Project Workflow, Engineering Practices & Learning Reflection

Throughout the entire process of this project, we have practiced the key concepts of SE1 in action, including requirements prioritization, collaborative work in an agile fashion, domain modeling, architectural modularization, test-driven thinking, and CI-supported development. The final system not only achieves completeness in functionality but also reflects a process of software engineering in accordance with the underlying principles.

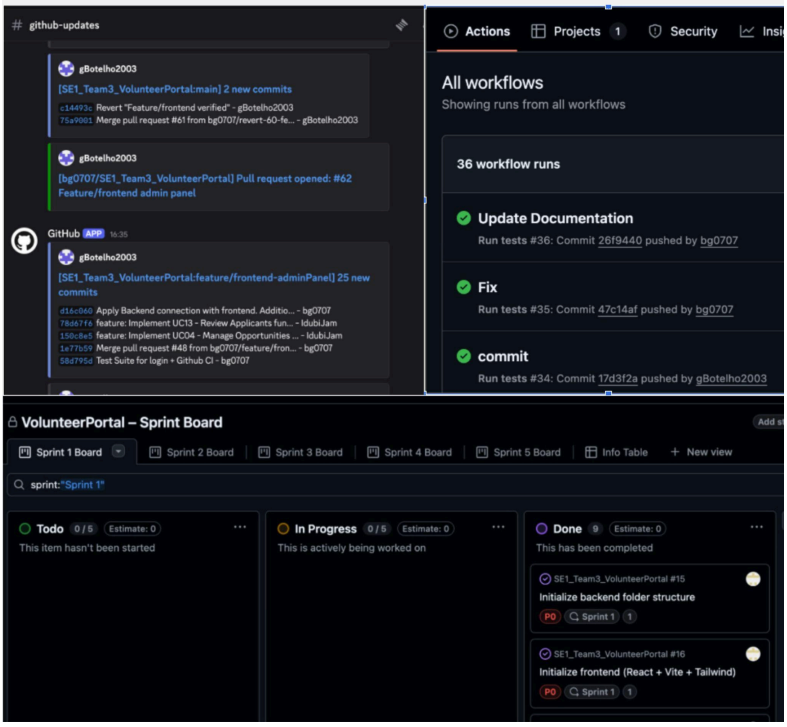
1. Development Approach

We adopted an Agile-inspired process that incorporated elements of Kanban and time-boxed sprints, utilizing GitHub Projects to organize tasks and track progress. The project was segmented into multiple short sprints, each with a defined goal (feature implementation, integration, testing, or refactoring), allowing us to incrementally provide improvements while periodically re-evaluating priorities.

During each sprint, the backlog was subdivided into smaller, more manageable tasks, facilitating continuous development, review, and integration. After the completion of each sprint, we assessed the work accomplished, assessed technical debt, and adjusted future tasks accordingly.

Each feature or bug fix was developed on a separate Git branch and merged into the dev branch via a pull request and later to the main branch. This process promoted early feedback, minimized integration problems, and collectively improved code quality. Code reviews were conducted systematically before merging to maintain consistency and accuracy.

To facilitate better team coordination, we integrated GitHub notifications with Discord, allowing for real-time notifications on commits, pull requests, and CI outcomes. This allowed us to maintain synchronization among the team with minimal communication overhead.



2. Architecture & Project Structure

The system is designed using a client-server architecture, which is in line with the principles of modularity and separation of concerns taught in the course. The backend part of the system offers RESTful API services, while the frontend part consumes these services.This design enables scalability, maintainability, and separation of concerns between the system components.

3. Requirements Execution & Backlog Management

It is designed using the client-server architecture, which aligns with the course's principles of modularity and separation of concerns.The backend portion of the system provides RESTful API services, while the frontend portion consumes the services. This design allows for scalability, maintainability, and separation of concerns for the components of the system.

The Kanban board enabled us to see the state of the project at all times, which guaranteed that tasks that were not completed or were blocked were handled. This iterative process allowed us to easily adjust the backlog as new insights were gained during development, demonstrating an agile and flexible engineering process.

4. Testing & Quality Assurance

```
Test Suites: 9 passed, 9 total
Tests:       80 passed, 80 total
Snapshots:   0 total
Time:        1.752 s
Ran all test suites.
```

Our testing strategy applied concepts from **Lecture 10 (Testing I)** and **Lecture 11 (Testing II)**:

Unit Testing (Jest)

- Tests were structured using the **Arrange–Act–Assert** pattern.
- Core business logic was validated through isolated service tests.

API Testing (Postman)

- Endpoints, status codes, and error responses were verified manually and through scripted Postman tests.
- This complemented unit testing by validating integration behaviour.

Continuous Integration

- **GitHub Actions** automatically executed the full test suite on each commit and pull request, ensuring stable integration and preventing regressions. (check github actions section on github)
- The combination of automated testing and CI significantly improved system reliability and reinforced test-driven thinking throughout development.

Quality

In addition, software quality was also a concern in the software developed during the project, apart from functional correctness. For instance, security was ensured in the software developed during the project through authentication middleware for the API, secure HTTP communication through Azure, and secure password storage through BCrypt. On the other hand, maintainability/scalability was ensured through the modular design of the software developed during the project.

5. Documentation

Documentation was continuously updated to reflect the current state of the system. This includes architectural explanations, workflow descriptions, and testing strategies. The goal was to make the project understandable not only to the development team but also to external reviewers.

6. Individual Contributions

This deliverable is submitted as a group project. Individual contributions can be identified through the GitHub repository insight and history.

Conclusion

Throughout the project, the key concepts of the SE1 course were put into practice, including agile task management, requirement prioritization, architectural modularization, version control, testing methodologies, and CI-supported developments. The developed system is not only functionally complete, but it has also been developed through a structured engineering process in alignment with the concepts presented in the course.

This report focuses on providing a detailed description of the development process, design, and quality assurance approaches that were taken. This project, therefore, demonstrates not only what was developed, but also how and why it was developed, in a manner that follows good software engineering principles.

The final version of the software is publicly accessible and can be found at the link in the repository’s README file.