Form Application with Python and Java Backends for Response Management and Authentication

Daniel Santiago Pe´rez
Faculty of Systems Engineering
Universidad Distrital Francisco Jose de Caldas
dsperezm@udistrital.edu.co

Jaider Santiago Avila
Faculty of Systems Engineering
Universidad Distrital Francisco Jose de Caldas
snmendivelsom@udistrital.edu.co

***Abstract*—The context of this work focuses on the growing need for cloud-based applications that allow users to create and manage forms efficiently. With the rise of productivity tools like Google Forms, there has been a demand for platforms that facilitate the creation of surveys, contact forms, and questionnaires for a wide and diverse audience. The secure handling of user data and flexibility in form types are key aspects to ensure usability and security in these applications.**

The proposed solution involves developing a form application with a bifurcated backend: Python (using FastAPI/Flask) and Java (using Spring Boot). This application allows user authentication, form creation and management with various question types, and secure storage of responses. Instead of a traditional relational database, persistence is achieved through JSON files that store users, forms, and responses. Additionally, features such as JWT-based authentication, email notifications, and a premium subscription system (offering functionalities such as response export and advanced form management) have been implemented.

Throughout development, the system demonstrated robust performance and scalability by leveraging design principles (SOLID) and patterns (Singleton, Factory Method, Strategy) to ensure a maintainable architecture. Tests performed with JSON file storage confirm the system's reliability for lightweight deployments, positioning it as a competitive alternative for production environments in scenarios where simplicity is paramount.

I. Introduction

In today's digital era, web applications have become central to both personal and professional activities. Among these, form creation applications are critical for data collection, surveys, and evaluations. Although platforms like Google Forms dominate the market, there exists a significant need for more flexible and customizable systems. Such systems must accommodate diverse user requirements while maintaining high standards of data security and performance.

This project aims to develop a form application that mirrors the core functionalities of commercial platforms but introduces a modular backend architecture. By employing two different technologies—Python and Java—we leverage Python's rapid development and ease of handling data processing alongside Java's enterprise-grade robustness and scalability. The application incorporates user authentication, JWT-based security, and real-time response management while providing a foundation for additional features such as notifications and premium services.

II. Methods and Materials

A. System Architecture

The solution is based on a bifurcated backend architecture:

Python Backend:

Developed using FastAPI (and alternatively Flask for certain modules), it focuses on response processing and report generation. Python's simplicity and powerful libraries (e.g., Pandas for data analysis) allow for the rapid development of RESTful APIs.

Java Backend:

Built with Spring Boot, it manages user authentication, form creation, and data persistence. Utilizing Spring Security, the Java backend ensures robust security and modular code design.

B. Data Persistence with JSON

Rather than using a traditional relational database, this project employs JSON files for data storage. Three primary JSON files are used:

users.json: Stores user data and authentication details.

forms.json: Contains form metadata and details, including questions.

responses.json: Archives the responses linked to each form.

This approach simplifies deployment and reduces overhead, making the system lightweight and easy to maintain while ensuring data can be managed in a structured format.

C. Core Functionalities

User Authentication:

Using JWT-based authentication, the system ensures secure access. Only authenticated users can create, modify, and view forms, while responses can be submitted by both authenticated and guest users.

Form Management:

Users can create custom forms comprising various question types (text, date, time, multiple choice). A flexible question creation process is supported by a Factory Method pattern, allowing easy extension for new question types.

Response Handling and Notifications:

Form responses are processed in real-time. Email notifications via SMTP inform form creators of new responses. Additionally, a premium subscription model enables features such as data export in CSV/PDF formats.

D. Design Patterns and SOLID Principles

The application's design is underpinned by fundamental software engineering principles:

SOLID Principles:

Single Responsibility: Each class/module (e.g., controllers, models, utilities) focuses on a single responsibility.

Open/Closed: The system is designed for extension (e.g., new question types via Factory Method) without modifying existing code.

Liskov Substitution: Subclasses (e.g., TextQuestion, DateQuestion) reliably replace their abstract base (Question) in all contexts.

Interface Segregation: Interfaces (like ValidationStrategy) ensure classes only implement relevant methods.

Dependency Inversion: High-level modules depend on abstractions (e.g., injecting ValidationStrategy in questions), facilitating loose coupling.

Design Patterns Implemented:

Singleton: Used in the FileManager classes (in both com.app.services and com.app.utils) to ensure a single instance for file operations.

Factory Method: Employed in the QuestionFactory to create different question types dynamically.

Strategy: Applied for validation logic, where various ValidationStrategy implementations (DateValidation, TextValidation, etc.) validate responses based on question type.

III. Results and Discussion

A. Implementation Successes

Robust Architecture:

The bifurcated backend approach has enabled the system to harness the strengths of both Java and Python. The Java backend provides a secure and scalable environment for user and form management, while the Python backend efficiently handles response processing and report generation.

Effective Use of Design Patterns:

The use of SOLID principles and design patterns has resulted in a modular, maintainable codebase. New question types or validation rules can be added with minimal impact on the overall system, demonstrating the extensibility of the design.

Performance with JSON Persistence:

The implementation of data storage using JSON files has proven sufficient for the project's scope. The lightweight persistence layer supports quick prototyping and easy maintenance, making it ideal for scenarios where a full-fledged database might be an overkill.

B. Challenges and Improvements

Integration Between Backends:

One of the primary challenges was ensuring seamless communication between the Java and Python backends. This was addressed by designing well-defined RESTful APIs and adhering to clear data exchange formats (JSON).

Concurrency in File Operations:

Managing concurrent access to JSON files required careful design in the FileManager implementations. Thread-safety and data integrity were ensured through proper handling of file I/O operations.

Future Enhancements:

Potential improvements include migrating to a more robust storage solution (such as a lightweight NoSQL database) if system demands increase, as well as integrating a full-featured front-end to replace the current console-based interface.

IV. Conclusion

This work has demonstrated the feasibility of developing a robust form application leveraging dual backends in Python and Java. By combining the rapid development capabilities of Python with the enterprise robustness of Java, the project has achieved a modular and secure system for form creation, response management, and user authentication.

The careful application of SOLID principles and design patterns such as Singleton, Factory Method, and Strategy has resulted in a codebase that is both maintainable and extensible. Performance tests with JSON file persistence confirm that the system is well-suited for lightweight deployments, with the potential to scale or migrate to other storage systems as needed