Building a Scalable and Secure Library Management System with FastAPI and PostgreSQL

By: Elias Nada

Introduction:

Libraries are essential for managing and organizing books, users, and borrowing activities. However, traditional library systems often lack scalability, security, and modern features. This case study explores the development of a Library Management System using FastAPI and PostgreSQL to address these challenges.

Problem Statement:

The goal was to design and implement a library management system that:

- Tracks book details, availability, and categories.
- Manages user authentication and roles.
- Records borrowing history and return status.
- Ensures scalability, security, and performance optimization.

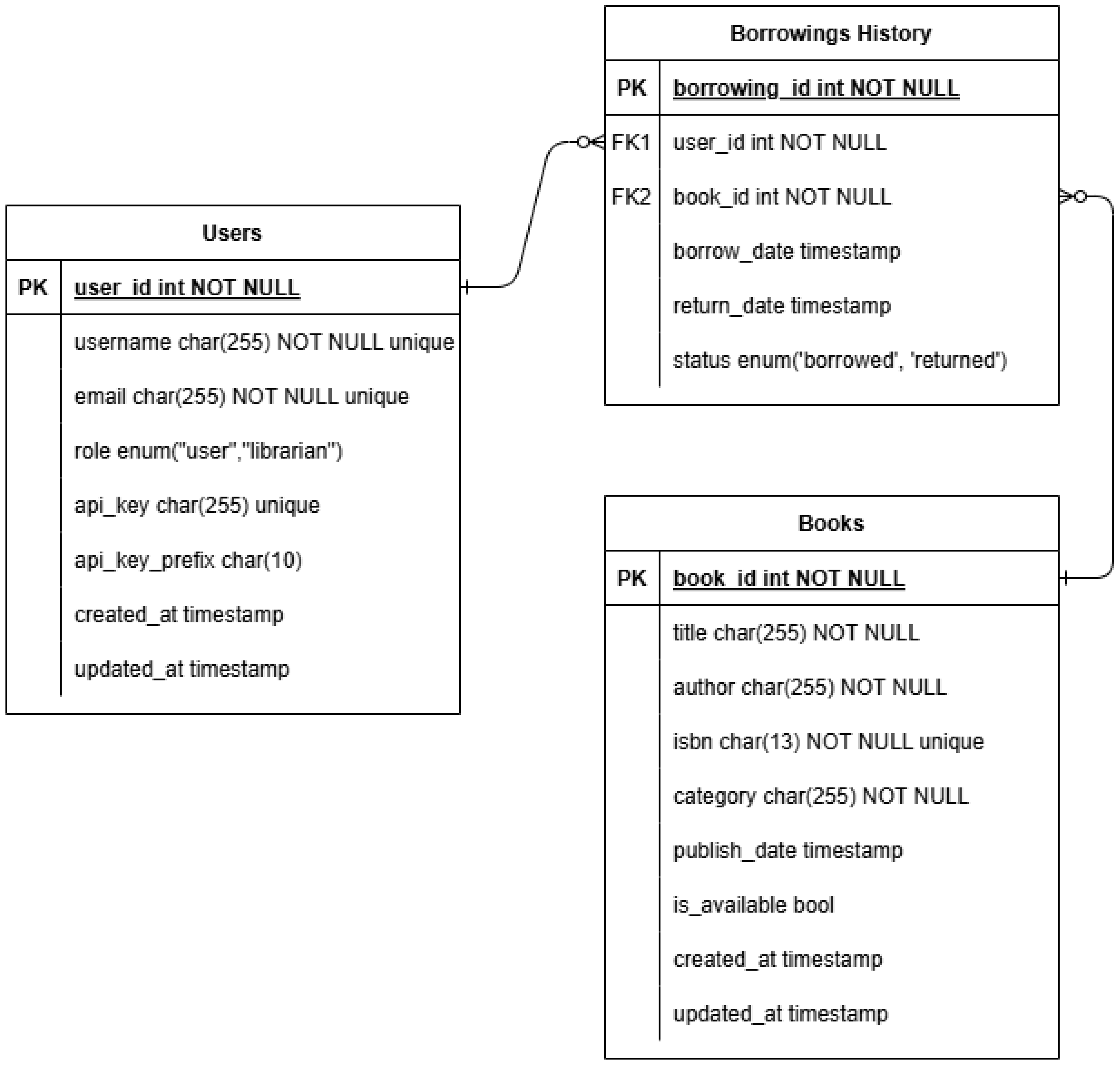
Solution Overview:

The solution is a RESTful API built with FastAPI and backed by a PostgreSQL database. Key features include:

- Book Management: Add, update, delete, and search books.
- User Management: Register, authenticate, and manage user roles.
- Borrowing History: Track book borrowing and returns.
- Authentication: Secure API access using JWT and API keys.
- Scalability: Optimized database schema and indexing for performance.

Implementation Details:

Database Design:

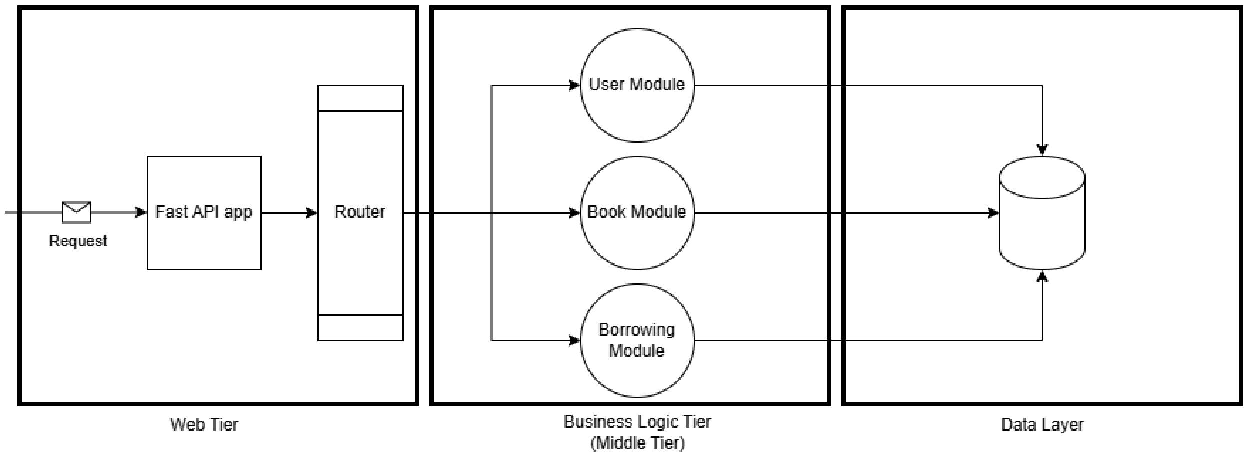


The database schema includes tables for books, users, and borrowing_history.

Relationships are defined using foreign keys (e.g., borrowing_history.book_id references book.id).

Indexes are added for efficient querying (e.g., on books.isbn and users.username).

API Development:



The API is built using FastAPI, with endpoints for book management, user actions, and borrowing history.

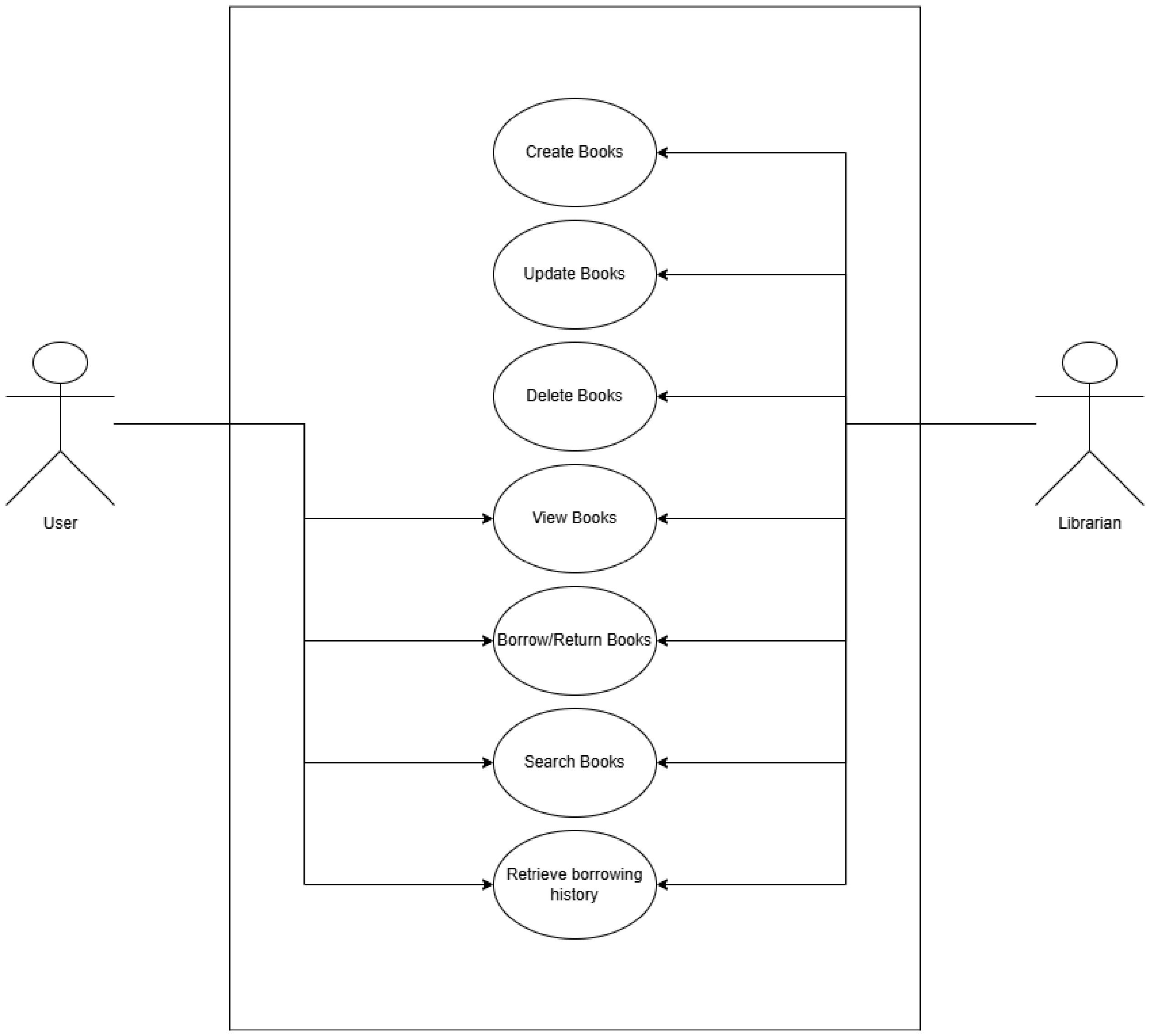
Role-based access control ensures that only librarians can manage books.

Error handling and logging are implemented for robustness.

Authentication:

Users can authenticate using JWT or API keys.

API keys are hashed before storage to ensure security.



Search Functionality:

Users can search for books by title, category, release date, or availability status.

Challenges and Solutions:

Challenge 1: Scalability

Problem: The system needed to handle a large number of books and users.

Solution: The database schema was normalized, and indexes were added for efficient querying. Pagination was implemented for search results.

Challenge 2: Security

Problem: API keys needed to be stored securely.

Solution: API keys were hashed using bcrypt before storage, ensuring they could not be retrieved even if the database was compromised.

Challenge 3: Authentication

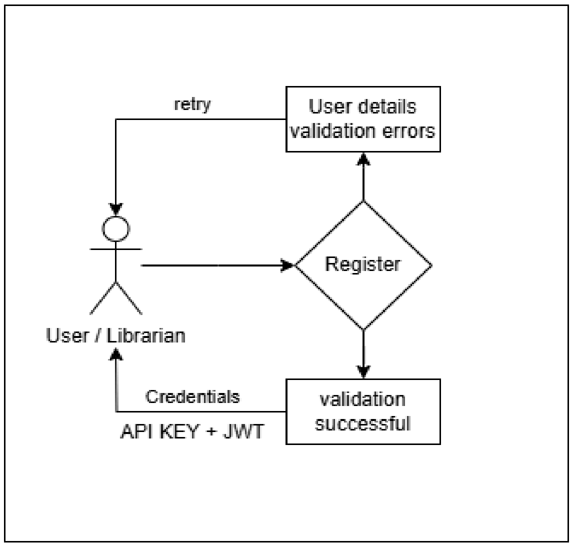
Problem: Users needed flexible authentication options.

Solution: The system supports both JWT and API key authentication, allowing users to choose their preferred method.

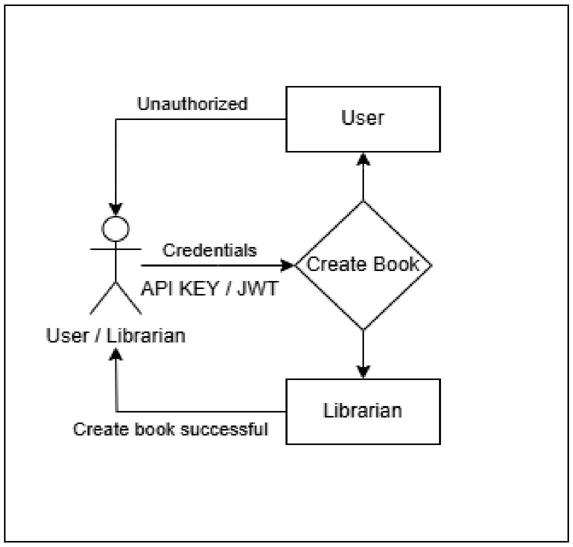
Results and Impact:

The library management system successfully addresses the requirements:

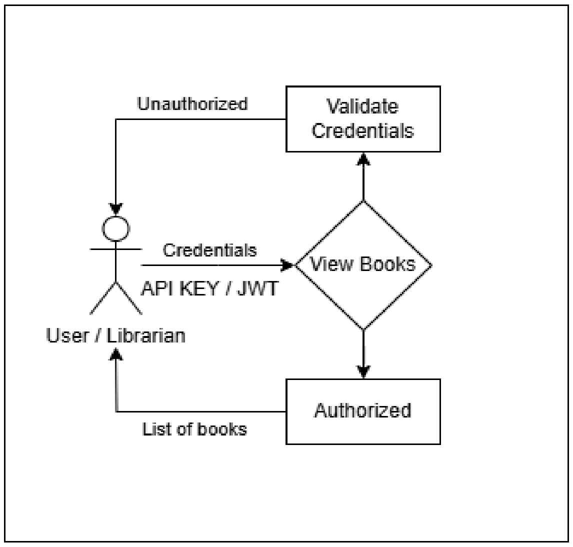
- Efficient Book Management: Librarians can easily add, update, and delete books.



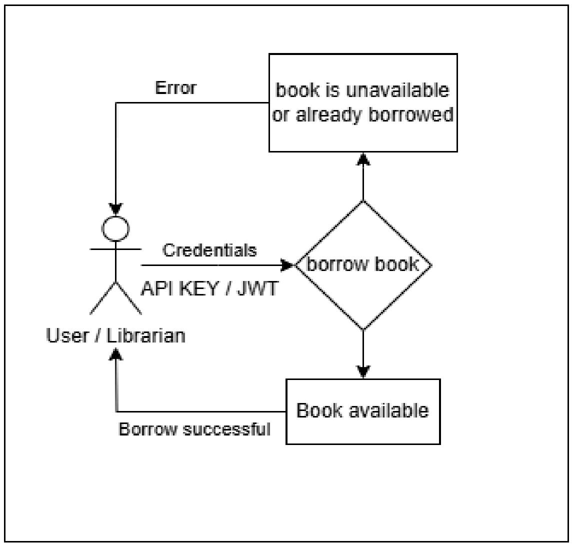
register flow



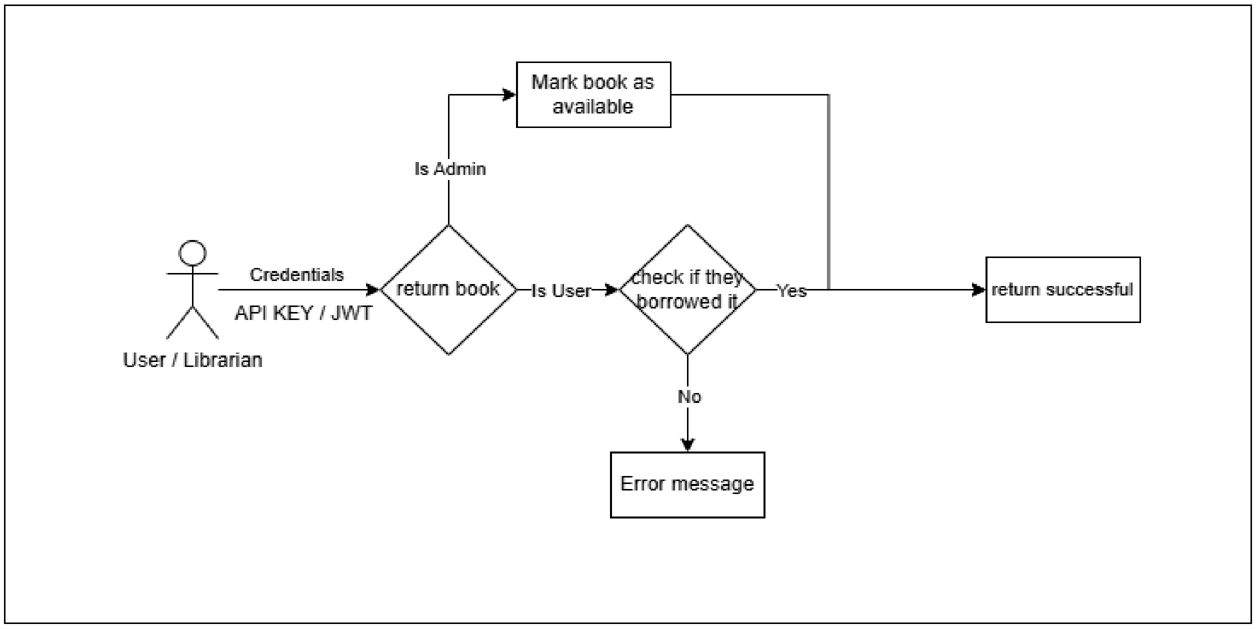
create book flow



view books flow



borrow book flow



borrowing history.
- Secure: Authentication mechanisms ensure that only authorized users can access the system.
- Scalable: The system can handle a growing number of books and users without performance degradation.

Conclusion:

This project demonstrates the power of modern web frameworks like FastAPI and relational databases like PostgreSQL for building scalable and secure systems. By addressing challenges like scalability, security, and flexibility, the library management system provides a robust solution for managing books and users.

Future Work:

Future enhancements could include:

- Advanced Search: Add support for full-text search and filtering by multiple criteria.
- Notifications: Send email reminders for overdue books.
- Analytics: Provide insights into borrowing trends and popular books.
- Database improvement: Implement Database sharding and partitioning to handle very high volume of information
- Caching: Add a cache layer for frequently accessed data.
- Asynchronous Processing: Use a message queue (e.g., Celery with RabbitMQ or Kafka) to handle background tasks, such as updating borrowing history and sending notifications (e.g., overdue books).
- Load Balancing and Replication: Use read replicas to distribute read queries across multiple database instances.
- Monitoring and Scaling: Use monitoring tools (e.g., Prometheus, Grafana) to track database performance and identify bottlenecks.