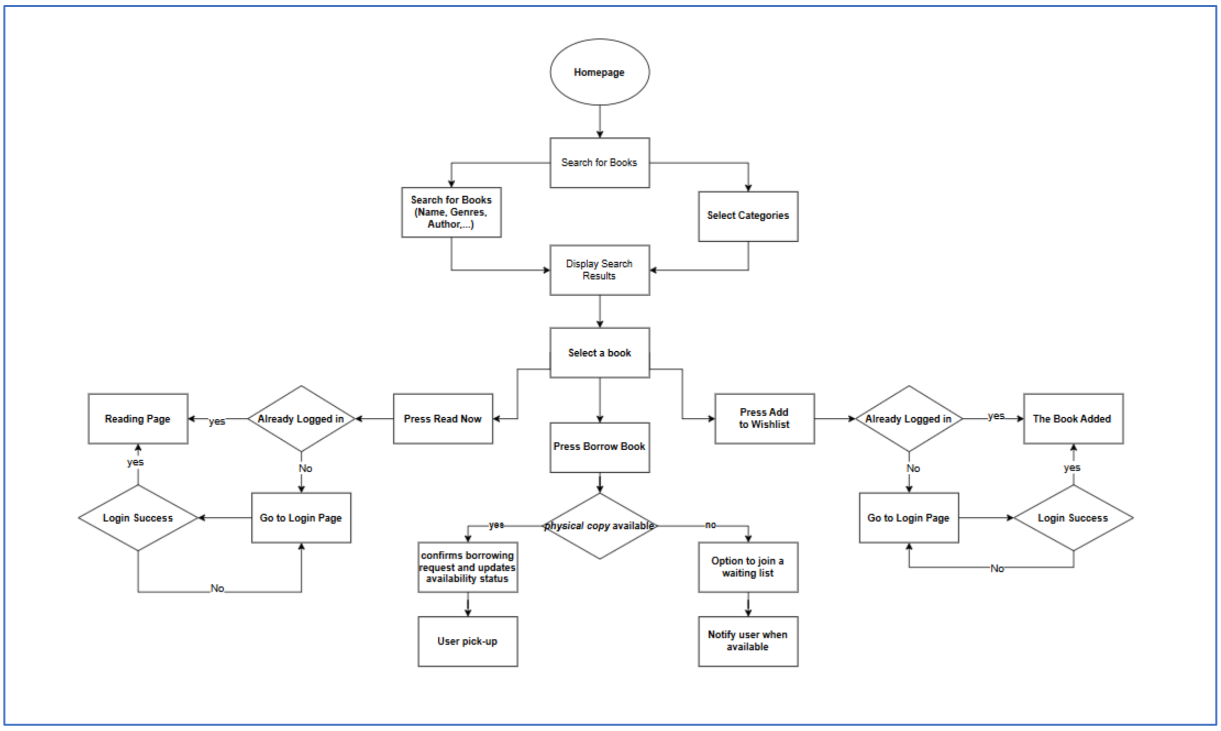


Figure 8. Wireframe

3.3.1. Front-end

The activity diagrams illustrate the workflow of different processes in the **Book Management System**. These diagrams help in understanding the functional flow and interactions within the system.

3.3.1.1. User Book Borrowing Process

The Book Borrowing Process activity diagram demonstrates how a user searches for a book, checks availability, and completes the borrowing process.

Description:

1. The user navigates to the Book Dashboard from the homepage.
2. The user searches for a book by title, author, or genre.
3. The system retrieves matching results and displays them.
4. The user selects a book and chooses the "Borrow" option.
5. The system checks if a physical copy is available:
   - o If available, the system confirms the borrowing request and updates the book's status.
   - o If unavailable, the system prompts the user to join a waiting list.
6. If borrowing is successful, the user can pick up the book from the library.

Key Considerations:

- o Users must be logged in to borrow a book.

- The system ensures that users do not exceed the borrowing limit.
- If the book is unavailable, users have the option to wait for availability.

3.3.1.2. Adding a Book to Wishlist

The **Wishlist Addition Process** allows users to save books for future reference.

Description:

1. The user selects a book and clicks "Add to Wishlist".
2. The system checks if the user is logged in:
    - If logged in, the book is added to the user's wishlist.
    - If not logged in, the system redirects the user to the login page.
3. Once the user logs in, the book is successfully added to the wishlist.

Key Considerations:

- Users must have an account to maintain a wishlist.
- This feature helps users keep track of books they plan to borrow or read later.

3.3.1.3. Reading a Digital Book

For books available in digital format, users can read them directly through the system.

Description:

1. The user selects a digital book and clicks "Read Now".
2. The system checks if the user is logged in:
    - If logged in, the system grants access to the reading page.
    - If not logged in, the user is redirected to the login page.
3. After logging in, the user can start reading the book online.

Key Considerations:

- Only digital books can be read within the system.
- The system may implement restrictions such as a time limit or page limit per session.

These activity diagrams help visualize the process flow, ensuring clarity in system operations.

3.3.2. Database Schema

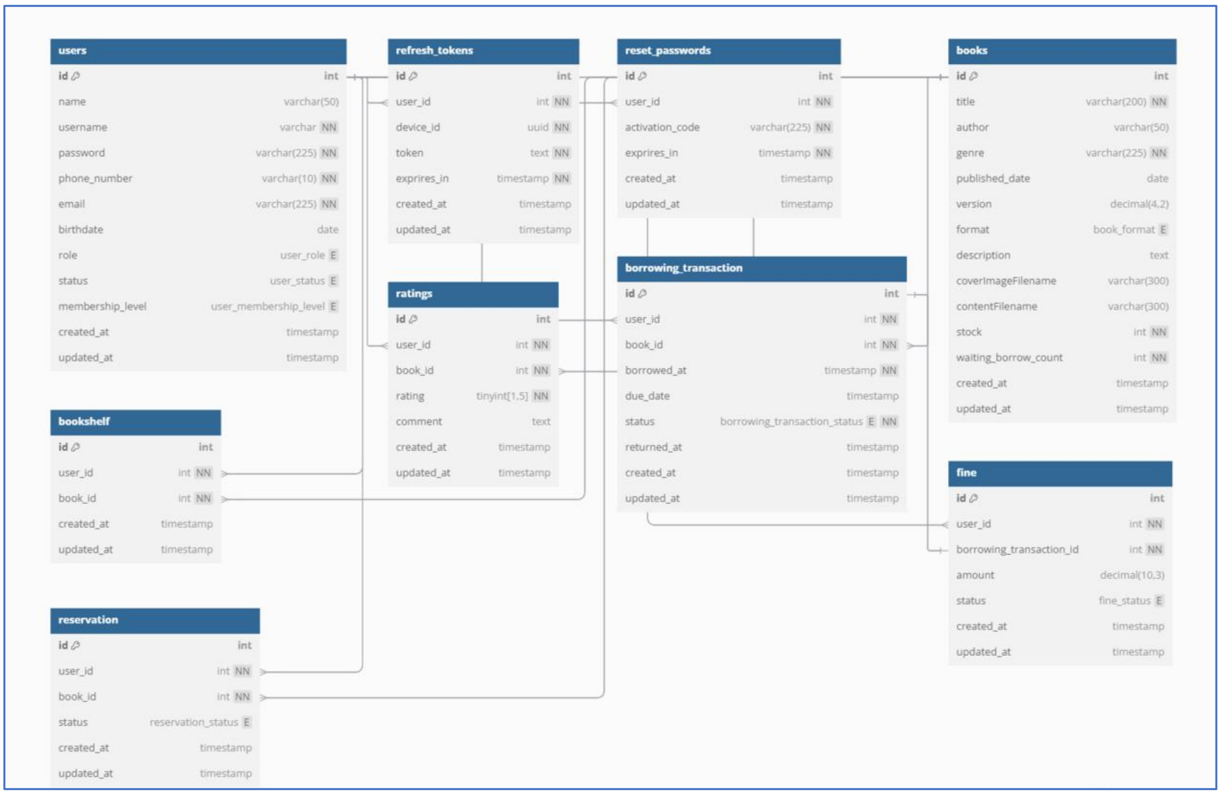


Figure 9. Database Schema

3.4. Use case

3.4.1. Overview

The Book Management System includes two main types of users:

- User: A general user who interacts with the system to search, rent, rate, and manage books.
- Admin: A privileged user responsible for managing book records and user accounts.

The diagram illustrates how these users interact with the system's functionalities

3.4.2. Actor

3.4.2.1. User (Regular member)

- Can search, view, and rent books.
- Can add books to a wishlist and rate them.
- Can edit personal user information.

3.4.2.2. Admin (System administrator)

- Has full control over book and user records.
- Can add, update, and delete books.

CHAPTER 4. Can manage user accounts.

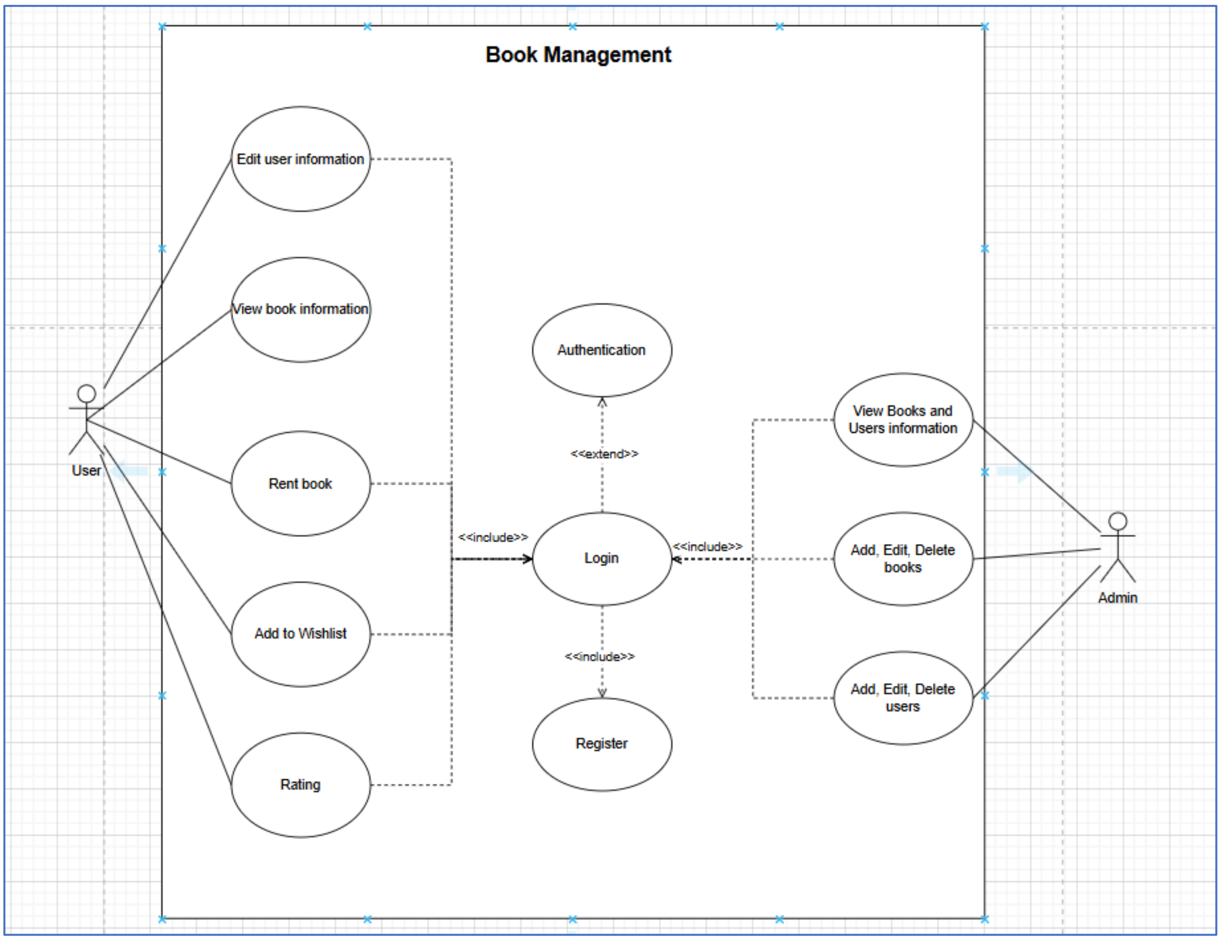


Figure 10. Use Case Diagram

This use case diagram and description outline the key functionalities of the Book Management System, demonstrating how users interact with the platform and the responsibilities of admins. It provides a structured approach for development and ensures clear system requirements.

4.1. Database

Table 1. User

No.	Field Name	Type	Description
1	id	int	Primary key
2	name	varchar(50)	nullable
3	username	varchar(50)	unique, not null
4	password	varchar(225)	not null
5	phoneNumber	varchar(10)	unique, not null
6	email	varchar(225)	unique, not null
7	birthdate	date	nullable

8	role	enum	[“admin”, “user”, “developer”]
9	status	enum	[“active”, “disable”]
10	membershiplevel	enum	[“bronze”, “silver”, “gold”]
11	created_at	timestamp	The time of the record is created
12	udated_at	timestamp	The time of the record is updated

Table 2. Refresh_Token

No.	Field Name	Type	Description
1	id	int	Primary key
2	userId	int	not null
3	deviceId	uuid	not null
4	token	text	not null
5	expriresIn	timestamp	not null
6	created_at	timestamp	The time of the record is created
7	udated_at	timestamp	The time of the record is updated

Table 3. Reset_Password

No.	Field Name	Type	Description
1	id	int	Primary key
2	userId	int	not null
3	activation_code	varchar(225)	not null
4	exprires_in	timestamp	nullable
5	created_at	timestamp	The time of the record is created
6	udated_at	timestamp	The time of the record is updated

Table 4. Book

No.	Field Name	Type	Description
1	id	int	Primary key
2	title	varchar(200)	not null
3	author	varchar(50)	nullable
4	gerne	varchar(225)	not null
5	publishedDate	date	nullable

6	version	decimal	nullable, precision: 4, scale: 2
7	format	enum	not null, [“bản in”, “bản điện tử”]
8	description	text	nullable
9	coverImageFilename	varchar(300)	nullable
10	contentFilename	varchar(300)	nullable
11	stock	int	not null, default = 0
12	waitingBorrowCount	int	not null, default = 0
13	created_at	timestamp	The time of the record is created
14	updated_at	timestamp	The time of the record is updated

Table 5. Bookshelf

No.	Field Name	Type	Description
1	id	int	Primary key
2	userId	int	not null
3	bookId	int	not null
4	created_at	timestamp	The time of the record is created
5	updated_at	timestamp	The time of the record is updated

Table 6. Rating

No.	Field Name	Type	Description
1	id	int	Primary key
2	userId	int	not null
3	bookId	int	not null
4	rating	tinyint	not null, range[1, 5]
5	comment	text	nullable
6	created_at	timestamp	The time of the record is created
7	updated_at	timestamp	The time of the record is updated

Table 7. Borrowing_Transaction

No.	Field Name	Type	Description
1	id	int	Primary key
2	userId	int	not null

3	bookId	int	not null
4	borrowedAt	timestamp	not null
5	dueDate	timestamp	nullable
6	status	enum	not null, [“canceled”, “waiting”, “borrowing”, “returned”, “overdue”, “book missing”]
7	returnedAt	timestamp	nullable
8	created_at	timestamp	The time of the record is created
9	updated_at	timestamp	The time of the record is updated

Table 8. Fine

No.	Field Name	Type	Description
1	id	int	Primary key
2	userId	int	not null
3	borrowingTransactionId	int	not null
4	amount	decimal	nullable, precision: 10, scale: 3
5	status	enum	[“paid”, “unpaid”]
6	created_at	timestamp	The time of the record is created
7	updated_at	timestamp	The time of the record is updated

Table 9. Reservation

No.	Field Name	Type	Description
1	id	int	Primary key
2	userId	int	not null
3	bookId	int	not null
4	status	enum	[“waiting”, “successful”, “canceled”]
5	created_at	timestamp	The time of the record is created
6	updated_at	timestamp	The time of the record is updated

4.2. Implementation Details

4.2.1. Frontend Development

The frontend of the system is built using **React.js**, a popular JavaScript library for building user interfaces. The UI components are styled using **Tailwind CSS**, which provides a flexible and efficient way to design responsive interfaces.

- The **home page** allows users to browse available books, view their borrowing history, and rate books.
- The **authentication system** is implemented using **JWT**, where users log in securely and maintain active sessions.
- **API integration** is handled through **Axios**, allowing the frontend to communicate with the backend efficiently.

4.2.2. Backend Development

The backend is developed using **Node.js with Nest.js**, a progressive framework that simplifies API development and enforces modular architecture. The backend handles user authentication, book management, and transaction processing.

- Authentication: Uses JWT tokens to ensure secure login and session management.
- Book Controller: Allows admins to add, edit, and delete books.
- Borrowing System: Users can borrow books, and the system tracks due dates.
- Rating System: Users can submit ratings, which are stored in the database.

4.2.3. Database Implementation.

The system uses **MySQL** to store book, user, and transaction data. The tables are structured to support relationships between users, books, and borrowing history.

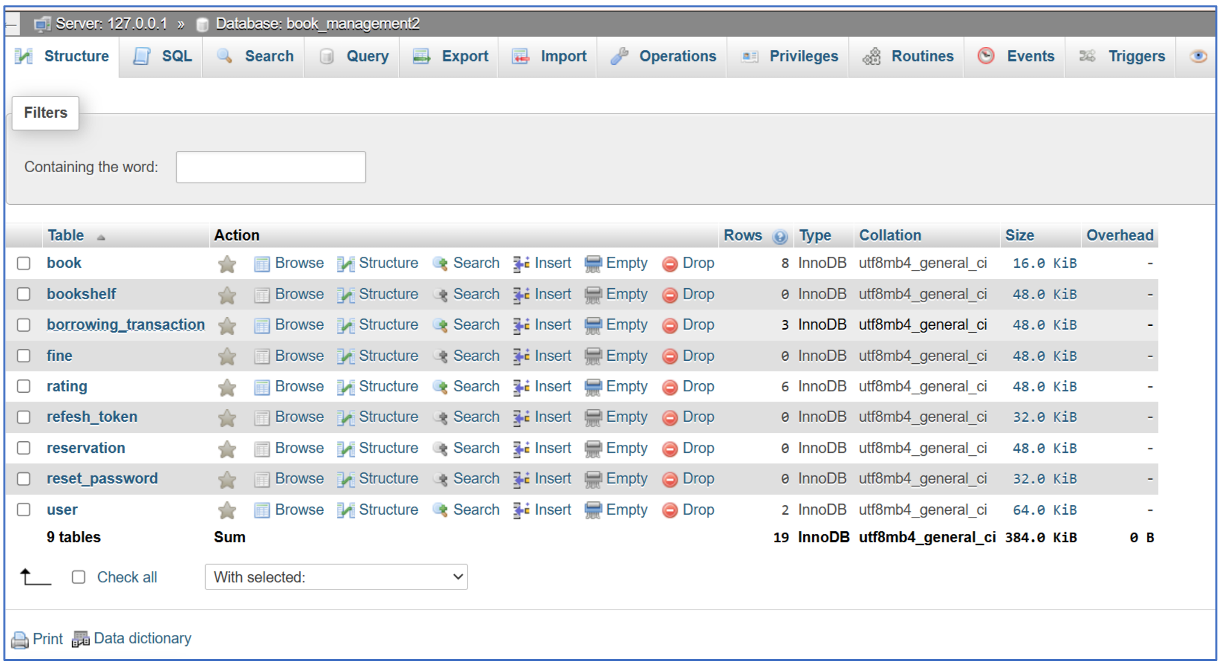

Server: 127.0.0.1 » Database: book_management2

	Table	Action	Rows	Type	Collation	Size	Overhead
☐	book	Browse Structure Search Insert Empty Drop	8	InnoDB	utf8mb4_general_ci	16.0 KiB	-
☐	bookshelf	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	48.0 KiB	-
☐	borrowing_transaction	Browse Structure Search Insert Empty Drop	3	InnoDB	utf8mb4_general_ci	48.0 KiB	-
☐	fine	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	48.0 KiB	-
☐	rating	Browse Structure Search Insert Empty Drop	6	InnoDB	utf8mb4_general_ci	48.0 KiB	-
☐	refesh_token	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	32.0 KiB	-
☐	reservation	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	48.0 KiB	-
☐	reset_password	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	32.0 KiB	-
☐	user	Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_general_ci	64.0 KiB	-
	9 tables	Sum	19	InnoDB	utf8mb4_general_ci	384.0 KiB	0 B

Figure 11. Database

- User Table: Stores user information, roles, and authentication details.
- Book Table: Keeps track of book details, stock levels.
- Borrowing Transactions Table: Records user borrowing activities.
- Rating Table: Records all rating activities.

4.3. Installation Environment.

4.3.1. Front-end Libraries

Icon libraries: fontawesome version 6.4.2 (CDN)

➢ Dependencies:

- "@emotion/react": "11.14.0",
- "@emotion/styled": "11.14.0",
- "@mui/material": "6.4.6",
- "@tailwindcss/vite": "4.0.9",
- "axios": "1.8.1",
- "react": "19.0.0",
- "react-dom": "19.0.0",
- "react-hook-form": "7.54.2",
- "react-pdf": "9.2.1",
- "react-toastify": "11.0.5",

- "tailwindcss": "4.0.9"

- Development Dependencies:
  - "@eslint/js": "9.19.0",
  - "@types/react": "19.0.8",
  - "@types/react-dom": "19.0.3",
  - "@vitejs/plugin-react": "4.3.4",
  - "eslint": "9.19.0",
  - "eslint-plugin-react-hooks": "5.0.0",
  - "eslint-plugin-react-refresh": "0.4.18",
  - "globals": "15.14.0",
  - "react-router-dom": "7.2.0",
  - "typescript": "~5.7.2",
  - "typescript-eslint": "8.22.0",
  - "vite": "6.1.0"

4.3.2. Back-end Libraries

- Dependencies:
  - "@nestjs-modules/mailer": "^2.0.2",
  - "@nestjs/axios": "4.0.0",
  - "@nestjs/common": "11.0.1",
  - "@nestjs/config": "4.0.0",
  - "@nestjs/core": "11.0.1",
  - "@nestjs/jwt": "11.0.0",
  - "@nestjs/mapped-types": "*",
  - "@nestjs/passport": "11.0.5",
  - "@nestjs/platform-express": "11.0.1",
  - "@nestjs/typeorm": "11.0.0",
  - "axios": "1.7.9",
  - "bcrypt": "5.1.1",
  - "class-transformer": "0.5.1",
  - "class-validator": "0.14.1",
  - "cookie-parser": "1.4.7",
  - "dayjs": "1.11.13",
  - "express-session": "1.18.1",

- "handlebars": "4.7.8",
- "mysql2": "3.12.0",
- "nest-winston": "1.10.2",
- "nodemailer": "6.10.0",
- "passport": "0.7.0",
- "passport-jwt": "4.0.1",
- "passport-local": "1.0.0",
- "reflect-metadata": "0.2.2",
- "rxjs": "7.8.1",
- "typeorm": "0.3.20",
- "uuid": "11.0.5",
- "winston": "3.17.0"

➢ Development Dependencies:

- "@eslint/eslintrc": "3.2.0",
- "@eslint/js": "9.18.0",
- "@nestjs/cli": "11.0.0",
- "@nestjs/schematics": "11.0.0",
- "@nestjs/testing": "11.0.1",
- "@swc/cli": "0.6.0",
- "@swc/core": "1.10.7",
- "@types/bcrypt": "5.0.2",
- "@types/cookie-parser": "1.4.8",
- "@types/express": "5.0.0",
- "@types/express-session": "1.18.1",
- "@types/jest": "29.5.14",
- "@types/multer": "1.4.12",
- "@types/node": "22.10.7",
- "@types/nodemailer": "6.4.17",
- "@types/passport-jwt": "4.0.1",
- "@types/passport-local": "1.0.38",
- "@types/supertest": "6.0.2",
- "@types/uuid": "10.0.0",
- "eslint": "9.18.0",

- "eslint-config-prettier": "10.0.1",
- "eslint-plugin-prettier": "5.2.2",
- "globals": "15.14.0",
- "jest": "29.7.0",
- "prettier": "3.4.2",
- "source-map-support": "0.5.21",
- "supertest": "7.0.0",
- "ts-jest": "29.2.5",
- "ts-loader": "9.5.2",
- "ts-node": "10.9.2",
- "tsconfig-paths": "4.2.0",
- "typescript": "5.7.3",
- "typescript-eslint": "8.20.0"

CHAPTER 5. RESULTS AND DISCUSSION

5.1. Results

5.1.1. Login Page

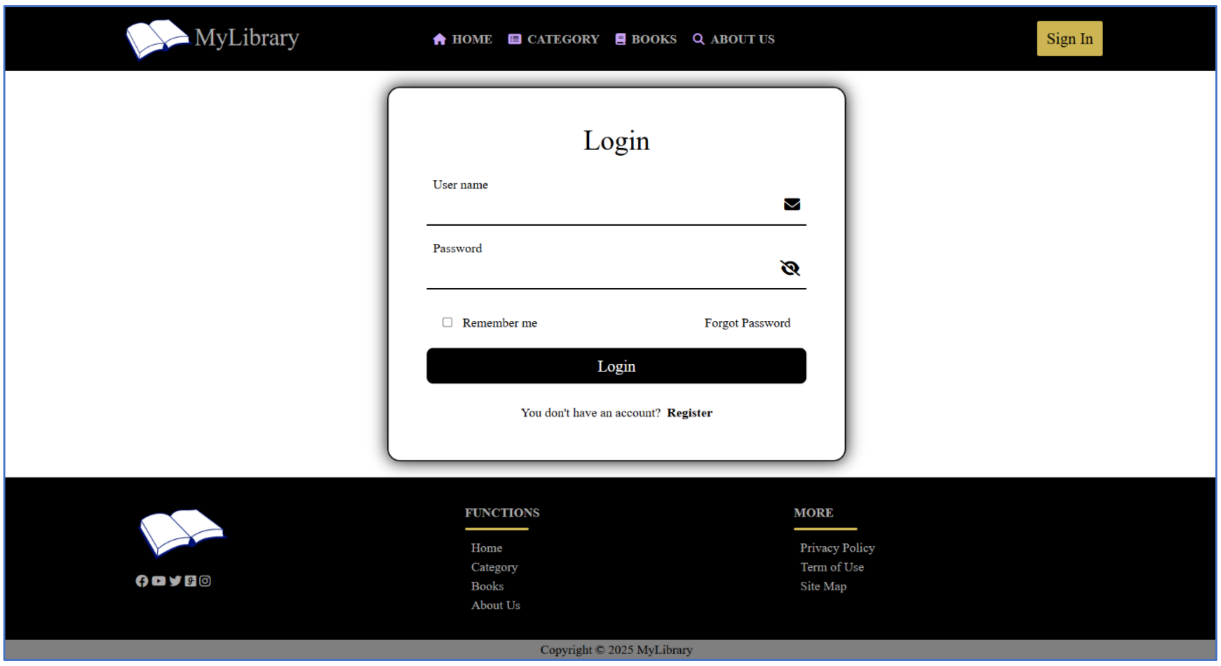


Figure 12. Login Page

- Allows users (members, admins) to access the system securely using username and password.
- Implements JWT authentication for session security.
- Validation checks prevent incorrect login attempts.

5.1.2. Register Page

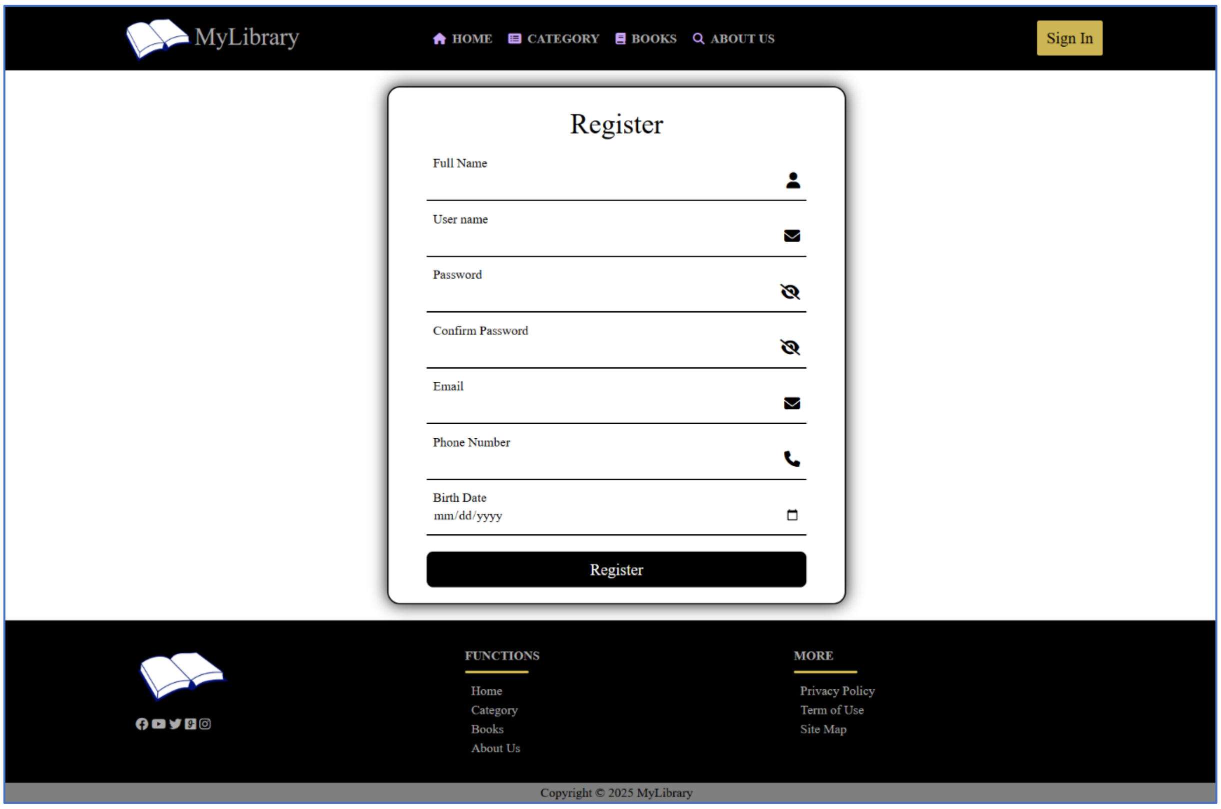


Figure 13. Register Page

- New users can create an account by providing missing input.
- Passwords are encrypted before being stored in the database.
- A success message is displayed upon successful registration.

5.1.3. Home Page

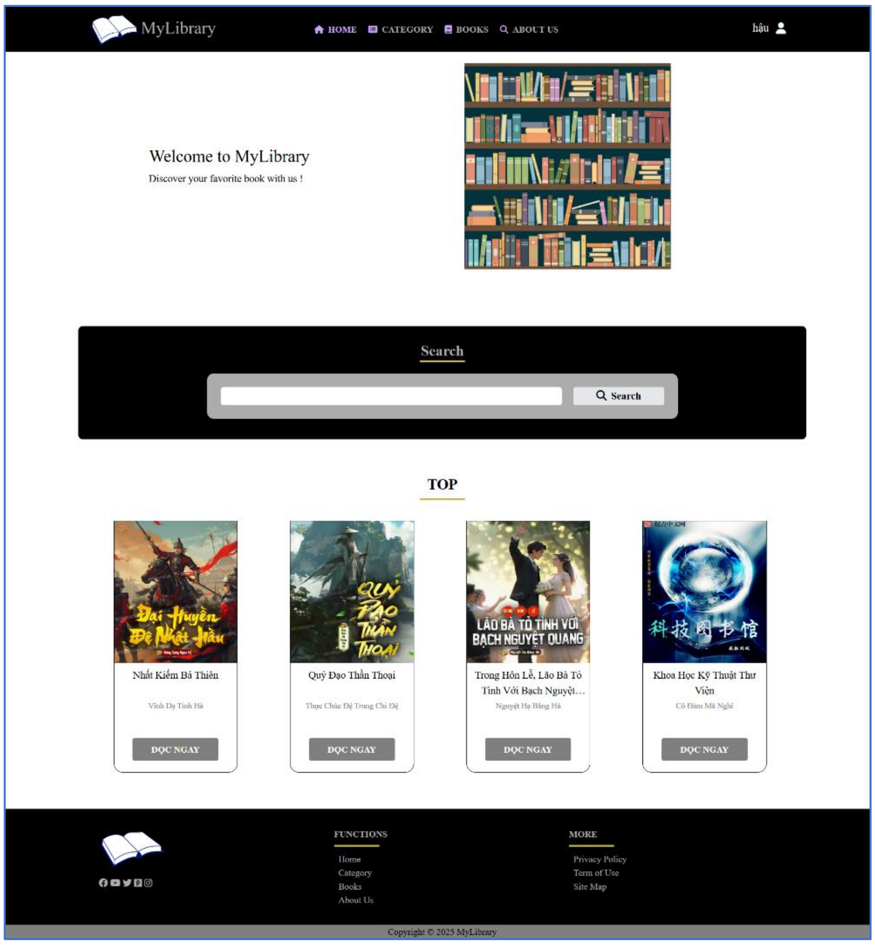


Figure 14. Home Page

- Displays home page of the web.
- Users can use the search function in this page.

5.1.4. Search Result Page.

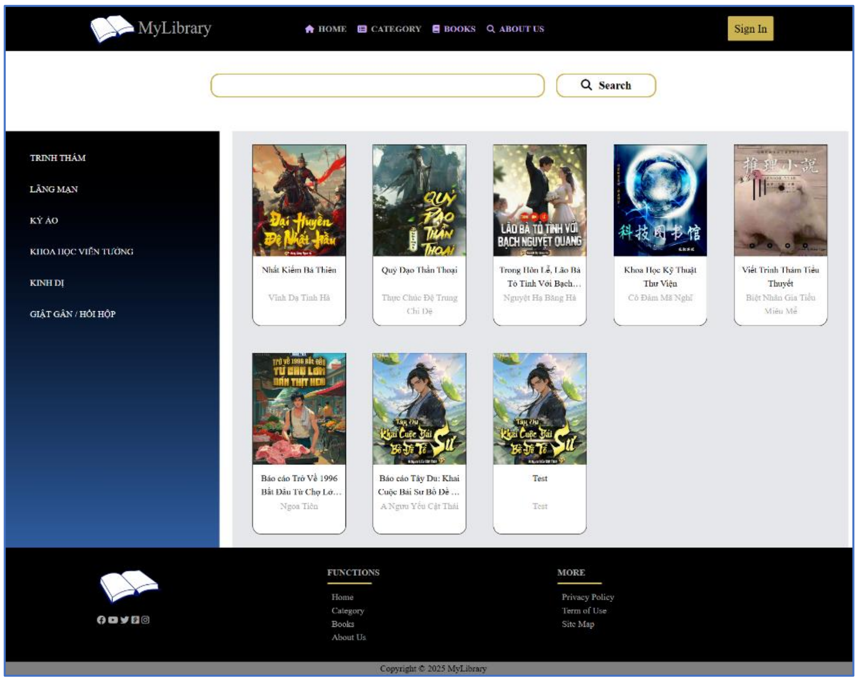
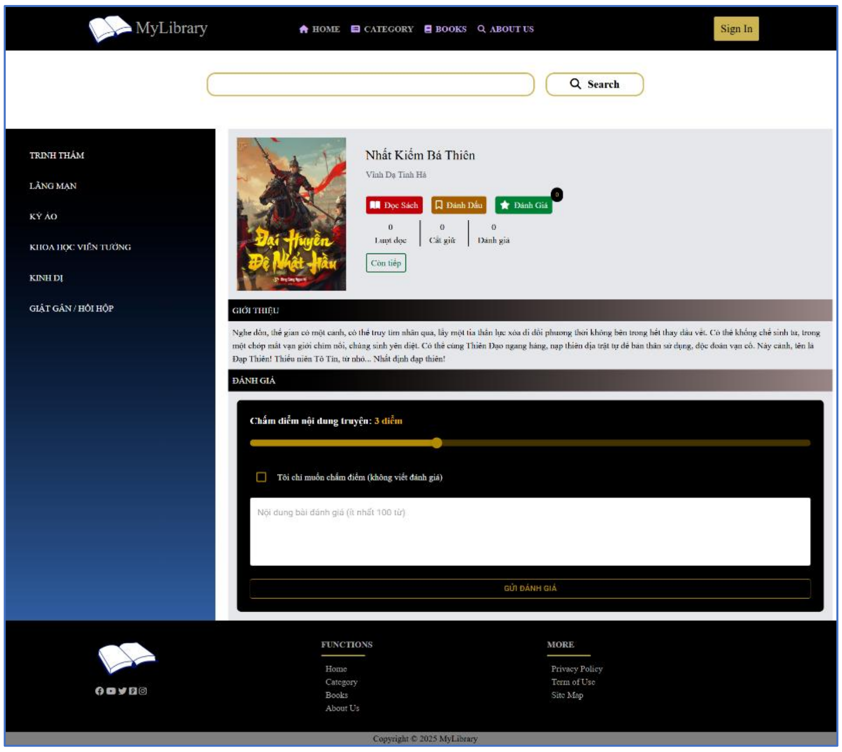


Figure 15: Search Result Page

- Users can search for books using **title, or genre**.

5.1.5. View Book Information Page

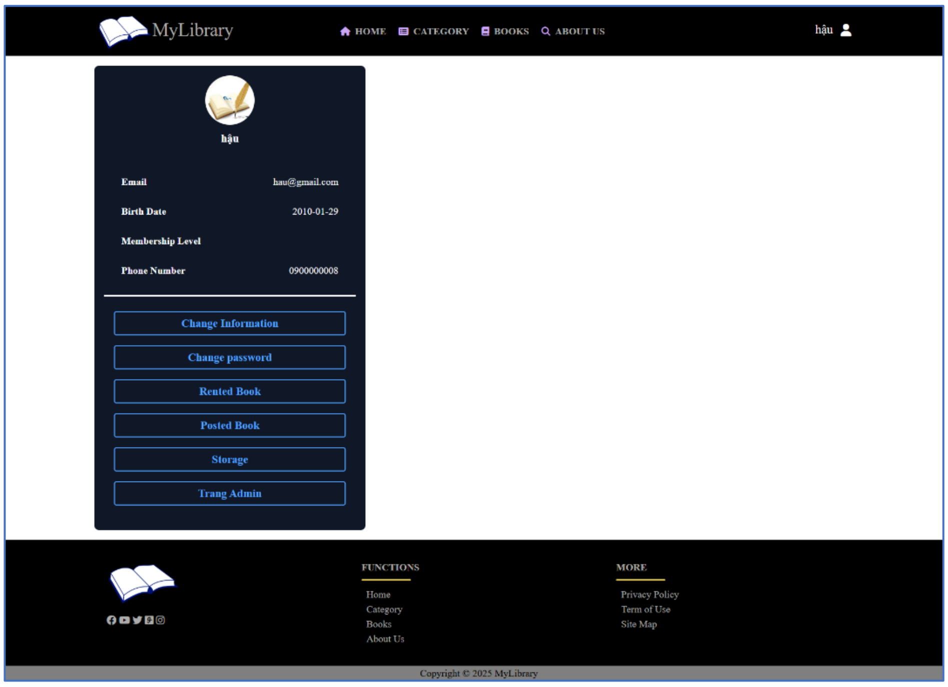


Nghe đồn, thế gian có một cảnh, có thể truy tìm nhân quả, lấy một tia thần lực xóa đi đối phương thời không bên trong hết thảy dấu vết. Có thể khống chế sinh tử, trong một chớp mắt vạn giới chìm nổi, chúng sinh yên diệt. Có thể cùng Thiên Đạo ngang hàng, nạp thiên địa trật tự để bản thân sử dụng, độc đoán vạn cổ. Này cảnh, tên là Đạp Thiên! Thiếu niên Tô Tín, từ nhỏ... Nhất định đạp thiên!

Figure 16. Book Information Page

- Shows **detailed book descriptions**, including **title, author, genre.**

5.1.6. User Page



hậu

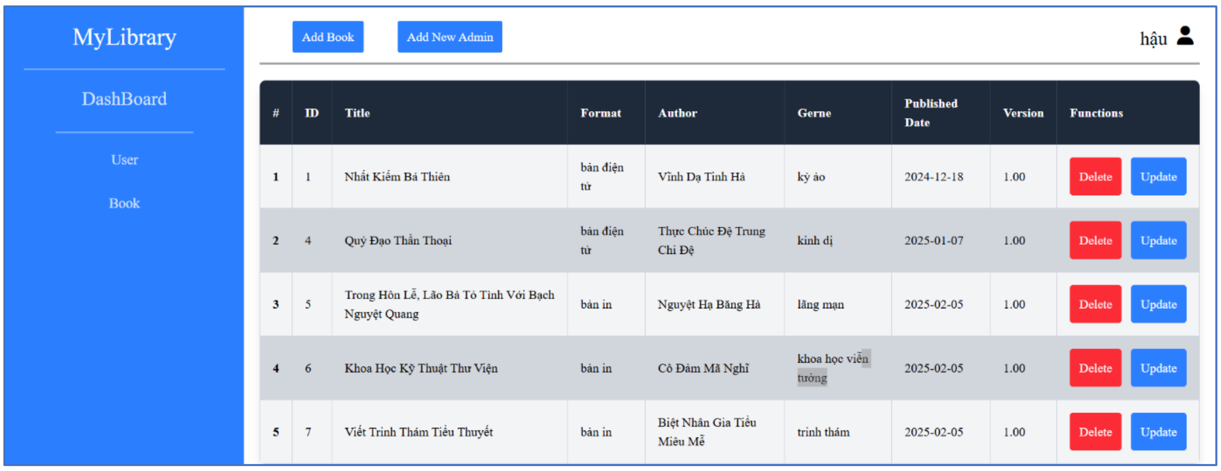
Email	hau@gmail.com
Birth Date	2010-01-29
Membership Level	
Phone Number	0900000008

Copyright © 2025 MyLibrary

Figure 17. User Page

- Displays user **profile details**.

5.1.7. Book Inventory Management Page (Admin Only)

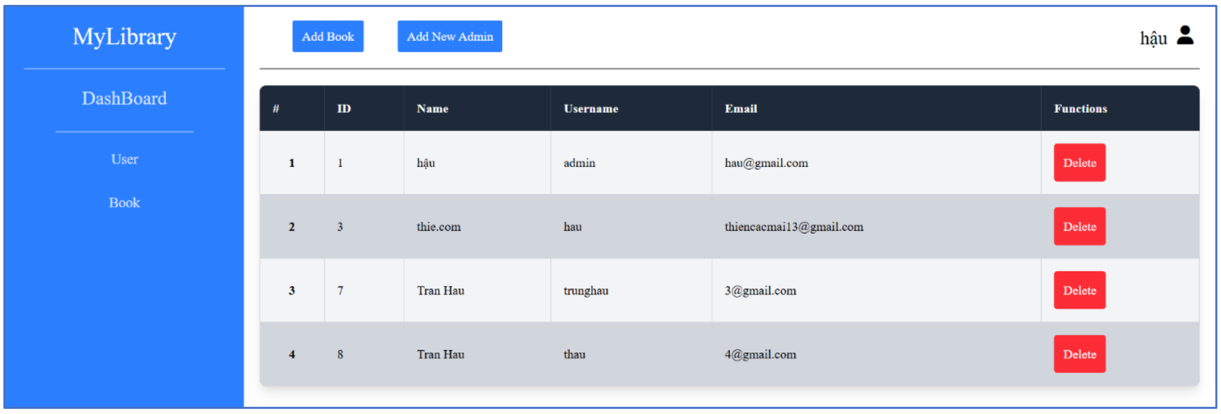


#	ID	Title	Format	Author	Gerne	Published Date	Version	Functions
1	1	Nhất Kiếm Bá Thiên	bản điện tử	Vĩnh Dạ Tinh Hà	kỳ ảo	2024-12-18	1.00	Delete Update
2	4	Quỷ Đạo Thần Thoại	bản điện tử	Thực Chúc Đệ Trung Chi Đệ	kinh dị	2025-01-07	1.00	Delete Update
3	5	Trong Hôn Lễ, Lão Bà Tỏ Tình Với Bạch Nguyệt Quang	bản in	Nguyệt Hạ Băng Hà	lãng mạn	2025-02-05	1.00	Delete Update
4	6	Khoa Học Kỹ Thuật Thư Viện	bản in	Cô Đảm Mã Nghĩ	khoa học viễn tưởng	2025-02-05	1.00	Delete Update
5	7	Viết Trinh Thám Tiểu Thuyết	bản in	Biệt Nhân Gia Tiểu Miêu Mễ	trinh thám	2025-02-05	1.00	Delete Update

Figure 18. Book Inventory Management Page

- Admins can **view all books**, check availability, and manage book details.
- Features **filters and sorting options** for easy book management.

5.1.8. User Management Page (Admin Only)



#	ID	Name	Username	Email	Functions
1	1	hậu	admin	hau@gmail.com	Delete
2	3	thie.com	hau	thiencacmai13@gmail.com	Delete
3	7	Tran Hau	trunghau	3@gmail.com	Delete
4	8	Tran Hau	thau	4@gmail.com	Delete

Figure 19. User Management Page

- Admins can **view all registered users** and manage their roles.
- Provides options to **delete or update user details**.
- **Security:** Only authorized admins can make changes.

5.2. Discussion

5.2.1. Project Outcomes

The developed **Book Management System** provides core functionalities such as **user authentication, book browsing and administrative control** over the book inventory. The system allows users to:

- **Search for books** by name, author, or genre.
- **Administrators** can **manage book records** (add, edit, delete) and **manage user (add, edit, delete)**.

The **frontend** was built using **React.js with Tailwind CSS**, ensuring a **responsive and user-friendly UI**. The **backend** was developed in **Node.js with NestJS**, handling API requests efficiently. Data is stored securely in **MySQL**, and authentication is managed through **JWT tokens** for secure access control.

5.2.2. Comparison with Objectives

- Rating system still not implemented in the initial version.
- Book borrowing system still under development.
- The frontend design was optimized for ease of use.
- JWT and encrypted passwords ensure security.
- Admins can fully manage books and users.

CHAPTER 6. CONCLUSION AND FUTURE WORKS

6.1. Conclusion.

The Book Management System was developed to provide a streamlined solution for book inventory management, user administration, and book search functionalities. The system successfully implements key features such as user authentication, book searching, Wishlist management, and admin functionalities for managing books and users.

The system was built using React.js for the frontend, Nest JS for the backend, and MySQL for database management, ensuring scalability, efficiency, and security. While core functionalities are fully implemented, some additional features like book rental and rating are still under development and will be introduced in future updates.

6.1.1. Challenges and Solutions

- Ensuring secure authentication.
- Managing large amounts of book data efficiently.

6.1.2. Solution

- Implemented JWT (JSON Web Token) for safe and efficient login/logout handling.
- Optimized database queries with indexing and pagination techniques.

As the system evolves, several enhancements and new features are planned to improve user experience and functionality.

6.1.3. Book Rental System

- Develop a fully functional book rental module that allows users to borrow and return books digitally.
- Implement a real-time availability check to prevent overbooking.
- Add rental history tracking for users and administrators.

6.1.4. Rating and Review System

- Allow users to rate and review books based on their reading experience.
- Implement an average rating display on each book's details page.

6.1.5. Advanced Search and Filtering

- Maybe enhance the search algorithm by including AI-based recommendations based on user preferences.
- Implement advanced filtering options (e.g., genre, author, publication year).

6.1.6. Improved User Dashboard

- Provide analytics for admins to track most viewed books, user engagement, and system usage statistics.

REFERENCES

[1] React Team. (n.d.). *React documentation*.

[2] Wieruch, R. (2020). *The Road to React*. Leanpub.

[3] Vite Team. (n.d.). *Vite documentation*.

[4] Tailwind Labs. (n.d.). *Tailwind CSS Documentation*.

[5] Fonticons, Inc. (n.d.). *Font Awesome Documentation*.

[6] NestJS Team. (n.d.). *NestJS Documentation*.

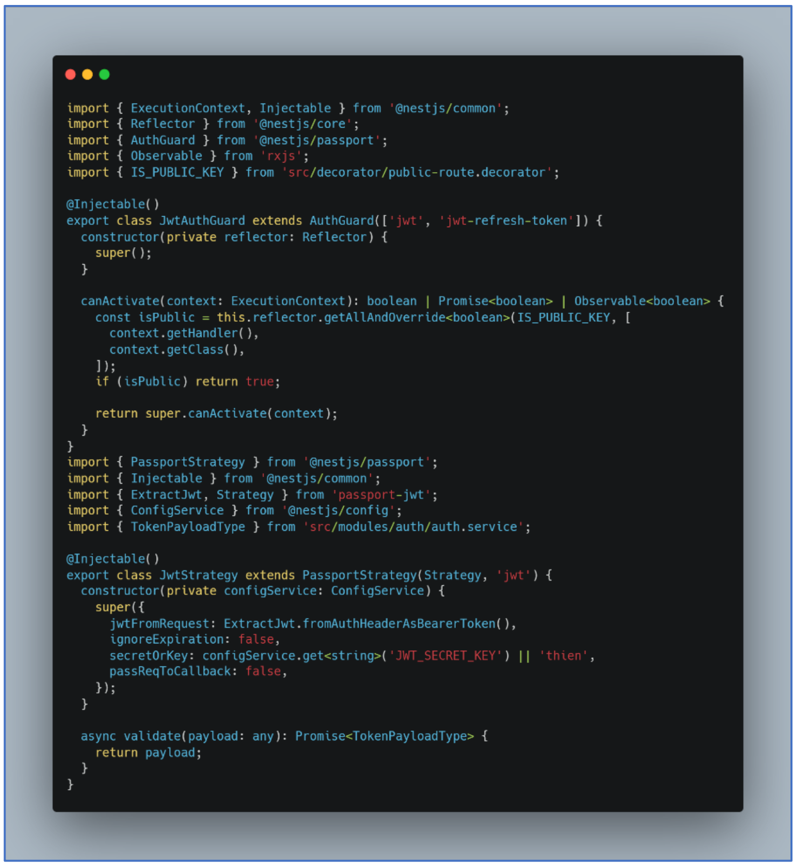
[7] MariaDB Foundation. (n.d.). *MariaDB Knowledge Base*.

APPENDIX

Code Snippets:

- Backend API – User Authentication (Nest JS):

The following code demonstrates how user authentication is implemented using Nest JS and JWT (JSON Web Token).

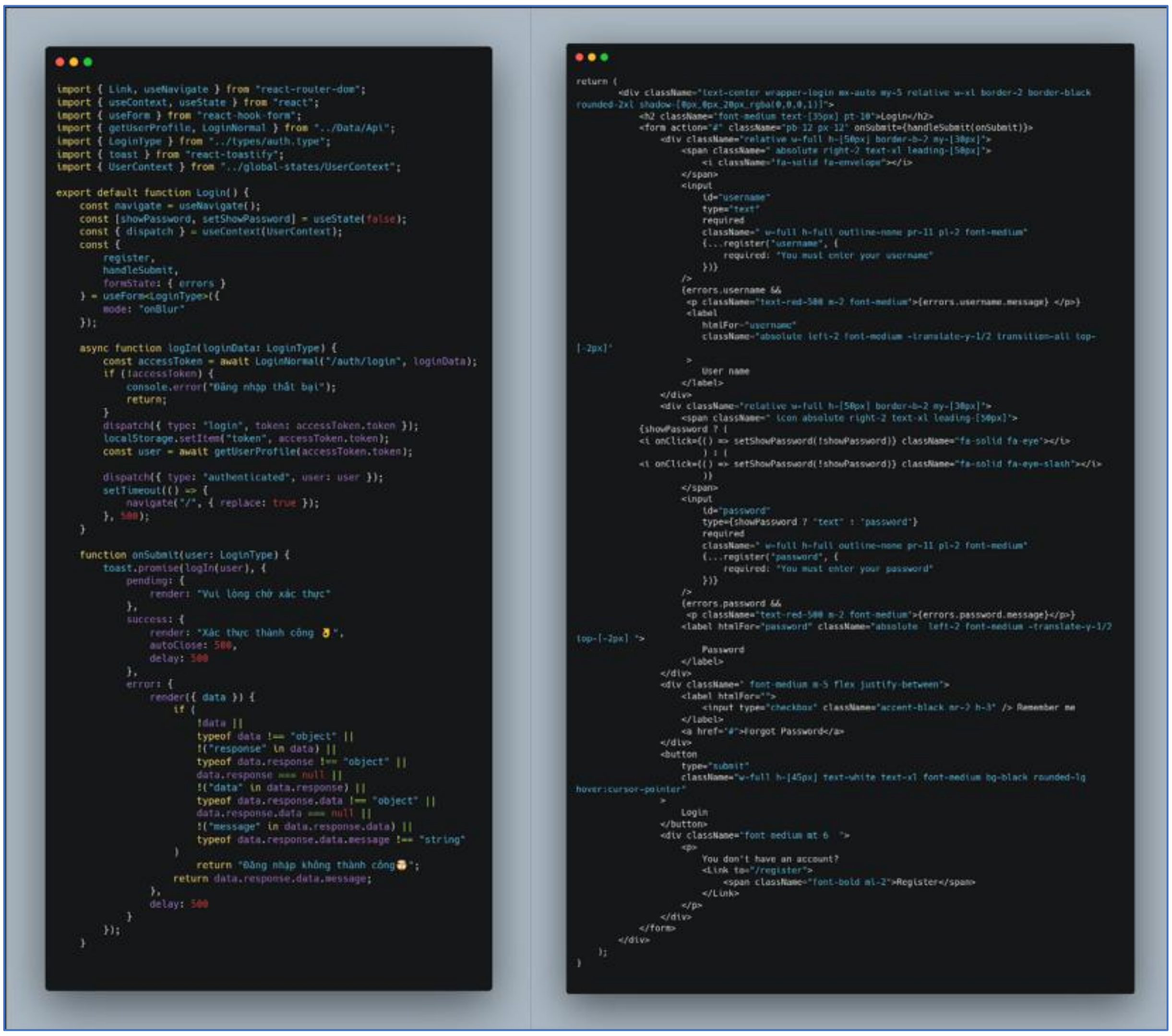


```
import { ExecutionContext, Injectable } from '@nestjs/common';
import { Reflector } from '@nestjs/core';
import { AuthGuard } from '@nestjs/passport';
import { Observable } from 'rxjs';
import { IS_PUBLIC_KEY } from 'src/decorator/public-route.decorator';

@Injectable()
export class JwtAuthGuard extends AuthGuard(['jwt', 'jwt-refresh-token']) {
  constructor(private reflector: Reflector) {
    super();
  }

  canActivate(context: ExecutionContext): boolean | Promise<boolean> | Observable<boolean> {
    const isPublic = this.reflector.getAllAndOverride<boolean>(IS_PUBLIC_KEY, [
      context.getHandler(),
      context.getClass(),
    ]);
    if (isPublic) return true;

    return super.canActivate(context);
  }
}
import { PassportStrategy } from '@nestjs/passport';
import { Injectable } from '@nestjs/common';
import { ExtractJwt, Strategy } from 'passport-jwt';
import { ConfigService } from '@nestjs/config';
import { TokenPayloadType } from 'src/modules/auth/auth.service';

@Injectable()
export class JwtStrategy extends PassportStrategy(Strategy, 'jwt') {
  constructor(private configService: ConfigService) {
    super({
      jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
      ignoreExpiration: false,
      secretOrKey: configService.get<string>('JWT_SECRET_KEY') || 'thien',
      passReqToCallback: false,
    });
  }

  async validate(payload: any): Promise<TokenPayloadType> {
    return payload;
  }
}
```

Figure 20. Backend User Authentication

- Frontend – User Login Component (React & Tailwind CSS):

This React component allows users to enter credentials and authenticate via the backend.



```tsx
import { Link, useNavigate } from "react-router-dom";
import { useContext, useState } from "react";
import { useForm } from "react-hook-form";
import { getUserProfile, LoginNormal } from "../Data/Api";
import { LoginType } from "../types/auth.type";
import { toast } from "react-toastify";
import { UserContext } from "../global-states/UserContext";

export default function Login() {
    const navigate = useNavigate();
    const [showPassword, setShowPassword] = useState(false);
    const { dispatch } = useContext(UserContext);
    const {
        register,
        handleSubmit,
        formState: { errors }
    } = useForm<LoginType>({
        mode: "onBlur"
    });

    async function logIn(loginData: LoginType) {
        const accessToken = await LoginNormal("/auth/login", loginData);
        if (!accessToken) {
            console.error("Đăng nhập thất bại");
            return;
        }
        dispatch({ type: "login", token: accessToken.token });
        localStorage.setItem("token", accessToken.token);
        const user = await getUserProfile(accessToken.token);

        dispatch({ type: "authenticated", user: user });
        setTimeout(() => {
            navigate("/", { replace: true });
        }, 500);
    }

    function onSubmit(user: LoginType) {
        toast.promise(logIn(user), {
            pending: {
                render: "Vui lòng chờ xác thực"
            },
            success: {
                render: "Xác thực thành công 👌",
                autoClose: 500,
                delay: 500
            },
            error: {
                render({ data }) {
                    if (
                        !data ||
                        typeof data !== "object" ||
                        !("response" in data) ||
                        typeof data.response !== "object" ||
                        data.response === null ||
                        !("data" in data.response) ||
                        typeof data.response.data !== "object" ||
                        data.response.data === null ||
                        !("message" in data.response.data) ||
                        typeof data.response.data.message !== "string"
                    )
                        return "Đăng nhập không thành công🤯";
                    return data.response.data.message;
                },
                delay: 500
            }
        });
    }
```

```tsx
return (
        <div className="text-center wrapper-login mx-auto my-5 relative w-xl border-2 border-black
rounded-2xl shadow-[0px_0px_20px_rgba(0,0,0,1)]">
            <h2 className="font-medium text-[35px] pt-10">Login</h2>
            <form action="#" className="pb-12 px-12" onSubmit={handleSubmit(onSubmit)}>
                <div className="relative w-full h-[50px] border-b-2 my-[30px]">
                    <span className=" absolute right-2 text-xl leading-[50px]">
                        <i className="fa-solid fa-envelope"></i>
                    </span>
                    <input
                        id="username"
                        type="text"
                        required
                        className=" w-full h-full outline-none pr-11 pl-2 font-medium"
                        {...register("username", {
                            required: "You must enter your username"
                        })}
                    />
                    {errors.username &&
                     <p className="text-red-500 m-2 font-medium">{errors.username.message} </p>}
                     <label
                        htmlFor="username"
                        className="absolute left-2 font-medium -translate-y-1/2 transition-all top-
[-2px]"

                     >
                        User name
                    </label>
                </div>
                <div className="relative w-full h-[50px] border-b-2 my-[30px]">
                    <span className=" icon absolute right-2 text-xl leading-[50px]">
            {showPassword ? (
            <i onClick={() => setShowPassword(!showPassword)} className="fa-solid fa-eye"></i>
                        ) : (
            <i onClick={() => setShowPassword(!showPassword)} className="fa-solid fa-eye-slash"></i>
                        )}
                    </span>
                    <input
                        id="password"
                        type={showPassword ? "text" : "password"}
                        required
                        className=" w-full h-full outline-none pr-11 pl-2 font-medium"
                        {...register("password", {
                            required: "You must enter your password"
                        })}
                    />
                    {errors.password &&
                     <p className="text-red-500 m-2 font-medium">{errors.password.message}</p>}
                    <label htmlFor="password" className="absolute  left-2 font-medium -translate-y-1/2
top-[-2px] ">

                        Password
                    </label>
                </div>
                <div className=" font-medium m-5 flex justify-between">
                    <label htmlFor="">
                        <input type="checkbox" className="accent-black mr-2 h-3" /> Remember me
                    </label>
                    <a href="#">Forgot Password</a>
                </div>
                <button
                    type="submit"
                    className="w-full h-[45px] text-white text-xl font-medium bg-black rounded-lg
hover:cursor-pointer"
                >
                    Login
                </button>
                <div className="font medium mt 6  ">
                    <p>
                        You don't have an account?
                        <Link to="/register">
                            <span className="font-bold ml-2">Register</span>
                        </Link>
                    </p>
                </div>
            </form>
        </div>
    );
}
```

Figure 21. Login Component

REPORT CONTENT AND STRUCTURE

Abstract

Acknowledgement

Table of Content

List of Figures

List of Tables

List of Abbreviations

I. Introduction

II. Related Works

III. Proposed Approach, Analysis, Design, Implementation

IV. Results and Discussion

V. Conclusion and Future Works

References

Appendix

GUIDELINES

1. Report cover page is WHITE
2. Chapter TITLE: Times New roman, size 14 points, bold, CENTER
3. Section Title: Times New roman, size 13 points, bold
4. Sub-section Title: Times New roman, size 12 points, bold
5. Texts: Times New roman, size 12 points, Left and Right justified
6. Line spacing: 1.5
7. Printing of Page: Single sided with page number (Bottom, Center) of each page
8. Diagrams: Colored (using colored printer)
9. Introduction: Should include project statement, background and motivation.
10. Reference Format Example:

    [8] E. H. E. Bayoumi, M. N. F. Nashed, *A Fuzzy Predictive Sliding Mode Control for High Performance Induction Motor Position Drives*, Journal of Power Electronics, Vol. 5, No.1, pp.20-28, 2005.

    [9] B. K. Bose, *Power electronics and variable frequency drives: Technology and Applications*, IEEE Press, New York, 1996.