A Rubik's Cube Timer Application: Design and Implementation

Santiago Andrés Benavides Coral
Engineering faculty
Universidad Distrital Francisco José de Caldas
Email: sabenavidesc@udistrital.edu.co

***Abstract*—This paper presents the design and implementation of a Rubik's Cube Timer application, developed to assist speedcubers in timing their solves and tracking their progress. The application features a user-friendly frontend built with HTML, CSS, and JavaScript, served via Nginx, and a robust backend implemented using FastAPI for handling user authentication, solve timing, and statistics. The backend uses a JSON file for data persistence, ensuring simplicity and ease of deployment. This paper discusses the system architecture, design principles, and implementation details, along with an analysis of the application's performance and future work.**

***Index Terms*—Rubik's Cube, Speedcubing, FastAPI, Frontend, Backend, JSON, Nginx**

I. Introduction

The Rubik's Cube Timer application is a web-based tool designed for speedcubers to time their solves, track their progress, and analyze their performance. The application provides a user-friendly interface for timing solves, along with features such as user authentication, solve history, and statistics. The backend is implemented using FastAPI, a modern Python web framework, and the frontend is built with HTML, CSS, and JavaScript. This paper outlines the development process, including system architecture, design, and implementation details.

II. System Architecture

The Rubik's Cube Timer application follows a client-server architecture, with the frontend and backend components deployed on a local machine. The frontend is served via Nginx, while the backend handles API requests and data persistence using a JSON file.

A. Frontend

The frontend of the application is built using HTML, CSS, and JavaScript. It provides a responsive and intuitive user interface for timing solves, viewing statistics, and managing user accounts. The frontend communicates with the backend via RESTful API calls.

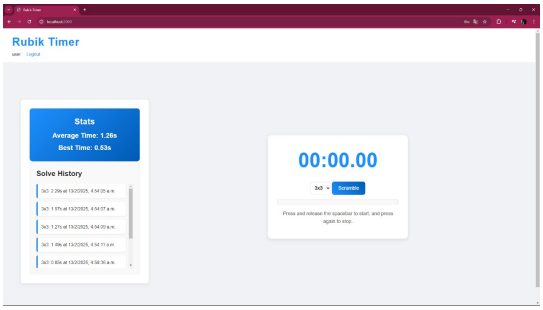

Fig. 1. Frontend: Home Screen

B. Backend

The backend is implemented using FastAPI, a modern Python web framework. It handles user authentication, solve timing, and statistics tracking. Data is persisted in a JSON file, ensuring simplicity and ease of deployment.

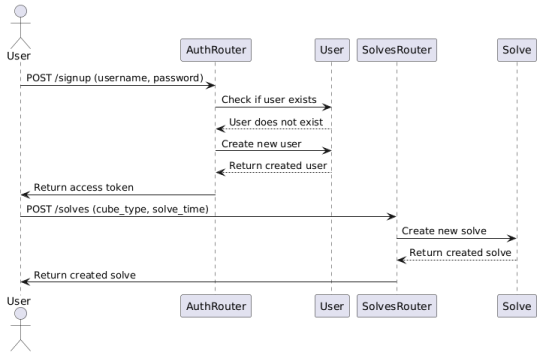

Fig. 2. Backend: Sequence Diagram for User Registration and Solve Creation

III. Business Model Canvas

The following Business Model Canvas outlines the key aspects of the Rubik's Cube Timer application:

Key Activities	Key Resources	Channels
• Implementing a database. • Adding more cube types. • Incorporating new statistics. • User engagement and feedback collection.	• FastAPI framework.	• Web platform.
Value Propositions	**Customer Relationships**	**Customer Segments**
• Precision timer for speedcubing. • Automatic scramble generation for different cube types • Automatic stadistics update.	• Feedback loop for continuos improvement.	• Casual speedcubers. • Casual cubers.

Fig. 3. Business Model Canvas for the Rubik's Cube Timer Application

IV. Design and Implementation

A. Frontend Design

The frontend is designed to be responsive and user-friendly. It consists of several screens, including the home screen, cube selection screen, timer screen, solve history screen, and statistics screen. Each screen is built using HTML and CSS, with JavaScript handling user interactions and API calls.

B. Backend Design

The backend is designed to be modular and scalable. It consists of several components, including the authentication service, solve service, scramble service, and database layer. The backend follows the principles of clean architecture, ensuring separation of concerns and ease of maintenance.

V. Analysis of SOLID Principles

The implementation of the Rubik's Cube Timer application adheres to the SOLID principles as follows:

- **Single Responsibility Principle (SRP)**: Each class and function has a single responsibility.
- **Open/Closed Principle (OCP)**: The system is open for extension but closed for modification.
- **Liskov Substitution Principle (LSP)**: Derived classes can substitute base classes without altering the behavior.
- **Interface Segregation Principle (ISP)**: The backend services are designed with specific interfaces.
- **Dependency Inversion Principle (DIP)**: High-level modules depend on abstractions rather than concrete implementations.

VI. Conclusion

The Rubik's Cube Timer application successfully addresses the needs of speedcubers by providing a reliable and user-friendly tool for timing solves and tracking progress. The application's backend, built with FastAPI, is modular, scalable, and adheres to SOLID principles, ensuring maintainability and extensibility. The frontend offers an intuitive interface for users to interact with the application. Future work could include adding support for more cube types, improving the user interface, and integrating with external databases for enhanced data persistence.

References

[1] FastAPI Documentation. Available: https://fastapi.tiangolo.com/