Portfolio - Maintenance Ticket Management

ESOF423 - Software Engineering Applications

Spring 2024

Group 2: Seth Barrett, Xander Burger, Sam Jacobsen, Erik Johns, and Emma Keeley

I. Program

Our project can be found at the following GitHub Repository:
https://github.com/423s24/Group_2

II. Teamwork

Team Member 1 - Xander

I worked mostly on the user side of the application, my logged hours estimated between 11 and 13 out of class hours per sprint, totalling between 40 - 45 hours spent on this project.

Primary Contributions

- Login and Registration forms and functionality on the user side.
- Created and worked on the maintenance request from.
- Loading and error handling on maintenance requests from.
- Add accessible features to forms.
- Created user homepage that grabs previously submitted tickets from firebase.
- Design user interface for login, registration, maintenance request, and home pages.
- Implemented automatic testing with Jest.
- Created CI workflow on github.

Team Member 2 - Sam

As Team Member 2, I focused on developing the messaging application, along with aspects of both the frontend and backend for the ticket submission form. Initially, I was estimated to work 60 hours on the entire project. As of now, I estimate that I have worked approximately 65 hours.

Primary Contributions

- Linking data from the frontend maintenance ticket form to the backend firestore database
- Worked on parts of the frontend form design and style.
- Worked on user and admin message frontend (style and design)
- Worked on the backend of messaging application data structure.
- Linked frontend and backend of messaging application.

Team Member 3 - Erik

As team member 3, I would say that I most certainly took charge of hosting. While the task seems simple, it took extensive effort to get the two projects hosted without errors. My time estimate for hours worked was 36 hours (12 hours per sprint). I actually contributed 40.5 hours of work outside of class.

Primary Contributions

- Designed Login, Register, and Forgot Password page on the admin side.
- Added functionality for login, register, and forgot password on the admin
- side using Firebase.
- Created protected routes so that users not logged in cannot access any private data.
- Designed and implemented the loading screen on the user and admin side.
- Created Firebase Firestore Database to store tickets and information about users.
- Created ticket view which shows detailed information about each ticket.
- Implemented seeing tickets at the same address and tickets submitted by the same user into the ticket view.
- Implemented editing a ticket within the ticket view.
- Managed hosting through Firebase Hosting for the user side, admin side, and documentation website.
- Designed and created the documentation website.

Team Member 4 - Seth

As Team Member 4 I focused on creating the admin-frontend page and working on this side mostly. I worked on little things on the user-frontend side. I worked about 12 hours a sprint in and out of class and a total of 40 in total.

Primary Contributions

- Admin-frontend page
- Worked on setting up search system on admin side
- Worked on setting up filtering and sorting options
- Created the ticket display
- Styling on the admin side
- Debugging
- Image submission and display for user and admin
- opt-in/out emails on account creation
- Forgot password page for users

Team Member 5 - Emma

I split my time between the frontend and the backend side. My logged hours estimated between 11 and 14 out of class hours per sprint, totalling between 40 - 45 hours spent on this project roughly.

Primary Contributions

- Email Notifications
- Worked on User Firebase Firestore
- Debugging

- Worked on ticket management features
- Extensive User Testing
- Diagramming and Documentation

III. Design Pattern

1. Module Pattern

The application is structured into modular components, each responsible for a specific part of the application's functionality. This pattern helps in encapsulating functionality, making the codebase more maintainable and scalable. Each React component acts as a module, and methods like fetchTickets or fetchUsers are encapsulated within these modules.

2. Singleton Pattern

Firebase instances such as auth and db from the Firestore database are likely imported and used across the application. These instances are typically initialized once and reused throughout the application, which is characteristic of the Singleton pattern. This ensures there is a single point of access to these resources.

3. Observer Pattern

The use of Firebase's real-time database features (like onSnapshot) subscribes components to updates in the database, making them observers of the data they depend on. When data changes, Firebase notifies these subscribed components, effectively updating the UI in real time.

4. Decorator Pattern

React components are often wrapped or enhanced with additional functionality. For example, higher-order components (HOCs) or hooks like useNavigate or useAuthState enhance component functionality without modifying their actual implementation. This can be seen as an application of the Decorator pattern, which allows behavior to be added to individual objects, either statically or dynamically.

5. State Pattern

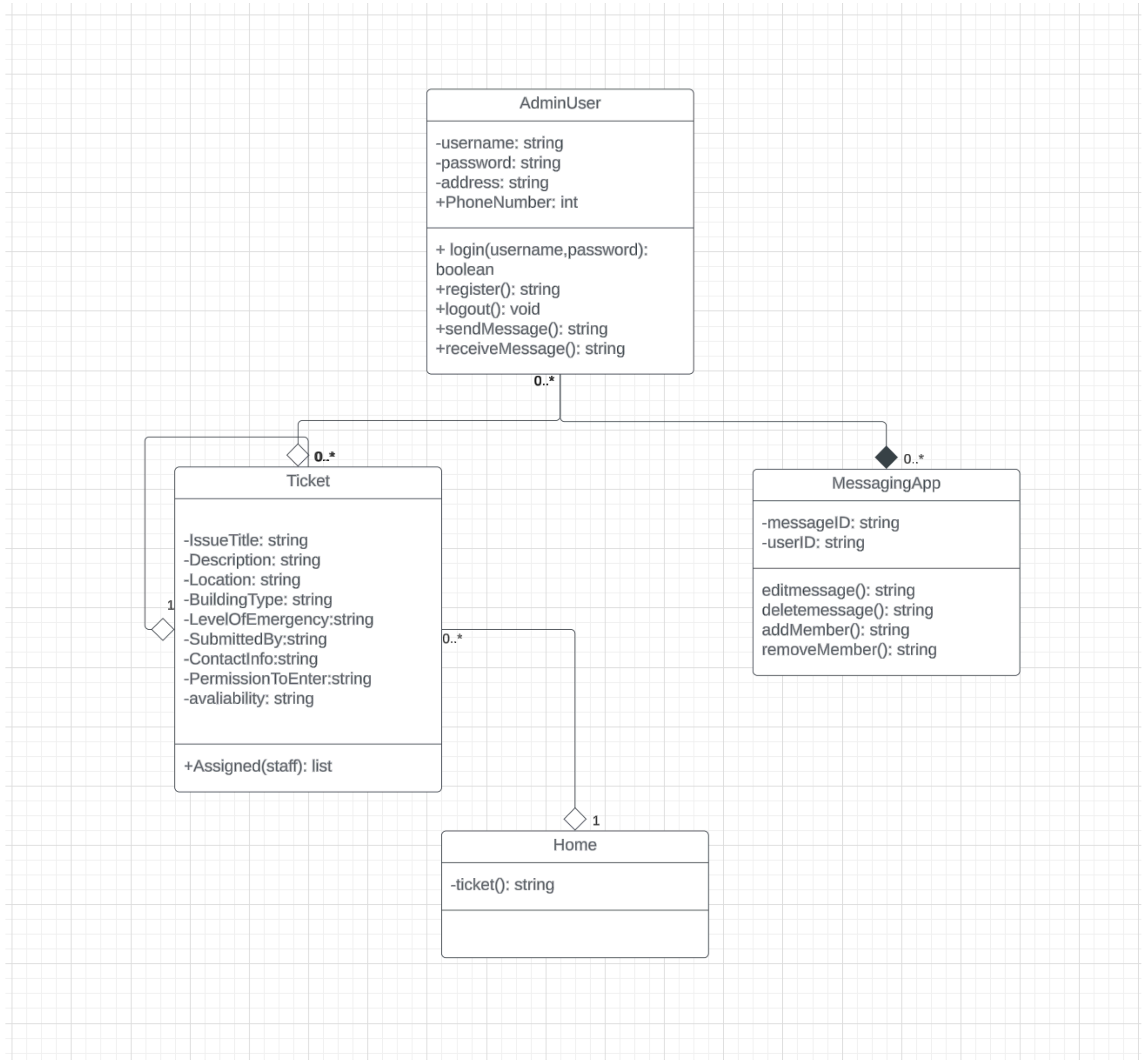
The extensive use of state management within React components, where the component's behavior is influenced by its current state, reflects the State pattern. State hooks like useState manage the state within each component, and changes to this state dictate how the component behaves and renders.

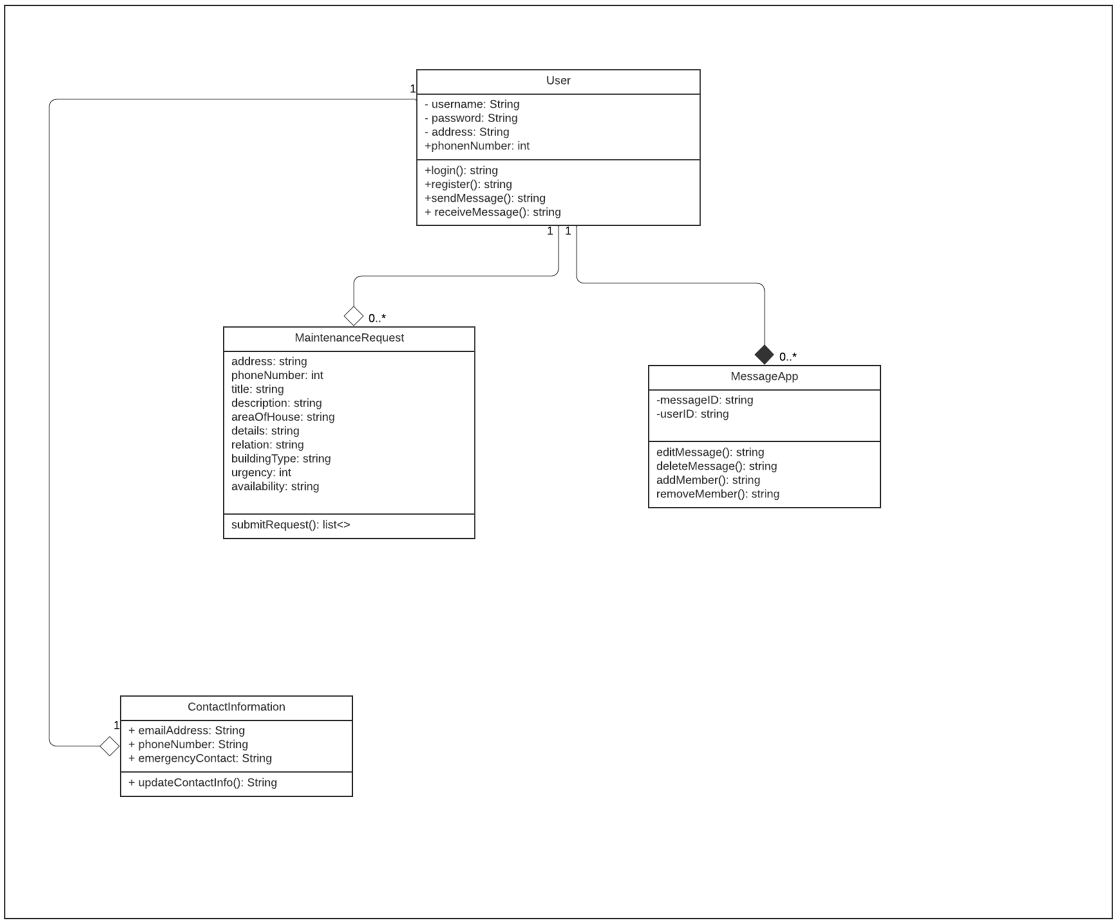
IV. Technical Writing

Our project documentation can be found here:
Documentation Website

V. UML

The UML diagrams below demonstrate the relationships for both the user side of the application and the admin side of the application.





VI. Design Trade-offs

admin and user web apps

We chose to create two different web apps one that handles the admin side and one that handles the tenant maintenance request.

Benefits:

This allowed us to separate our concerns in developing the software. Not having to handle user roles directly in our software but rather on authentication to each individual site. This makes our code cleaner and easier to understand

Trade-off

This also meant that we had more code to write and manage causing us to spend time on syncing changes between the admin and user web applications.

VII. Software Development Lifecycle Model: Agile Methodology

Working in the Scrum framework offers numerous benefits for software development teams. By breaking the project into manageable sprints and working off a prioritized product backlog, Scrum promotes focus and efficiency. The iterative nature of Scrum allows for quick adaptation to changing requirements and feedback, ensuring that the team remains responsive to stakeholder needs. Additionally, the use of daily stand-up meetings, sprint planning sessions, and retrospectives fosters clear communication and collaboration within the team. The use of a burndown chart provides a visual representation of progress, helping to keep the team motivated and on track towards their goals. Overall, Scrum provides a structured and organized approach to development, enabling teams to deliver high-quality products in a timely manner while maintaining flexibility and adaptability.