Software Design Document

Group 4

Calm Chores

Software Engineering
Fall 2024

CS673

BOSTON UNIVERSITY

Keshav Saraogi
Chun Qin Huang
Anup Sindagi
Xiaoyue Zhang
Fernanda Nano

High-Level Architecture of the Project:

The project will be divided into two sub-projects, frontend and backend. The frontend will feature a mobile application, which will be built using React Native and Expo Go as the sandbox dev environment to develop and test the application. The backend will be a hosted public API endpoint with secure authentication measures implemented using Firebase Auth, Firestore for the database, and node.js as the backend language.

FRONTEND:

- **UI Components:** The UI will feature many custom components such as buttons, links, date picker, time picker, navigational tabs, and buttons. All the UI components will be coded with a mix of React Native components and Expo Go components for maximum compatibility between different versions of iOS and Android. The components will be styled using NativeWindCSS, which is the modified version of TailwindCSS for the React Native application.
- **Navigation Flow:** The app will feature many screens for navigation between the different features and options. The app will start with a login and signup screen, which will be authenticated using Firebase Authentication. Once the user is logged in or signed up, the user will enter the main app. The main app will feature a home screen, tasks screen, profile screen, add house screen, add task screen, task info screen, and more. All these screens will feature different elements based on whether the user has added or joined a house the tasks assigned to the user or if the user is the owner of the house group. The navigation will be handled by the Expo Router, which by default features a screen-based routing, meaning the structure of the folder of the main "app" will be the routes for each individual screen.
- **State Management:** The app's state will be managed by using the React hooks such as useState, useEffect, and React Context API to hold all the necessary states needed across different screens. There will be an AuthContext file that will manage the user authentication tokens and house details of the users, based on these details different components will display customized information. The AuthContext will be synced with the main Firestore database and pull all the essential changes as soon as they happen from any of the users in the house.

User Interface Design

- **Wireframes:** All of the navigation and layout of the screens will be first done in Figma, which is a tool for designing UI/UX before developing the app. This will include the navigation between the different screens, the main layout of the app, and the internal layout for different functionalities within the app.
- **UI Design Principles:** We will be using the built-in TailwindCSS feature to enable Dark Mode and Light Mode, the modes of the UI will be changed based on the OS settings set

by the user. The UI will feature a simple styling with a blue theme for both light and dark modes, which would help enhance readability and usability across the app. We will be using the standard icon pack provided by the Expo component library across the app to maintain consistency.

BACKEND:

The backend of the project consists of a client-server architecture that handles the interaction between the user and the application. The Client layer is a mobile application that interacts with an Express.js server that is responsible for handling all the HTTP requests and the business logic, including with the interaction with the Firebase database. There is also a service later that is responsible for business logic and the core functionality of the application. In addition to this, there is a Data Layer that stores the data for all users, chores, maintenance requests and other important information.

Presented below is a high-level architecture diagram of the application along with an Entity Relationship Diagram for the application.

Backend

- **RESTful API**: The REST API service for the project has been developed in stages to ensure that there is clear communication between the client and the server. An example has been provided below:

  Tasks Management Endpoints:
  - POST /addChore: Adds a new chore to a specific house, with attributes such as chore name, assigned user, due date, and recurrence.
  - PUT /updateChore/:chore_id: Updates an existing chore with new information (e.g., due date, status, assigned user)
  - DELETE /deleteChore/:chore_id: Deletes a specific chore based on its unique identifier.
  - GET /chores/:house_id: Fetches all chores for a given house ID, with optional filters (e.g., status or assigned user)

- **Business Logic**: The application's business logic focuses on implementing Task Rotation, Verification, Vacation Mode, and Rating Calculation.
  1. Task Rotation: Reassign tasks to another roommate if a user is on vacation mode.
  2. Verification: Confirm the action of a chore with user roles and permissions.

3. Vacation Mode: Activate / Deactivate a vacation mode for a user based on their date inputs. This also reassigns tasks based on availability during the mode.
4. Rating Calculation: Calculate ratings based on chore completion, ignored tasks, maintenance handling, and timely completion.

Database Design

- **Schema Design**: Key entities are presented in the Entity Relationship Diagram attached as a diagram in this document.

Security Considerations

- **Authentication and Authorization**: Used Firebase authentication setup to ensure only authorized users access their account.

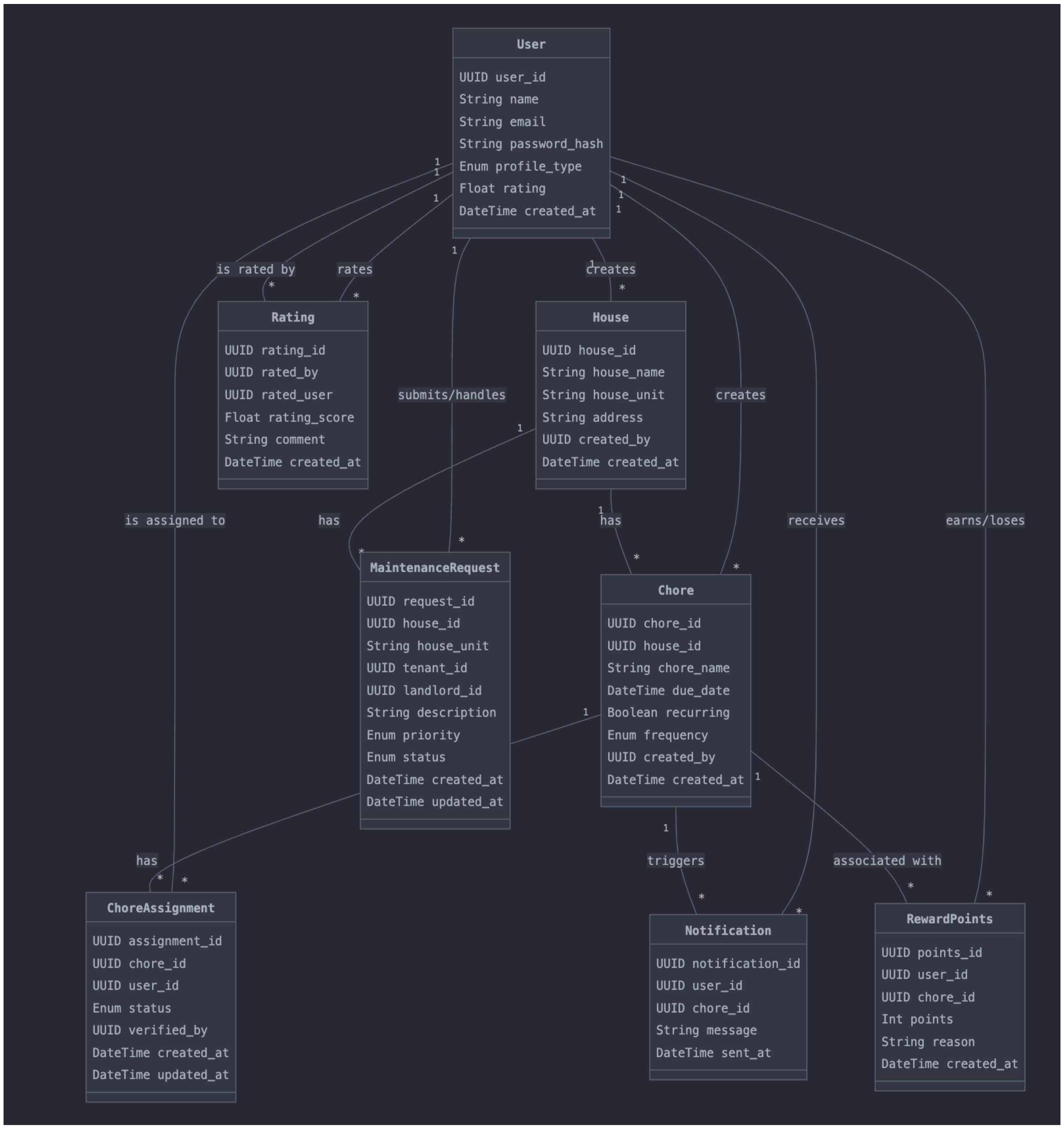
Quality Assurance and Testing

- **Test Cases**: Unit tests, integrated tests, and end-to-end tests

Deployment Plan

- **Deployment Strategy**: Verison control setup and branching on GitHub
- **Rollback Strategy**: We will revert a version of the deployment in case of failure.

Entity Relationship Diagram


User
UUID user_id
String name
String email
String password_hash
Enum profile_type
Float rating
DateTime created_at
Rating
UUID rating_id
UUID rated_by
UUID rated_user
Float rating_score
String comment
DateTime created_at
House
UUID house_id
String house_name
String house_unit
String address
UUID created_by
DateTime created_at
MaintenanceRequest
UUID request_id
UUID house_id
String house_unit
UUID tenant_id
UUID landlord_id
String description
Enum priority
Enum status
DateTime created_at
DateTime updated_at
Chore
UUID chore_id
UUID house_id
String chore_name
DateTime due_date
Boolean recurring
Enum frequency
UUID created_by
DateTime created_at
ChoreAssignment
UUID assignment_id
UUID chore_id
UUID user_id
Enum status
UUID verified_by
DateTime created_at
DateTime updated_at
Notification
UUID notification_id
UUID user_id
UUID chore_id
String message
DateTime sent_at
RewardPoints
UUID points_id
UUID user_id
UUID chore_id
Int points
String reason
DateTime created_at
is rated by
rates
creates
submits/handles
creates
is assigned to
has
has
receives
earns/loses
has
triggers
associated with
1
*