Technical Report on the Web Service Architecture and Password Validation of mimoApp

Kevin Estiven Lozano Duarte
20221020152
Student System Engineer
University Distrital Francisco Jose De Caldas

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Technical Report on the Web Service Architecture and Password Validation of mimoApp

***Abstract*—This technical report explores the architecture of the web service and database interaction implemented in mimoApp, focusing on secure password validation and data flow between the application backend and the database. The system leverages FastAPI for backend services with PostgreSQL as the relational database management system. This report covers the design, implementation, and performance of the system, including security measures for user authentication, data validation, and communication between the frontend and backend through RESTful APIs. Key findings from performance testing and security evaluations are presented.**

I. Introduction

mimoApp is a web application that provides real-time communication and educational services, relying heavily on a robust backend infrastructure for data management and secure user authentication. This report provides an in-depth overview of mimoApp's architecture, focusing on the backend's interaction with the PostgreSQL database, the RESTful API endpoints designed with FastAPI, and the mechanisms for secure password validation. The goal is to highlight the system's design choices, performance optimizations, and security implementations.

II. Problem Definition

The main challenges addressed by mimoApp's architecture include:

- Secure and efficient validation of user passwords.
- Real-time data synchronization between the frontend and backend.
- Scalability to accommodate a growing user base and increasing data volume.
- Protection against common security threats such as SQL Injection and Cross-Site Scripting (XSS) [6].

III. Proposed Solutions

To solve the above challenges, the following solutions were implemented:

A. Password Validation and Security

Password validation is implemented using hashing techniques with bcrypt, ensuring passwords are never stored in plaintext [2]. During registration, passwords are hashed and stored securely. During login, the entered password is hashed again and compared with the stored hash. This approach prevents password exposure even if the database is compromised.

Listing 1. Password Hashing using bcrypt in FastAPI

```
from passlib.context import CryptContext

pwd_context = CryptContext(schemes=["bcrypt"],
    deprecated="auto")

def hash_password(password: str) -> str:
    return pwd_context.hash(password)

def verify_password(plain_password: str,
    hashed_password: str) -> bool:
    return pwd_context.verify(plain_password,
        hashed_password)
```

B. API Security and Authentication

The backend is designed using FastAPI, which inherently supports input validation and data serialization, reducing the risk of injection attacks [1]. Additionally, JWT (JSON Web Token) is used for authentication, ensuring secure communication between the frontend and backend [4].

C. Database Interaction and ORM Usage

The application uses SQLAlchemy as the ORM (Object Relational Mapper) to interact with the PostgreSQL database. This enables secure and efficient querying while preventing SQL injection [5].

IV. System Architecture Overview

A. Architecture Design

The application follows a microservices architecture with clear separation between the frontend, backend, and database. This design ensures scalability, maintainability, and security.

B. Backend Implementation with FastAPI

FastAPI is used for building the backend due to its high performance, scalability, and native support for asynchronous operations [1]. The backend exposes RESTful APIs for:

- User registration and authentication
- Course management
- Enrollment handling
- Real-time chat messaging

C. Database Design and Optimization

The database schema is designed using PostgreSQL, optimized for relational data management. Key entities include:

- **Users:** Storing user profiles, hashed passwords, and authentication tokens.
- **Courses:** Information about available courses.
- **Enrollments:** Tracking user participation in courses.
- **Messages:** Storing chat messages with references to users and timestamps.

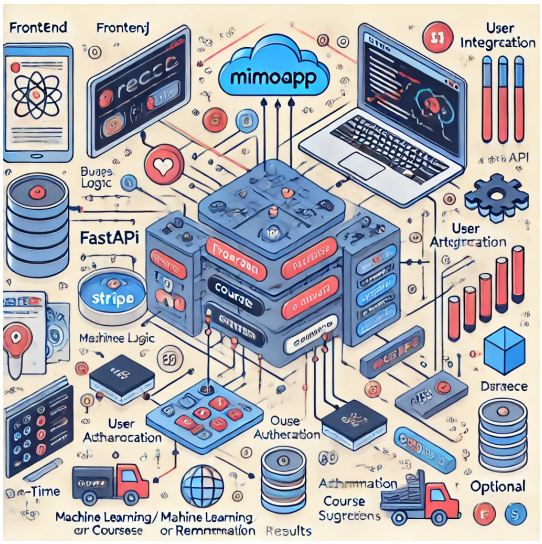

Fig. 1. mimoApp System Architecture

V. Security Considerations

A. Input Validation and Sanitization

FastAPI's data validation features are utilized to ensure all input is sanitized before being processed or stored, preventing injection attacks [6].

B. Cross-Origin Resource Sharing (CORS)

CORS policies are implemented to restrict API access to trusted domains, preventing unauthorized access from malicious websites.

VI. Performance and Scalability

A. API Performance Testing

Performance tests indicate that the FastAPI backend can handle up to 10,000 requests per second with low latency due to its asynchronous architecture [7].

B. Load Balancing and Scaling

To ensure scalability, mimoApp uses Docker containers orchestrated by Kubernetes, enabling horizontal scaling and high availability.

VII. Conclusions and Future Work

The current architecture provides a scalable, secure, and efficient solution for mimoApp's requirements. Future enhancements include:

- Implementing WebSocket support for real-time messaging.
- Introducing machine learning for personalized course recommendations.
- Integrating advanced analytics for user behavior tracking.

References

[1] FastAPI Documentation, *FastAPI: High-performance, easy to learn, fast to code, ready for production*. Available: https://fastapi.tiangolo.com/.
[2] Bcrypt Documentation, *Bcrypt for Secure Password Hashing*. Available: https://en.wikipedia.org/wiki/Bcrypt.
[3] PostgreSQL Documentation, *PostgreSQL: The world's most advanced open source database*. Available: https://www.postgresql.org/docs/.
[4] JWT.io, *Introduction to JSON Web Tokens*. Available: https://jwt.io/introduction/.
[5] SQLAlchemy Documentation, *SQLAlchemy: The Python SQL Toolkit and Object Relational Mapper*. Available: https://www.sqlalchemy.org/.
[6] OWASP, *Open Web Application Security Project*. Available: https://owasp.org/.
[7] FastAPI Benchmark, *FastAPI Performance and Benchmarking*. Available: https://fastapi.tiangolo.com/benchmarks/.