AKAASH DHAYALAN (A0272377W)

SADHANASRI DASARATHAN (A0282575U)

Level of Achievement **5**
Project Scope 5
Solution Description **5**
Problem Motivation 5
Basic Solution Description & Impact 5
Features **6**
User Stories **6**
System Design **8**
Unified Modeling Language (UML) class diagram 8
Navigation Diagram 10
Technologies used **11**
React Native 11
Firebase 11
Accessing the project **12**
Usage **13**
Accessing the app 13
User Authentication 13
Calendar View 13
Task Management & Auto Scheduling 13
Asynchronous Storage 14
Offline Functionality 14
Themes 14
Free Time Slot management 14
User Authentication **15**
Components 15
LoginScreen.js 15
ForgotScreen.js 15
Key Functionalities 15
Sign In/Sign Up 15
Password Reset 15
User Session Management 16
Authentication Flow 16
Calendar View **19**
Task Management **20**
Breakdown of Screens 20
HomeScreen.js 20
TaskScreen.js 20
ViewScreen.js 20
Functionalities of the Code 21
HomeScreen.js 21
TaskScreen.js 21
ViewScreen.js 21
Task Management Flow 21
Time Slot Management **25**

Components 25
time slotScreen.js 25
Addtime slotScreen.js 25
time slotScreen.js 25
Key Functionalities: 25
Time Slot Format: 25
Addtime slotScreen.js 25
Key Functionalities: 25
Auto scheduling 28
Components: 28
Functionality: 28
Code Flow 28
Navigation Bar 30
Components: 30
Functions: 30
Asynchronous Storage 31
Functions used 31
Flow of Asynchronous Storage 31
Offline Accessibility 33
Components Used 33
Flow of Offline Accessibility 33
Themes 35
Components and Flow 35
1. App.js 35
2. AppNavigator.js 35
3. Index.js (Themes) 35
4. ThemeContext.js 35
Rendering Screens and Toggling Themes 36
User Testing 37
Version Control 41
Development Plan 42
Completed sprint logs 42
Milestone 1 Sprint Goal 42
Milestone 2 Sprint Goal 45
Planned Sprint Logs 47
Milestone 3 Sprint Goals 47
Software Engineering Principles 50
1. Modularity 50
2. Persistence 52
3. Reusability 52
4. Security and Environment Configuration Management 53
5. Material Design Principles 54
Testing & CI/CD 55
Future Plans 56

Sleep tracking using Machine Learning 56
Reminders 56
Synchronisation 56
User collaboration 56
Time Management Analytics 56
Others 57
Challenges Faced **58**
Choosing the tech stack 58
Steep learning curve 58
Complexity of User Interface (UI) Design 58
Need for Large Language Models (LLMs) in the auto scheduling feature 58
Current Limitations **59**
Uneven Auto Scheduling 59
Inefficient sign in 59
Insufficient Machine Learning Model (Anticipated) 59
Project Management **60**
Project logs **60**

Level of Achievement

Apollo 11

Project Scope

Our app, TaskMaster is an app that aims to help university students with their time management skill, to ensure that the plans that they make eventually become progress. Hence, our tagline is, where plans become progress.

Solution Description

Problem Motivation

As students in NUS, we have experienced times where we have had a significant amount of workload piled up within one or two days. While we may have time management skills, we may be too distracted with our work to consider a well-planned schedule to evenly distribute our workload. Eventually, this kind of situation can lead to stress and eventually burn out. Hence, this calls for a mobile application to assist students in managing their workload and time.

Basic Solution Description & Impact

Our app, TaskMaster, has core features such as creating tasks & deleting tasks, auto-scheduling (to be incorporated in the future), sleep tracker which ensures that the user has sufficient rest despite the hectic schedule. Hence, our app will have a significant positive impact as it ensures that our users' workload will be evenly distributed and. Having an efficient schedule helps our users to not burn the midnight oil, which also gives them a significant peace of mind knowing that TaskMaster reinforced to them that their life is in control.

Features

Feature 1 (core): Registration and login of users

Feature 2 (core): Automatic scheduling of tasks

Feature 3 (core): Task management Creating, repeating, updating and deletion of tasks, events and assignments with deadlines)

Feature 4 (extension): Calendar View
Feature 5 (extension): Sync with timetable from NUSmods

Feature 6 (extension): Reminders of upcoming tasks and events

Feature 7 (extension): Time management analytics
Feature 8 (extension): Customization Options Colour schemes, fonts, etc)
Feature 9 (extension): Offline Accessibility
Feature 10 (extension): Ensures routine sleep schedule and sufficient rest
Feature 11(extension): Collaboration with other users
Feature 12(extension): Asynchronous storage

Feature 13(extension): Navigation Bar

Feature 14(extension): Free time management

User Stories

1. As a student who wants a personalised and secure experience, I want to be able to sign up, log in, and customise the app's appearance (such as switching between light and dark themes) according to my preference.

   Features:

   1) Registration and login of users - Feature 1
   2) Customization Options - Feature 8
   3) Navigation Bar - Feature 13z

2. As a student aiming to efficiently manage my schedule and tasks, I want the app to automatically create, update, and delete tasks, events, and assignments based on my availability and preferences. I also want to view my tasks in a calendar view.

   Features:

   4) Automatic scheduling of tasks - Feature 2
   5) Task management Creating, repeating, updating and deletion of tasks, events, and assignments with deadlines) - Feature 3

6) Calendar View - Feature 4
7) Free time slot management - Feature 14

3. As a student who may lose internet connection occasionally, I want the app to store my tasks and events offline and sync them when the connection is restored. I also want faster sign-in after the initial sign-in.

   Features:

   8) Offline Accessibility - Feature 9
   9) Asynchronous storage (faster sign in) - Feature 12

4. As a student seeking to streamline my academic and social schedules, I want to sync my timetable with NUSMods, collaborate with friends to find common free times, and receive reminders for scheduled tasks through my phone's alarm function.

   Features:

   10) Sync with timetable from NUSMods - Feature 5
   11) Reminders - Feature 6

System Design

Unified Modeling Language (UML) class diagram

This UML class diagram illustrates the structural relationships between the primary entities in a user scheduling and analytics application. The diagram provides a clear visual representation of how the `User`, `Schedule`, `Task`, `Event`, and `Analytics` classes interact with one another. Specifically, it highlights the following relationships:

- Each `User` has a one-to-one association with a `Schedule`.

- The `Schedule` is composed of multiple `Task` and `Event` instances, establishing one-to-many relationships.

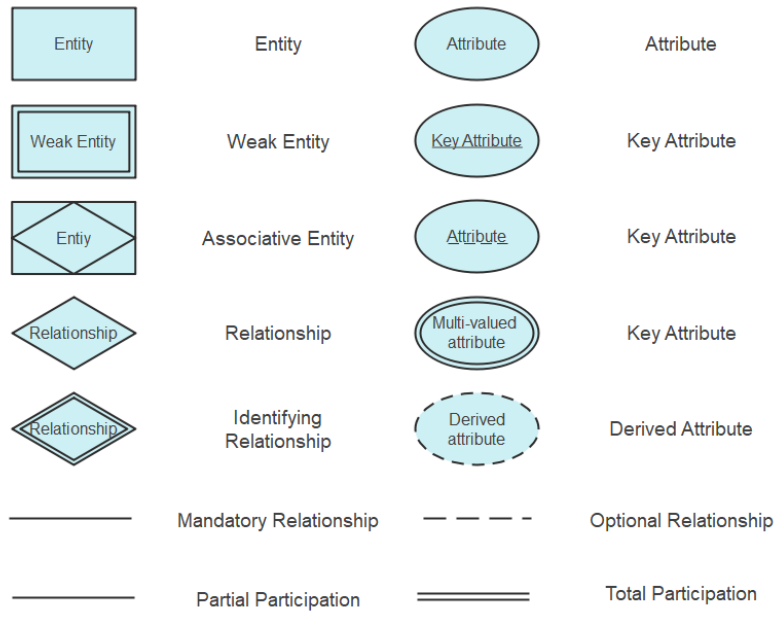
- Both `Task` and `Event` classes maintain a many-to-one association with the `User` class, signifying that each task and event is linked to a specific user.

- The `Analytics` class is associated with the `User` class on a one-to-one basis, capturing various productivity metrics for the user.

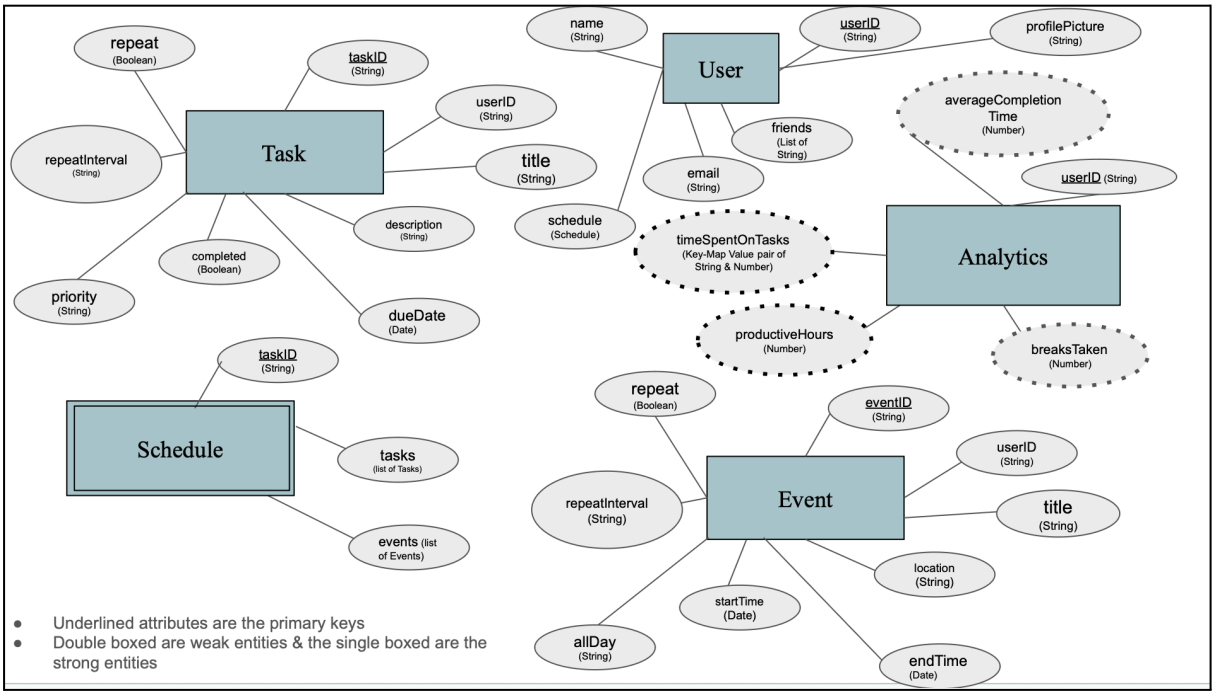
This diagram serves as a blueprint for developers to understand the data model, facilitating the implementation and maintenance of the application's functionality

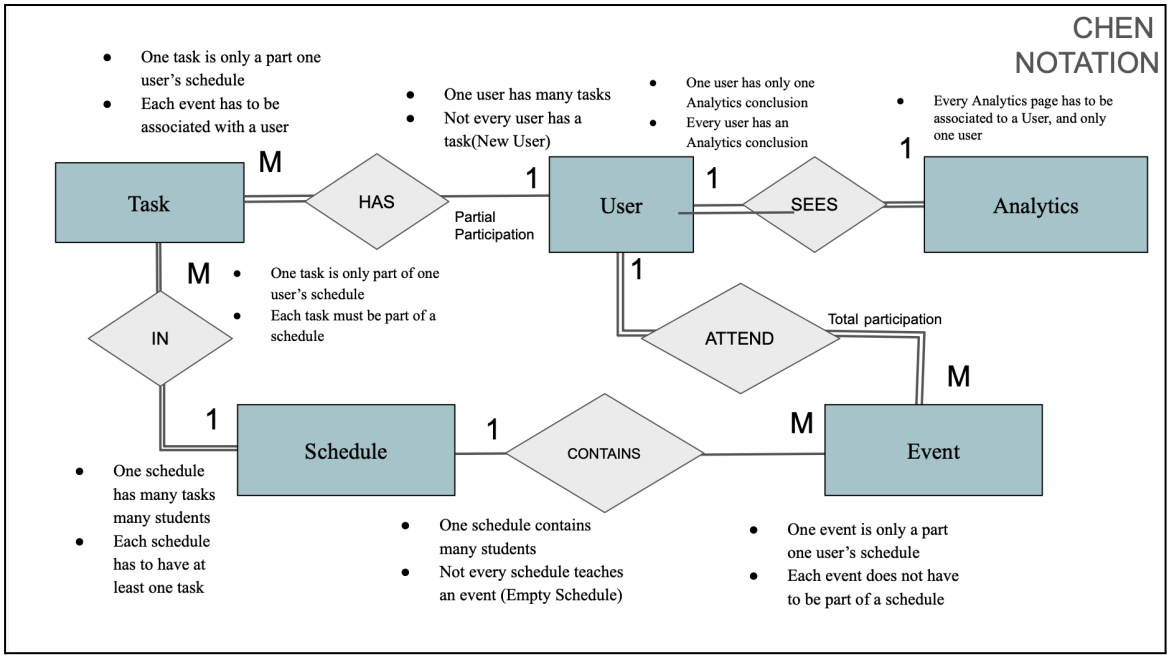
Chen Notation Class Diagram for Database

Legend:



Entities & Attributes (and its datatypes)

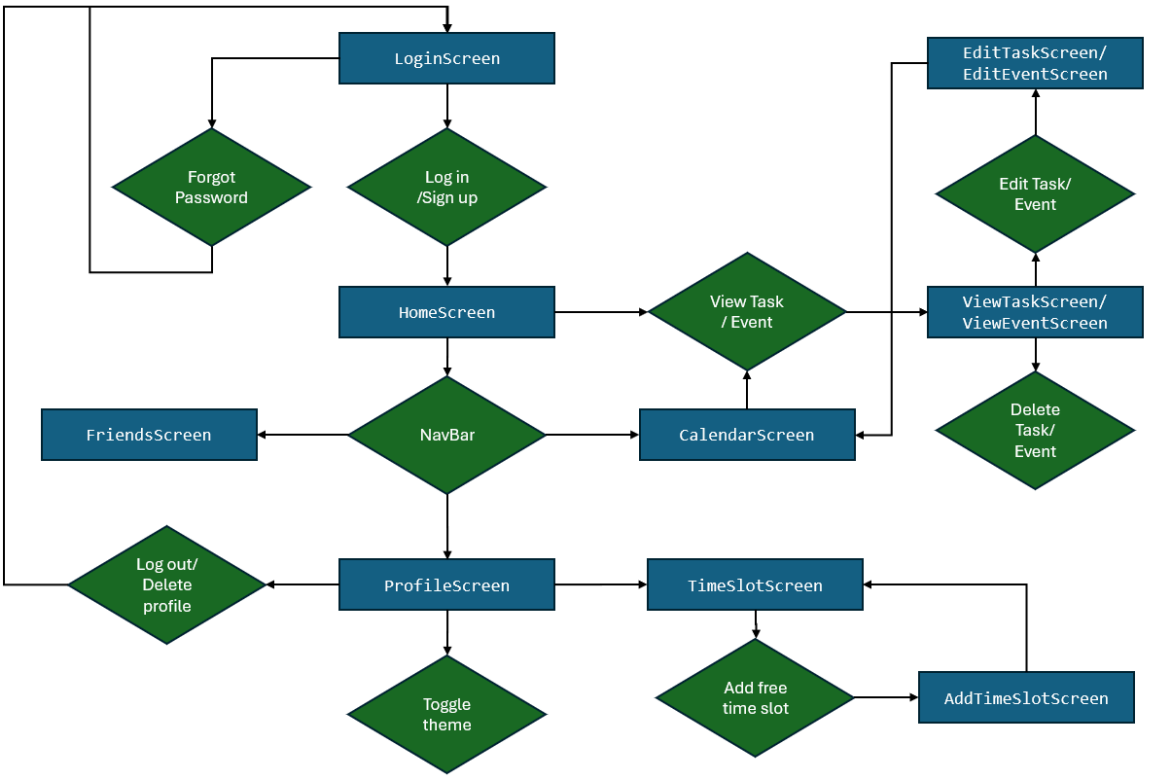




Navigation Diagram

The navigation diagram shows the user's navigation throughout the screens of the app, between the various screens and by the functions. The blue rectangles indicate the

Navigation Diagram



Technologies used

React Native

Justification:

Cross-Platform Development: React Native allows for simultaneous development of both iOS and Android applications from a single codebase, significantly reducing development time and effort compared to maintaining separate codebases for each platform.

Community and Ecosystem: React Native has a vast and active community, providing a wealth of libraries, tools, and shared knowledge that can accelerate development and problem-solving.

Performance: With its ability to use native components, React Native can achieve near-native performance, which is critical for ensuring a smooth user experience.

Developer Experience: Leveraging JavaScript and React, React Native offers a familiar environment for web developers transitioning to mobile development. Hot reloading and a strong set of debugging tools further enhance productivity.

Alternatives:

Flutter: While Flutter is a strong contender with its rich set of pre-designed widgets and performance benefits, its use of Dart (a language less commonly known compared to JavaScript) can pose a learning curve. Additionally, React Native's ecosystem and community support are more mature.

Swift (for iOS and Kotlin (for Android): These languages are optimal for native development but require maintaining separate codebases for iOS and Android, doubling the development and maintenance efforts. This approach is less efficient for a small team or a project aiming for rapid development and iteration.

Firebase

Justification:

Integrated Services: Firebase provides a comprehensive suite of backend services such as authentication, real-time database, cloud storage, and cloud functions, which simplifies backend development and maintenance.

Real-Time Capabilities: Firestore, part of Firebase, offers real-time synchronisation, making it ideal for applications requiring real-time updates, such as task management and scheduling apps.

Scalability: Firebase scales seamlessly with the growth of the user base, handling everything from a few users to millions without significant changes to the infrastructure.

Ease of Use: Firebase's SDKs and documentation are well-designed, making it easier to integrate and use, which accelerates development and reduces the need for extensive backend infrastructure management.

Alternatives:

Supabase: Although Supabase offers an open-source alternative to Firebase with similar real-time capabilities and PostgreSQL database, it is relatively newer and less mature compared to Firebase. Firebase's long- standing presence in the market ensures better stability, support, and a larger community.

MongoDB While MongoDB is a powerful NoSQL database, integrating it requires setting up and managing additional backend infrastructure (such as a server environment), which increases complexity and maintenance effort. Firebase's serverless architecture and integrated services offer a more streamlined and hassle-free experience.

Accessing the project

In order to view the code for the project, you may refer to the Github repository.

In order to access the app itself, refer to the Usage section below.

You can also view the project poster and the project demo video for Milestone 2.

Usage

Accessing the app

This app is available as an Android Application Package (APK). You can download it here. Enter the link and click “Download” or “install” to access the app on your phone.

If you are unable to access the APK file, you can also use Expo Go to access the app. Follow the steps below to access the app:

1. Install the Expo Go app from the App store or Play store.
2. Create an account and sign in.
3. Enter this link.
4. Click on “Open” next to “Expo Go”. this will redirect you to the app.

Note: Initially the app may take a while to load.

User Authentication

1. Click “Need an account? Sign Up”, enter your email, password and name, then click “Sign Up”.
2. If you already have an account, enter your email and password and click “Sign In”.
3. If you have forgotten your password, enter your email and click “FORGOT PASSWORD”. The link to reset your password should be sent to your email.
4. Navigate to Profile Screen by clicking the profile icon on Home Screen or the navigation bar in the other screens and click “Logout” to sign out of the app
5. Navigate to Profile Screen by clicking the profile icon on Home Screen or the navigation bar in the other screens and click “Delete profile” to delete your profile on the app.

Note: While signing in, if you see a screen with a red logout button, ignore it. It means the main pages of the app are still loading. This may take a while on your initial sign in.

Calendar View

1. On the Home Screen, click “My Tasks & Events” to access the Calendar.
2. If you are in another screen, click the calendar icon in the navigation bar to access the calendar.

Task Management & Auto Scheduling

1. Click on the “+” icon on the Home Screen/Calendar Sceen.

2. Select “Task” or “Event” based on your preference
3. Fill out the details and then click “Save Task”/”Save Event”
4. Once on the Home Screen /Calendar Screen, click on the task to view it in detail
5. Click “Back” to navigate to the Calendar Screen or click “Delete Task/Event” to remove the task/event

Asynchronous Storage

1. Close the app completely.
2. When you enter the app, it should sign you in automatically

Offline Functionality

1. Sign in to the app
2. Turn off your network connectivity.
3. Try viewing your tasks & events in the Home Screen/Calendar Screen
4. Try creating an deleting tasks by following the instructions under task management

Note: This function may not work perfectly on all phone models as it is dependent on storing and retrieving data in the phone’s cache.

Themes

1. Navigate to Profile Screen by clicking the profile icon on Home Screen or the navigation bar in the other screens.
2. Toggle the “Dark mode” toggle and it will prompt you to logout and sign in again
3. Logout and sign in again and see the theme update.

Free Time Slot management

1. Navigate to Profile Screen by clicking the profile icon on Home Screen or the navigation bar in the other screens.
2. Click on the “Free time slots” button and then “Add Timeslot”.
3. Fill out the details of your time slot and click “Save”.

User Authentication

The user authentication feature allows users to sign up, log in, log out, and reset their passwords. It integrates with Firebase Authentication and Firestore to manage user sessions and store user profile data.

Components

LoginScreen.js

Manages user authentication (sign in and sign up). Handles user session state changes.
Provides functionality for password reset.

ForgotScreen.js

Allows users to reset their password via email. The user authentication feature of the app is primarily managed through two main components: LoginScreen.js and ForgotScreen.js . The LoginScreen component handles both user sign-in and sign-up processes. It leverages React's useState hook to manage form data (such as email, password, and user name) and the useEffect hook to listen for authentication state changes using Firebase's onAuthStateChanged listener. When a user logs in or signs up, the

handleAuthentication function is invoked. This function determines the user's authentication state: if a user is logged in, it signs them out; if not, it either signs them in or creates a new account depending on the isLogin state. Upon successful sign-up, a user profile is created in Firestore with initial data such as name, email, and an empty analytics structure. The component also includes a password reset feature, which, when invoked, sends a password reset email to the user. The ForgotScreen component is dedicated to handling password resets, allowing users to enter their email and receive a reset link.

Key Functionalities

Sign In/Sign Up

Users can sign in if they already have an account or sign up to create a new account. On successful sign-up, a user profile is created in Firestore.On successful sign-in, the user is redirected to the Home screen.

Password Reset

Users can reset their password by providing their email address.A password reset email is sent to the provided email address.

User Session Management

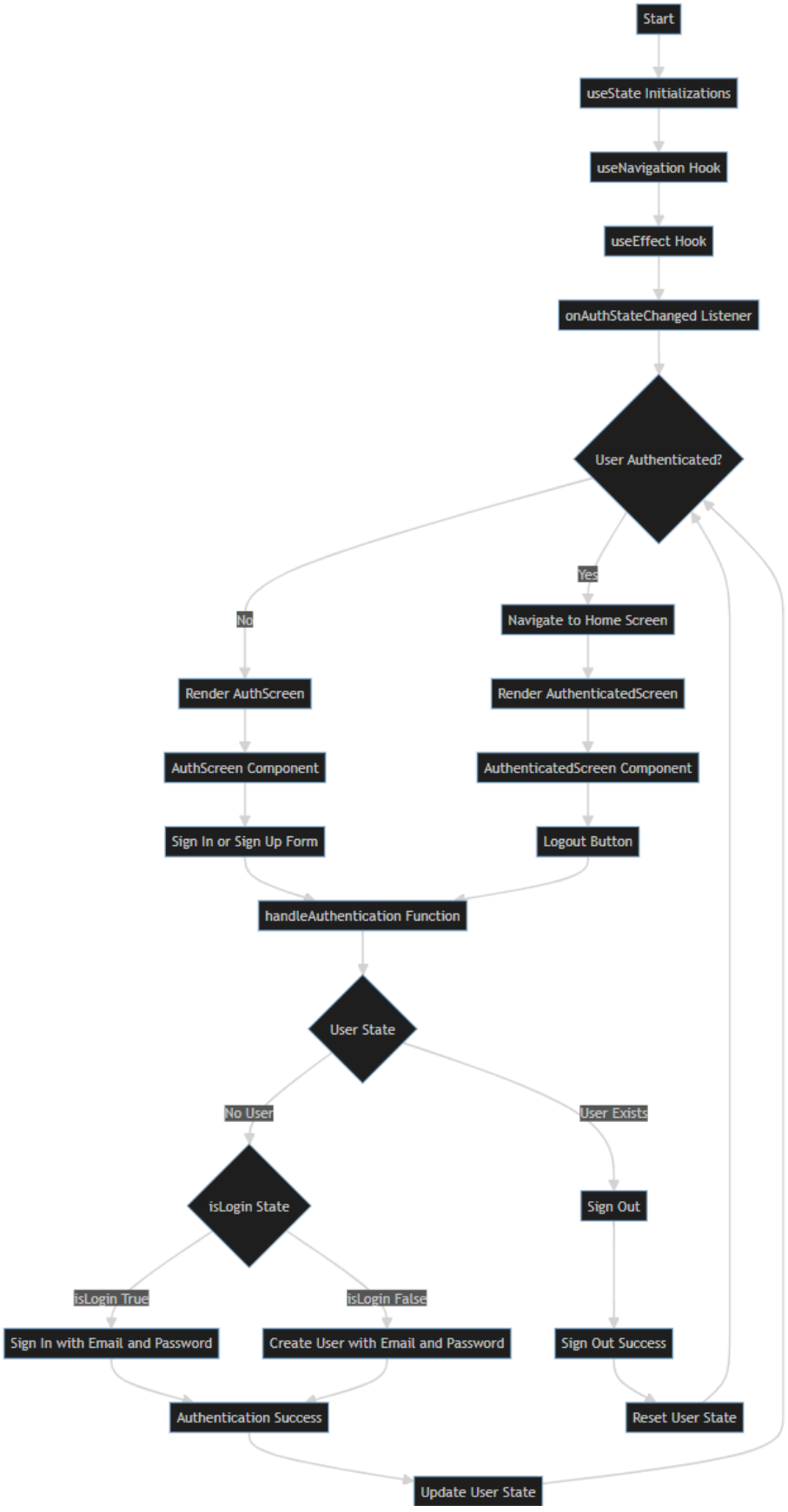
The app listens for authentication state changes and updates the UI accordingly. If a user is authenticated, they are redirected to the Home screen. If no user is authenticated, the login screen is displayed.

Authentication Flow

The authentication flow begins with the initialization of state variables and navigation hooks. When the LoginScreen component mounts, the useEffect hook sets up the onAuthStateChanged listener to monitor the authentication state. If a user is authenticated, the app fetches their profile data from Firestore and navigates to the Home screen, updating the UI to reflect the logged-in state. If no user is authenticated, the login form is displayed. Users can then choose to sign in or sign up using the form. The handleAuthentication function processes the form input, authenticating the user through Firebase. On successful authentication, the user's state is updated, and they are redirected to the Home screen. If a user chooses to reset their password, the handlePasswordReset function sends a reset email, guiding the user through the process of regaining access to their account. This comprehensive flow ensures seamless user authentication and management, providing a robust and user-friendly experience.

LoginScreen.js

Diagram of user authentication feature

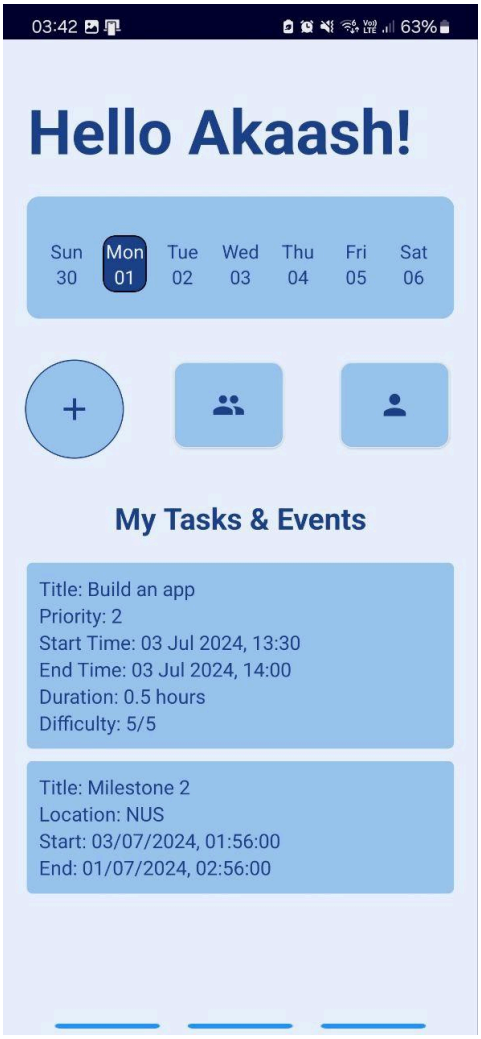


Calendar View

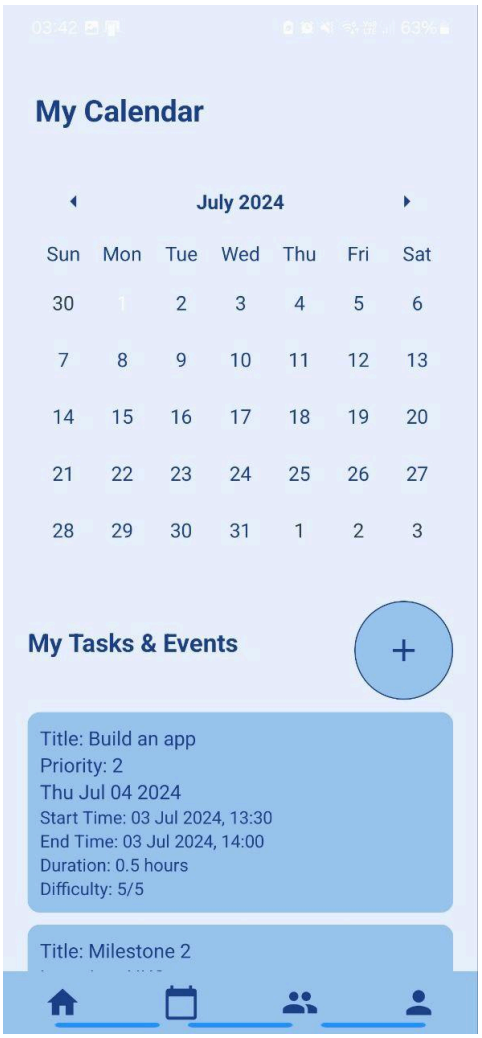
In HomeScreen.js, the flow begins with importing necessary dependencies and setting up state variables using the useState hook.The current date and day are obtained using JavaScript's Date object, formatted, and stored in variables. These values are then displayed on the screen using the Text component.

In CalendarScreen.js, the process starts with importing the necessary dependencies, including React Native components and the Calendar component from react-native-calendars. The Calendar component is then rendered, providing a calendar view where users can interact with it to view the dates and days of individual months.

Date in HomeScreen.js



Calendar in CalendarScreen.js



Task Management

Task management is done across three components: HomeScreen.js, TaskScreen.js and ViewScreen.js. The same is done for event management in HomeScreen.js, EventScreen.js and ViewEventScreen.js respectively. Both sets of screens have the exact same functionality to manage tasks and events. Hence, the explanations for task management can also be assumed to be the same for event management.

Breakdown of Screens

HomeScreen.js

- **Components:**
    - View, StyleSheet, ScrollView, TouchableOpacity
    - Text, Card, Icon
    - CircularDropdown
- **Firebase:**
    - auth, firestore
    - onAuthStateChanged
    - collection, doc, onSnapshot, getDoc
- **Hooks:**
    - useState, useEffect, useCallback, useContext
    - useFocusEffect
- **Context:**
    - ThemeContext
    - darkTheme, lightTheme

TaskScreen.js

- **Components**:
    - View, StyleSheet, ScrollView, TouchableOpacity, TextInput, Alert
    - Text
    - DateTimePickerModal
- **Firebase**:
    - firestore
    - collection, addDoc
- **Other Libraries:**
    - AsyncStorage
    - NetInfo
    - CommonActions
- **Hooks:**
    - useState, useEffect, useContext
- **Context:**
    - ThemeContext

ViewScreen.js

- **Components**:

    - View, StyleSheet, Text, TouchableOpacity, ActivityIndicator, Alert
    - Card
- **Firebase:**
    - firestore
    - doc, getDoc, deleteDoc
- **Other Libraries:**
    - AsyncStorage
    - NetInfo
- **Hooks:**
    - **useState, useEffect, useContext**
- **Context:**
    - ThemeContext

Functionalities of the Code

HomeScreen.js

- **User Authentication**: Listen for changes in authentication state.
- **Fetch Data**: Retrieve user name, tasks, and events from Firestore.
- **Render Tasks and Events**: Display tasks and events with navigation to detailed view screens.
- **Navigation**: Navigate to task/event creation screens and other sections (Friends, Profile, Calendar).

TaskScreen.js

- **Task Creation**: Create new tasks and save them to Firestore.
- **Draft Management**: Save task drafts locally using AsyncStorage and sync them when online.
- **Priority and Due Date**: Set task priority and due date using DateTimePicker.
- **Offline Support**: Handle task creation when offline by saving drafts.
- **Task Scheduling:** After saving a task, call the fetchAndScheduleTasks algorithm to schedule tasks based on available free time slots and existing events.

ViewScreen.js

- **Fetch Task Data**: Retrieve task details from Firestore and local storage.
- **Display Task Details**: Show task title, description, due date, priority, and repeat settings.
- **Delete Task**: Remove task from Firestore and local storage.
- **Navigation**: Navigate to task editing screen or back to calendar view.

Task Management Flow

In **HomeScreen.js**, when the component mounts, it listens for authentication state changes. If a user is authenticated, it fetches the user's name and their tasks and events from

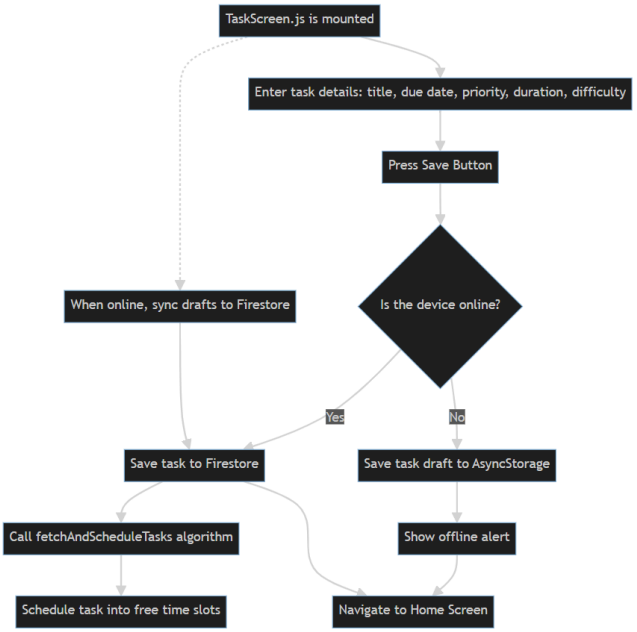
Firestore. These tasks and events are then rendered in a scrollable view. The user can navigate to different sections or create new tasks/events using the provided buttons and dropdowns. The task and event data is kept up-to-date with real-time listeners.

In **TaskScreen.js**, the user can create a new task by entering a title, setting a due date, choosing a priority, specifying a duration, and setting a difficulty level. The app supports offline functionality by saving drafts to AsyncStorage if there is no internet connection. When the save button is pressed, the task data is either saved to Firestore or stored locally if offline. Previously saved drafts are also loaded and synced to Firestore when the user is online. Additionally, after saving a task, the fetchAndScheduleTasks algorithm is called to automatically schedule tasks into available free time slots.

In **ViewTaskScreen.js**, the task details are fetched from Firestore or local storage and displayed to the user. The user can delete the task, which will remove it from both Firestore and local storage. The screen also provides options to navigate to the task editing screen or back to the calendar view.

Diagram of HomeScreen.js and HomeScreen.js

Diagram of TaskScreen.js and TaskScreen.js



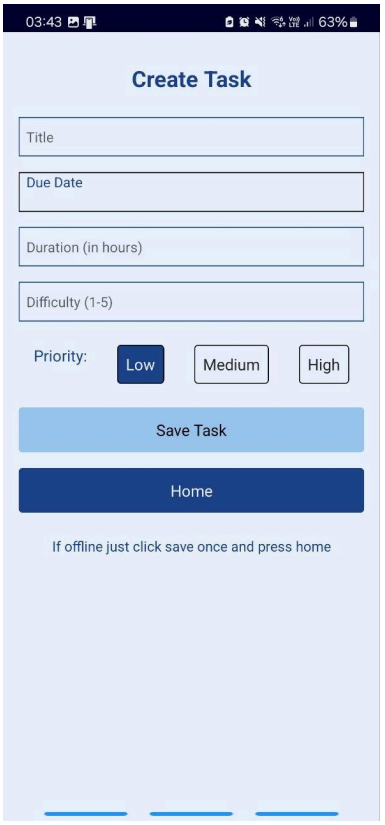
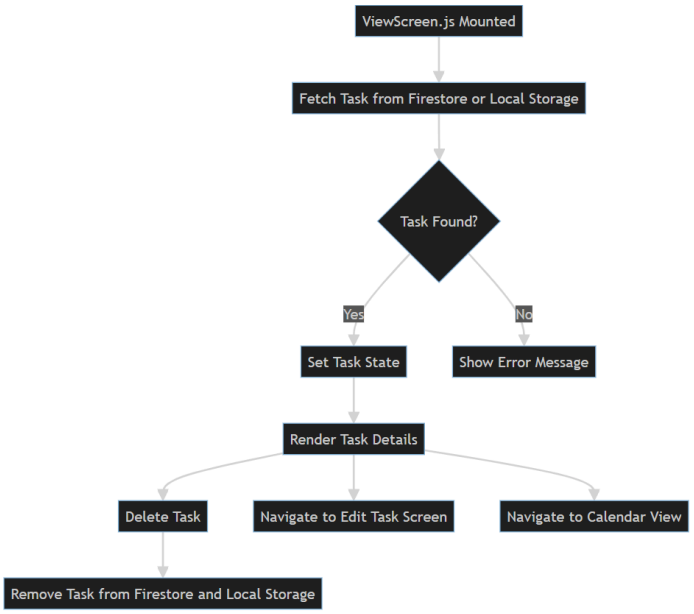
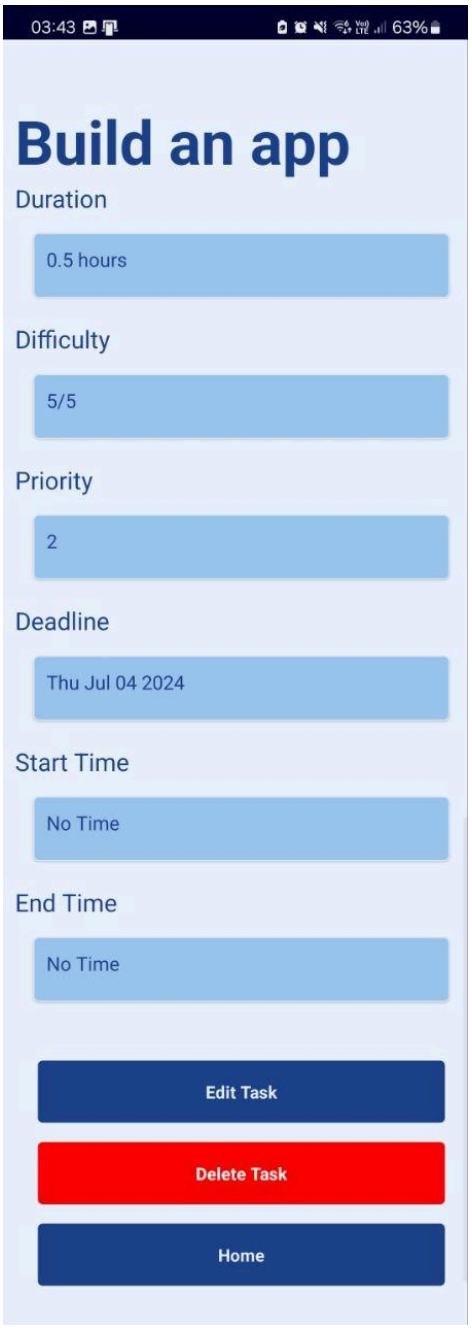


Diagram of ViewTaskScreen.js and ViewEventScreen.js





Time Slot Management

The Time Slot Management feature allows users to view, add, and delete their available time slots. It integrates with Firebase Firestore to manage and store the time slots data.

Components

time slotScreen.js

- Renders the list of time slots.
- Provides functionality to delete a time slot.

Addtime slotScreen.js

- Allows users to add a new time slot.
- Checks for conflicts with existing time slots.

time slotScreen.js

The time slotScreen component is responsible for displaying the user's existing time slots and providing the functionality to delete them. It leverages React Native's hooks to manage state and Firebase Firestore for data management.

Key Functionalities:

1. Fetch User ID:
    - Uses Firebase Authentication to fetch the current user's ID.
    - Stores the user ID in state.
2. Fetch Time Slots:
    - Fetches the time slots from Firestore for the logged-in user.
    - Stores the fetched time slots in state.
    - Displays a loading indicator while fetching data.
3. Delete Time Slot:
    - Deletes a time slot from Firestore.
    - Updates the state to remove the deleted time slot from the displayed list.

Time Slot Format:

- Time slots are displayed in a readable format (e.g., "Monday: 9:00 AM - 10:00 AM").

Addtime slotScreen.js

The Addtime slotScreen component allows users to add new time slots. It provides input fields to select the day of the week, start time, and end time, and checks for conflicts with existing time slots before adding the new time slot to Firestore.

Key Functionalities:

1. Fetch User ID:

- Fetches the current user's ID if not already provided.
- Displays an error if the user is not found in Firestore or not signed in.

2. Time Selection:
   - Allows users to select the day of the week.
   - Provides date-time pickers for selecting start and end times.
   - Formats the selected times for display.
3. Add Time Slot:
   - Validates the selected time slot to ensure the start time is before the end time.
   - Checks for conflicts with existing time slots.
   - Adds the new time slot to Firestore if no conflicts are found.

TimeSlotScreen.js and AddTimeSlotScreen.js respectively

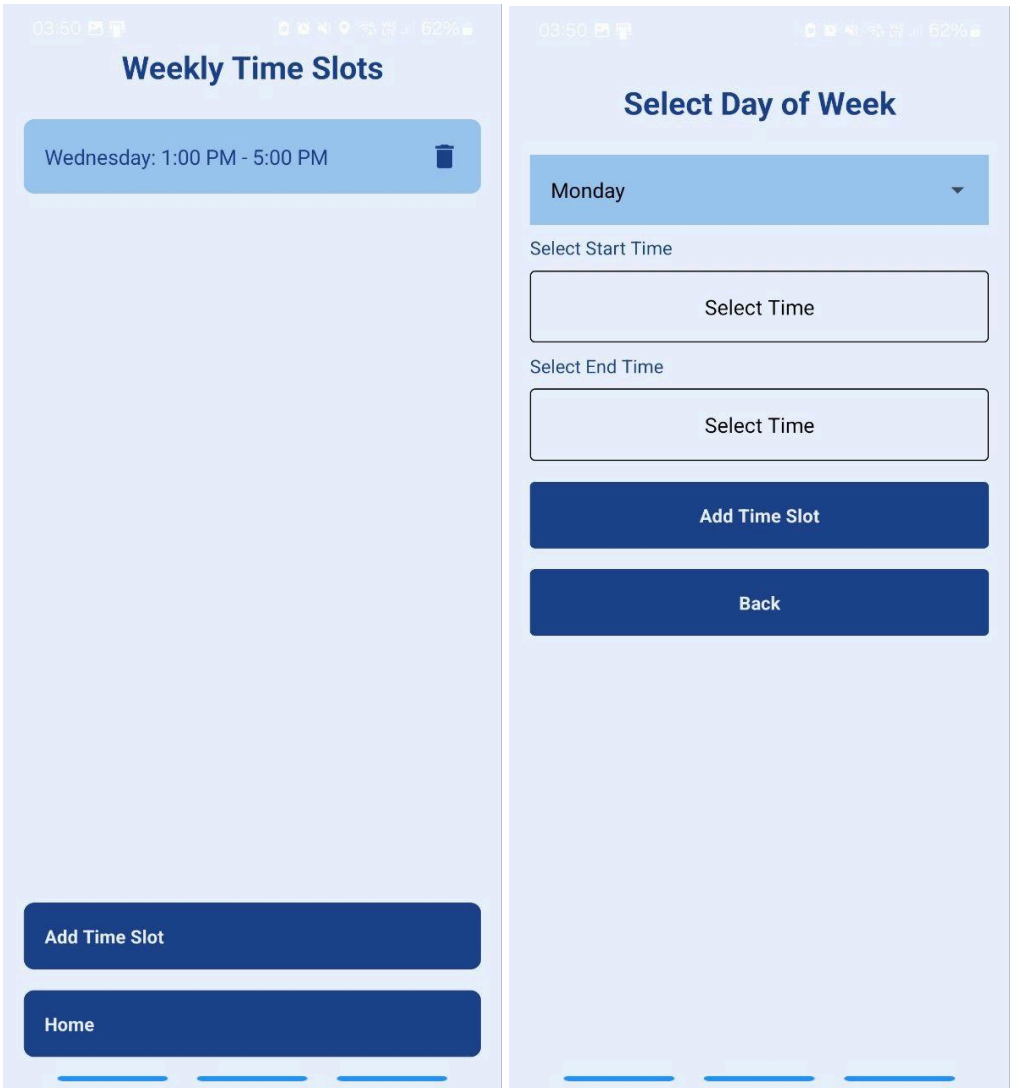
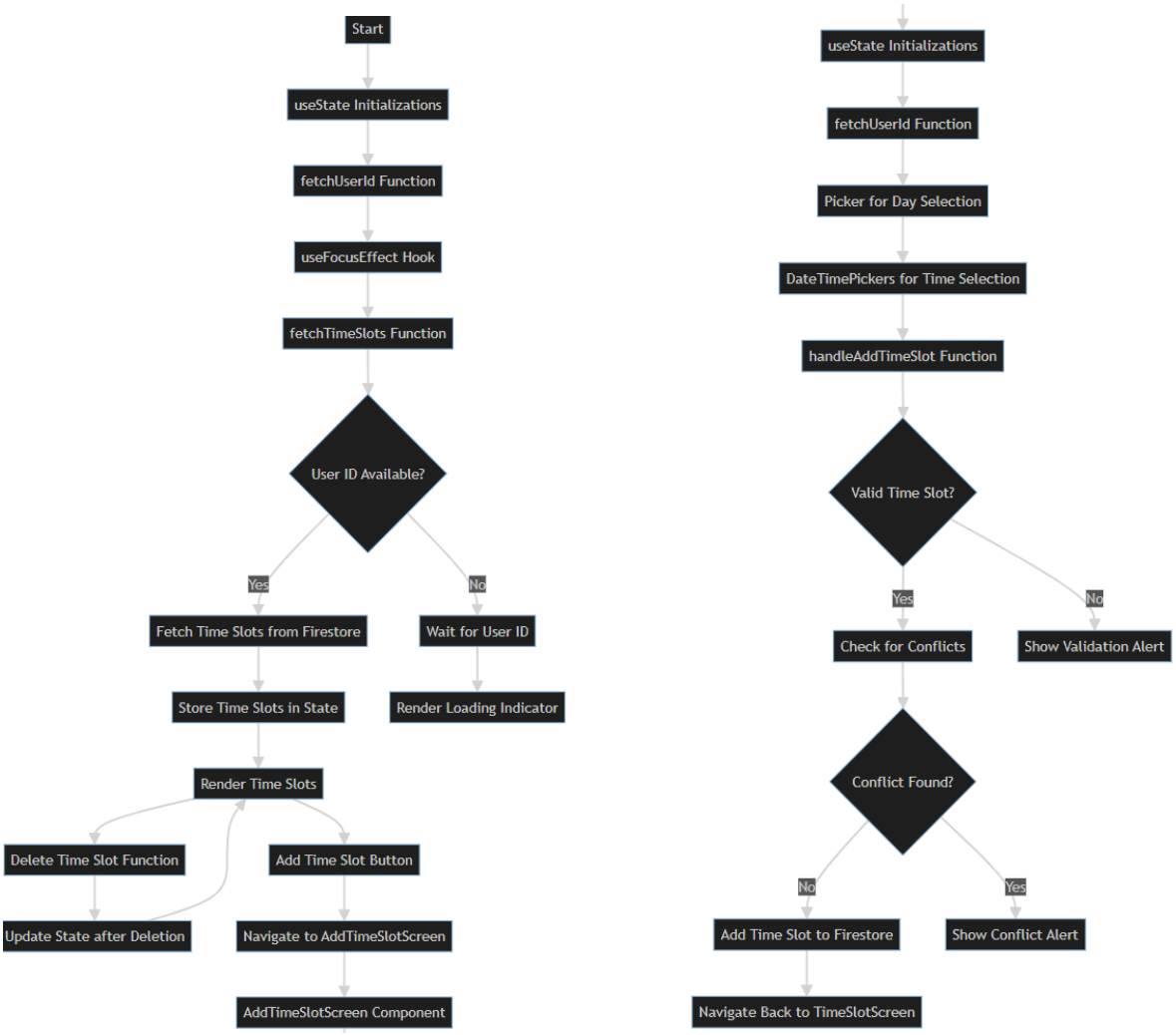


Diagram for time slot functionality (left is first half and right is second half)



Auto scheduling

Components:

- **Firebase:**
    - collection, getDocs, updateDoc, doc
- **Custom Functions:**
    - fetchAndScheduleTasks
    - Helper functions: isValidWeeklySlot, isClashingWithEvents, getSlotTimeInMinutes, getSlotDate, addMinutes

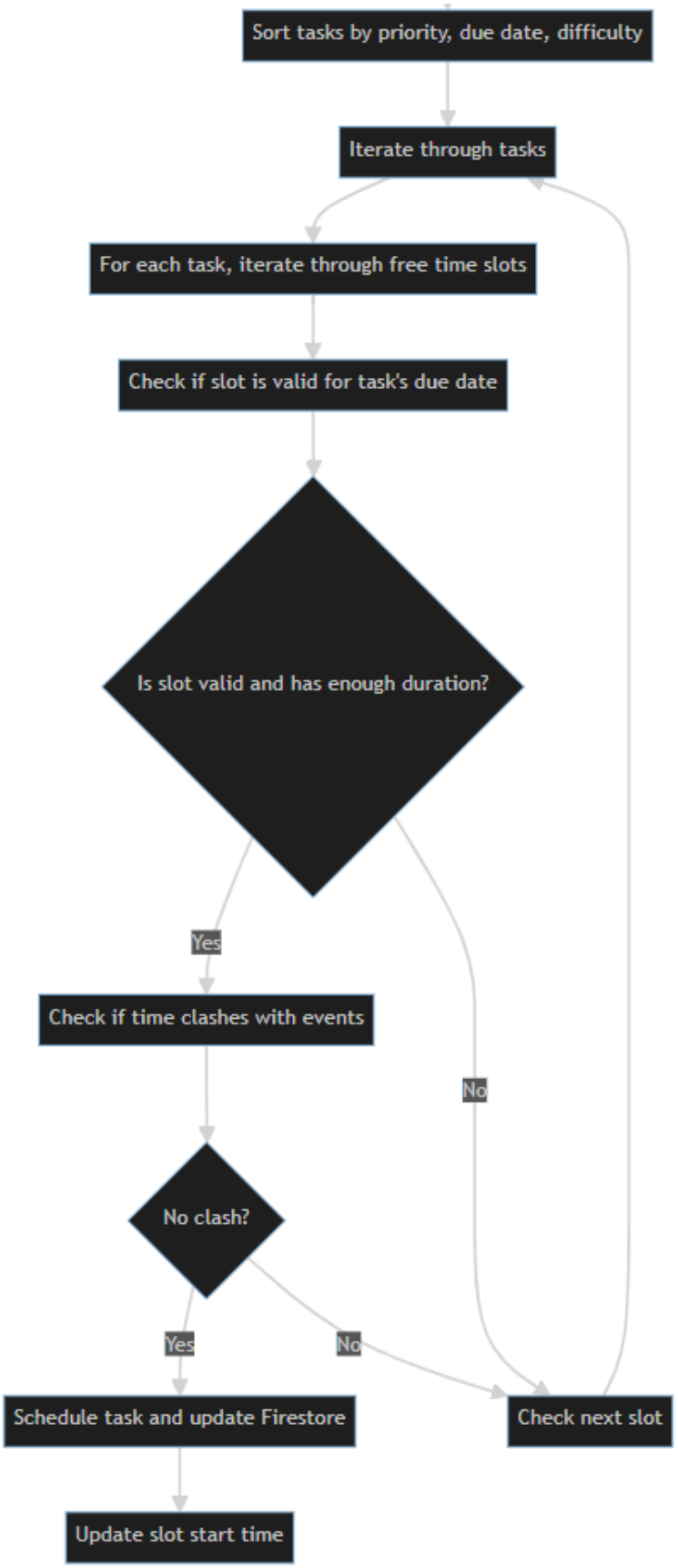
Functionality:

1. **Fetch Data:**
    - Tasks: Retrieve tasks from Firestore.
    - Free Time Slots: Retrieve free time slots from Firestore.
    - Events: Retrieve events from Firestore.
2. **Sort Tasks:**
    - Sort tasks by priority, due date, and difficulty.
3. **Schedule Tasks:**
    - For each task, iterate through available free time slots.
    - Check if the slot is valid for the task's due date.
    - Check if the slot has enough duration for the task.
    - Ensure the scheduled time does not clash with existing events.
    - Update task's start and end time in Firestore if a valid slot is found.

Code Flow

The `fetchAndScheduleTasks` function automates the scheduling of tasks by first fetching tasks, free time slots, and events from Firestore. The tasks are then sorted by priority, due date, and difficulty. For each task, the function iterates through the free time slots to find a valid slot that matches the task's due date and has sufficient duration. It also ensures that the scheduled time does not clash with any existing events. If a suitable slot is found, the task's start and end times are updated in Firestore, and the slot's start time is adjusted accordingly. This process continues until all tasks are scheduled.

Diagram of the auto scheduling algorithm



Navigation Bar

Components:

1. **View**: Renders a container view for the navigation bar.
2. **Icon**: Renders icons for navigation actions using icons from the react-native-elements library.

Functions:

1. **useContext**: Hooks into the ThemeContext to retrieve the current theme for styling icons.
2. **onPress handlers**: Navigate to various pages when pressed

These functions handle the navigation within the app based on user interactions with the icons, providing seamless access to different sections of the app.

The NavBar component provides navigation functionality within the app using icons for Home, Calendar, Friends, and Profile. Each icon is configured with an onPress handler that navigates to its respective screen when tapped. The icons are styled based on the current theme retrieved from the ThemeContext, ensuring consistent appearance across different screens. The component utilises React Native's Icon component from react-native-elements for rendering icons with specified names, types, sizes, and colours. This bottom navigation bar design enhances user interaction by offering quick access to essential app sections from any screen.

Image of the navigation bar in ight theme.

Asynchronous Storage

Asynchronous storage in the application handles real-time data operations using Firebase Firestore. This ensures efficient data handling and a smooth user experience by performing actions in the background.

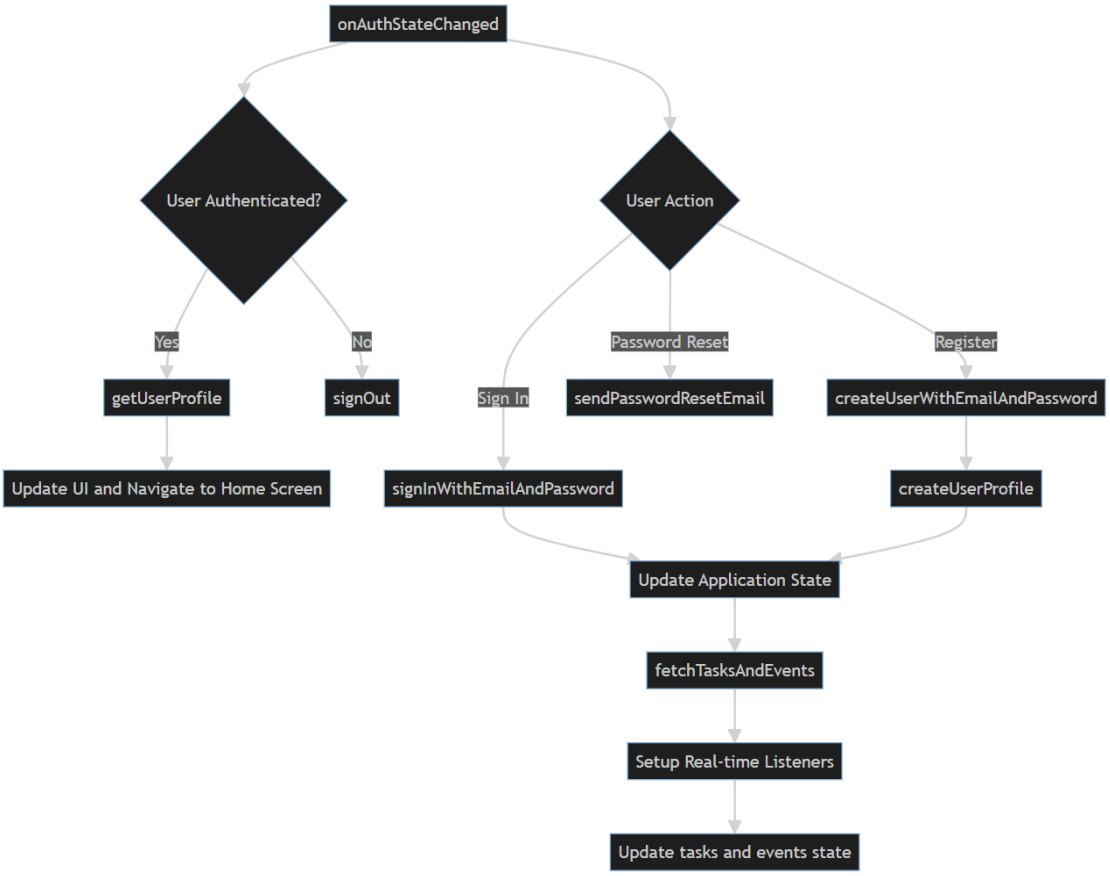
Functions used

- onAuthStateChanged
- getUserProfile (utilises getDoc)
- createUserProfile (utilises setDoc)
- signInWithEmailAndPassword
- createUserWithEmailAndPassword
- signOut
- sendPasswordResetEmail

Flow of Asynchronous Storage

1. **Authentication State Monitoring:**
    - **onAuthStateChanged**: Monitors authentication state changes and triggers actions when the user's authentication status changes.
2. **User Profile Management:**
    - **getUserProfile**: Retrieves the user's profile data from Firestore using the getDoc method.
    - **createUserProfile**: Stores the user's profile in Firestore using setDoc.
3. **User Authentication:**
    - **signInWithEmailAndPassword**: Authenticates the user when logging in.
    - **createUserWithEmailAndPassword**: Creates a new user account when registering.
    - **signOut**: Logs out the user.
    - **sendPasswordResetEmail**: Sends a password reset email.
4. **Home Screen Data Handling:**
    - **onAuthStateChanged**: Updates the userId state when a user signs in.
    - **getUserProfile**: Fetches the user's name to update the userName state.
    - **fetchTasksAndEvents**: Sets up real-time listeners using onSnapshot on the user's tasks and events collections in Firestore.
5. **Calendar Screen Data Handling:**
    - **useFocusEffect**: Triggers data fetching when the screen comes into focus.
    - **fetchTasksAndEvents**: Sets up real-time listeners using onSnapshot for tasks and events collections.

Diagram of Asynchronous Storage



Offline Accessibility

Offline accessibility ensures that the application can still function and provide a seamless user experience even when the user is not connected to the internet. This is achieved by leveraging Firestore's offline persistence capabilities.

Components Used

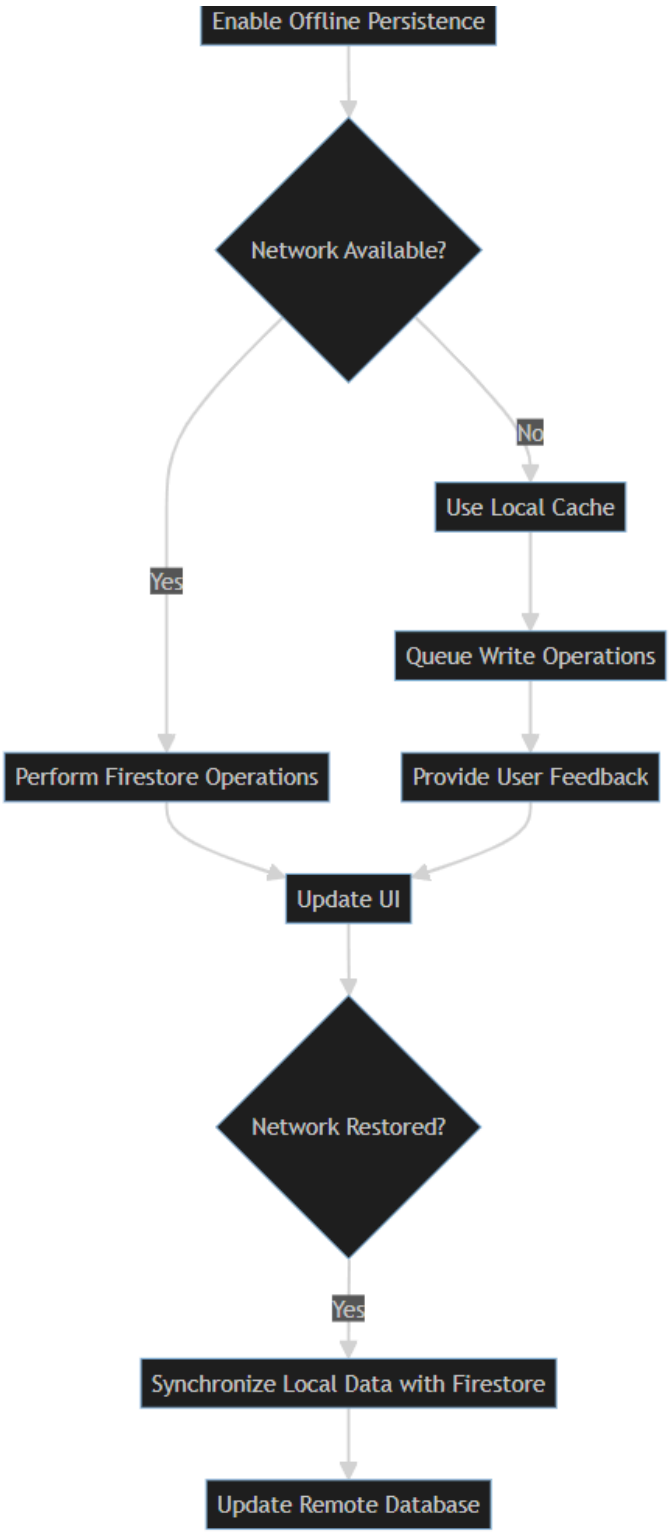
- **Firestore's Offline Persistence**

Flow of Offline Accessibility

1. **Enable Offline Persistence:**
    - Configure Firestore to enable offline data persistence.
2. **Local Cache Usage:**
    - When offline, Firestore uses the local cache to serve data and queue write operations.
3. **Synchronisation:**
    - Once the network is restored, Firestore synchronises local data with the remote database, ensuring all changes are updated.
4. **Error Handling:**
    - Handle errors gracefully by providing feedback to the user when offline.

The asynchronous storage and offline accessibility components work together to provide a seamless user experience. The asynchronous methods ensure efficient data handling and real-time updates, while offline accessibility ensures the application remains functional even without an internet connection.

Diagram of Offline Accessibility



Themes

This documentation provides a comprehensive overview of how dark and light themes are implemented in a React Native application. The application utilises Firebase for user authentication and Firestore for storing user preferences. A theme context is provided, allowing users to toggle between dark and light modes. The implementation ensures that the user experience is consistent and seamless across different themes.

Components and Flow

1. App.js

The App.js file serves as the main entry point for the application. It renders the AppNavigator component, which is responsible for managing the navigation and theme context across the application. This setup ensures that the entire application is aware of the current theme and can update the UI accordingly.

2. AppNavigator.js

AppNavigator.js is a critical component that handles both navigation and theme management. It uses React's useState and useEffect hooks to manage the user state and the selected theme. When the application starts, it subscribes to Firebase authentication state changes using the onAuthStateChanged function. When a user logs in, it fetches the user's theme preference from Firestore and updates the theme state accordingly. If the user's document does not exist in Firestore, it creates a new document with a default light theme. The component also provides a method to save the user's theme preference back to Firestore. The NavigationContainer from React Navigation applies the selected theme and manages the navigation stack, ensuring that all screens are displayed with the correct theme settings.

3. Index.js (Themes)

The Index.js file under the /src/themes directory defines and exports two theme objects: darkTheme and lightTheme. Each theme object contains properties such as colours, which define colour values for various UI elements like background, text, card, and notifications. These theme objects are used throughout the application to ensure a consistent look and feel based on the user's preference.

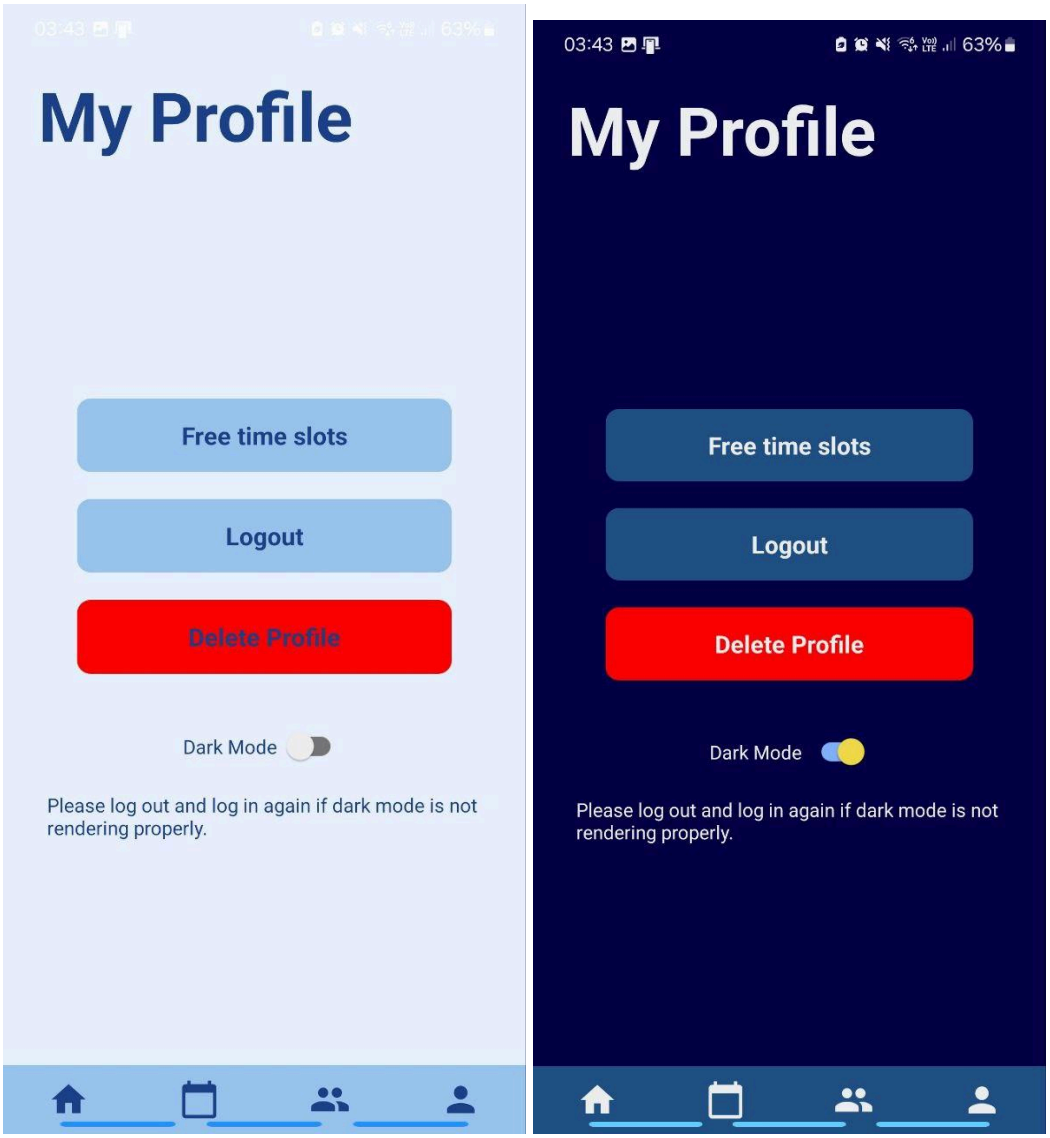
4. ThemeContext.js

ThemeContext.js manages the theme context and user preferences. It creates a ThemeContext using React's createContext and implements a ThemeProvider component to manage the theme state. This component listens for Firebase authentication state changes and fetches the theme preferences from Firestore. When the theme preference is updated, it updates the context and ensures the new theme is applied. The ThemeProvider also provides a method to save the theme preference to Firestore. This ensures that the theme is applied only after the user and theme data are fully loaded, providing a seamless experience for the user.

Rendering Screens and Toggling Themes

The ProfileScreen component demonstrates how screens are rendered and how themes can be toggled. It uses the ThemeContext to access the current theme and provides a toggle switch for users to change between dark and light modes. When the theme is toggled, the new theme is set, and the preference is saved to Firestore. Additionally, users are prompted to log out and log in again to apply the changes fully.

Light and Dark modes in ProfileScreen.js



User Testing

User Testing					Results	
Test ID	**User Story**	**Testing Objective**	**Steps Taken**	**Expected Results**	**Pass/Fail**	**Date Tested**
1	As a new user, I want to create a new account.	To ensure that the user can successfully sign up and create an account	1) Open the app, wait for login screen to load 2) Key in details on the “SIGN UP” component 3) Click “Sign Up”	1) Database should have user’s email registered	Pass	01/06/2024
2	As a returning user, I want to log in so that I can access my account	To ensure that users can log in without having to create an account again	1) Open the app, wait for login screen to load 2) Click “Already have an account? Sign up” 4) Key in username and password	1) User should be able to toggle across the home screens & see the creating, deleting tasks and profile page	Pass	01/06/2024
3	As a user, I want to reset my password if I forget it	To ensure users can reset password,with the same username and login to see the same things	1) Open the app. 2) Navigate to the login screen 3) Click "Forgot Password" and enter email 4) Check the registered email 5) Type in the new password	1) Password reset email successfully sent to the registered email 2)	Pass	01/06/2024

			on TaskMaster's login page and click "SIGN IN"			
4	As a user, I want to see the current date and day	To ensure that the current date and day are displayed, ensuring accuracy	1) Open the app. 2) Navigate to the login screen 3) Key in username & password and login & click "SIGN IN"	1) Name, current date and day should be correctly indicated on the home screen of TaskMaster	Pass	01/06/2024
5	As a user, I want to create a task so that I can keep track of my activities.	To ensure that users that users can create tasks with the correct details indicated	1) Open the app. 2) Click on the "+" icon 3) Enter task details such as timing, start date, end date and click "Save"	1) The task should appear in the task list 2) The task details should be correctly saved and displayed.	Pass	02/06/2024
6	As a user, I want to delete a task.	To ensure that the user can delete a task successfully.	1) Open the app. 2) Select a task. 3) Click on the "Delete" button/icon. 4) Confirm the deletion if prompted.	1) The task should be removed from the task list. 2) The task should be deleted from the database.	Pass	02/06/2024
7	As a user, I want my tasks to be available even when I'm offline.	To ensure that tasks are stored and accessible offline.	1) Open the app. 2) Turn off the device's internet connection.	1) The task should be available and editable offline.	Pass	02/06/2024

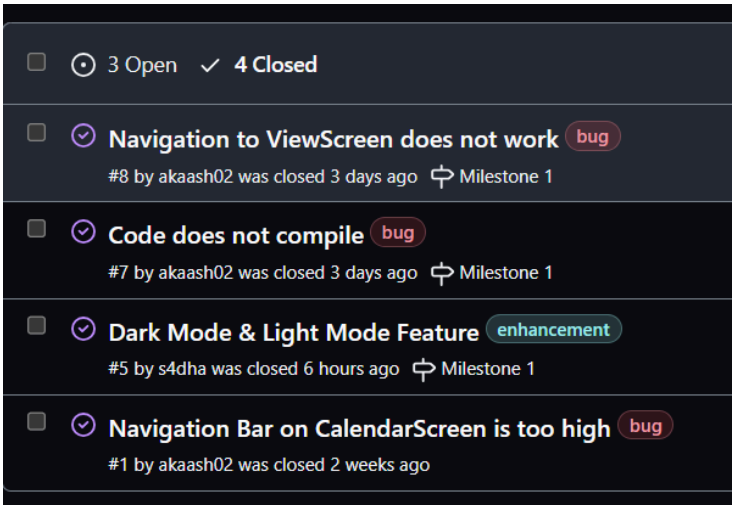
			3) Click on the “+” icon 4) Enter task details such as timing, start date, end date and click “Save”			
8	As a user, I want to switch between light and dark themes.	To ensure that the user can toggle between different themes.	1) Open the app. 2) Navigate to the Profile Screen by pressing the profile icon. 3) Toggle the theme 4) Log in again.	1) New theme should be visible when logging in again.	Pass	02/06/2024
9	As a user, I want to add free time slots to my schedule.	To ensure users can add free time slots with correct details.	1) Open the app. 2) Navigate to the free time slots section. 3) Click on the “Add time slot” button. 4) Enter details such as day of the week, start time, end time, and whether it’s a custom time slot. 5) Click “Save”.	1) The free time slot should appear in the free time slots list. 2) The free time slot details should be correctly saved and displayed.	Pass	23/06/2024
10	As a user, I want to delete a free time slot.	To ensure users can delete free time slots successfully.	1) Open the app. 2) Navigate to the free time slots section.	1) The free time slot should be removed from the free time slots list. 2) The free time slot should be	Pass	23/06/2024

			3) Select a free time slot. 4)Click on the “Delete” button/icon. 5) Confirm the deletion if prompted.	deleted from the database.		
12	As a user, I want the app to schedule my tasks automatically based on my free time slots and task priorities.	To ensure the auto scheduling algorithm works correctly and schedules tasks as expected.	1) Open the app. 2) Ensure there are free time slots and tasks available. 3) Trigger the auto scheduling feature. 4) Check the scheduled tasks for correct start and end times.	1) Tasks should be scheduled in available free time slots. 2) Tasks with higher priority should be scheduled first. 3) Tasks should not clash with existing events.	Pass	27/06/2024
13	As a user, I want the auto scheduling feature to avoid scheduling tasks during my existing events.	To ensure the auto scheduling algorithm does not schedule tasks during events.	1) Open the app. 2) Ensure there are events, free time slots, and tasks available. 3) Trigger the auto scheduling feature. 4) Check the scheduled tasks for correct start and end times, ensuring no	1) Tasks should be scheduled in free time slots that do not overlap with events. 2) Tasks should be rescheduled if they clash with any existing event.	Pass	27/06/2024

			overlap with events.			

Version Control

We utilise Git for version control and to streamline code management. The remote main branch consistently hosts a functional codebase. When tackling new features, we pull from the feature branch and commit to it. The nativebase branch is meant for creating design layouts with NativeBase. This practice of working on distinct branches safeguards against code mixing, as both branches have a large difference in the dependencies utilised. Once a feature is ready, we commit the changes in the feature branch. The feature branch will be integrated into the main branch after all features are added. The native base branch will then be integrated into the main branch in the end when the design layout and the required dependencies are decided.



Furthermore, Git Issues served as a tool for monitoring any ongoing issues or bugs within the application. Tags were added to differentiate the issues and the issues were closed upon completion.

Development Plan

The development plan for the app is based on two week sprints. The completed and planned sprint logs for each milestone can be found below.

Completed sprint logs

Below are the project logs for the completed sprint. Each time period is 3/4 days with the tasks completed during that time period.

Milestone 1 Sprint Goal

Establish the foundational features of the app and ensure basic user functionality, including user authentication, calendar view, asynchronous storage, task management, offline functionality, and colour themes.

Time Period	Tasks Completed
16 - 18 May	**Project Setup:** ● Initialize Expo project. ● Set up a project repository on GitHub. ● Install necessary dependencies (React Native, Firebase, Firestore, Expo, etc.). **Basic Interface Setup:** ● Design basic UI components (navigation, header, footer). ● Create placeholders for main screens (Login, Calendar, Tasks, Settings). **Firebase Integration:** ● Set up a Firebase project. ● Integrate Firebase and Firestore with the Expo project. ● Set up Firebase Authentication.
19 - 22 May	**User Authentication (Part 1):** ● Implement sign-up functionality. ● Implement login functionality. ● Design and develop UI for sign-up and login screens.

	• Test sign-up and login functionalities. **Calendar View:** • Design basic UI for the calendar view. • Implement a calendar component to display the current date and day. • Integrate calendar view with the navigation. **Review and Testing:** • Review code and ensure consistency with coding standards. • Conduct initial testing of user authentication and calendar view. • Fix any bugs identified during testing.
23 - 25 May	**User Authentication (Part 2):** • Implement password reset functionality. • Finalise UI for authentication screens. • Test password reset functionality. • Ensure user authentication is completed by 24 May. **Asynchronous Storage:** • Set up asynchronous storage using AsyncStorage. • Implement logic for faster sign-in after the initial sign-in. • Test asynchronous storage functionality. **Task Management (Part 1):** • Design UI for task management (creation, deletion, editing). • Implement task creation functionality. • Test task creation functionality.

26 - 29 May	**Task Management (Part 2):** ● Implement task deletion and editing functionalities. ● Test task deletion and editing functionalities. ● Integrate task management with Firebase Firestore. **Offline Functionality:** ● Implement caching for tasks and events using AsyncStorage. ● Ensure tasks and events are stored locally when offline. ● Implement synchronisation logic to update Firestore when connection is restored. ● Test offline functionality thoroughly. **Colour Themes:** ● Design and implement dark and light themes. ● Add a toggle switch in settings to switch between themes. ● Ensure all screens and components are compatible with both themes. ● Test theme switching functionality. **Review and Testing:** ● Conduct final testing for all implemented features. ● Review code for any final adjustments. ● Prepare for the next sprint planning.

Milestone 2 Sprint Goal

Finalise the auto scheduling feature, integrate and test the custom algorithm, implement software engineering best practices, and establish a CI/CD pipeline for the app along with automated testing using Jest.

Time Period	**Tasks to be completed**
17 - 19 June	**Auto Scheduling Feature Planning:** ● Discussed and planned the auto scheduling feature to allow users to input their free time. ● Conducted initial tests with LLMs to assist in scheduling tasks. ● Explored various LLMs to find a suitable one for task scheduling.
20 - 24 June	**Auto Scheduling Implementation Attempts:** ● Attempted to integrate LLMs for task scheduling. ● Faced issues with obtaining an API key due to payment method failures. ● Reassessed the approach and considered alternative solutions.
25 - 26 June	**Custom Algorithm Development:** ● Decided to develop a custom greedy algorithm for task scheduling. ● Designed and implemented the greedy algorithm. ● Conducted thorough testing of the custom algorithm to ensure accuracy and efficiency.
27 - 28 June	**Implementation of SWE Practices:** ● Explored various software engineering practices to enhance the development process. ● Implemented three selected practices to improve code quality

	and maintainability. ● Reviewed and documented the implemented practices for future reference.
29 - 30 June	**Review and Testing:** ● Conducted comprehensive testing of the auto scheduling feature with the custom algorithm. ● Reviewed code and ensured adherence to coding standards and best practices. ● Finalised the setup of the CI/CD pipeline to automate future builds and deployments. ● Prepared for the next sprint planning**.**

Planned Sprint Logs

These are the planned sprint logs for the upcoming milestones.

Milestone 3 Sprint Goals

Implement synchronisation with NUSMods, reminders feature, friends feature, sleep schedule tracker, time management analytics, complete testing with Jest, set up CI/CD pipeline, and finalise documentation.

Time Period	**Tasks to be completed**
7 - 9 July	**User Collaboration (Part 1):** ● Design the UI for the friends list and collaboration features. ● Implement user search and add friend functionality. ● Integrate basic collaboration features with Firestore. **Sync with NUSMods (Part 1):** ● Research and understand NUSMods API. ● Design the UI for timetable synchronisation. ● Implement initial data fetching from NUSMods. **Sleep Schedule Tracker (Part 1):** ● Design the UI for sleep schedule tracking. ● Implement basic functionality for tracking sleep schedule and setting reminders.
10 - 13 July	**User Collaboration (Part 2):** ● Finalise and test friend request and approval functionality. ● Implement and test task sharing and collaboration features. **Sync with NUSMods (Part 2):** ● Finalise timetable synchronisation logic.

	• Test the synchronisation process with sample data. • Integrate synchronised timetable with the main app. **Sleep Schedule Tracker (Part 2):** • Implement advanced features for sleep tracking and analytics. • Integrate sleep schedule with task and event management. **Testing with Jest (Part 1):** • Write unit tests for user collaboration features. • Write unit tests for NUSMods synchronisation.
14 - 17 July	**Sleep Schedule Tracker (Part 3):** • Finalise and test sleep tracking functionality. • Integrate with analytics dashboard for sleep insights. **Testing with Jest (Part 2):** • Write unit tests for sleep schedule tracker. • Execute tests and fix any identified issues. **Integration Testing:** • Conduct integration testing for all new features. • Ensure seamless interaction between user collaboration, NUSMods sync, and sleep schedule tracker. **Time Management Analytics:** • Design and implement analytics dashboard for time management insights. • Integrate analytics with other app features.

18 - 21 July	**Final Testing and Review:** ● Perform comprehensive testing of all features. ● Conduct code reviews and finalise any adjustments. ● Ensure all features are ready for production deployment. **Sprint Review and Retrospective:** ● Present completed features to stakeholders. ● Gather feedback and document improvements. ● Plan for the project completion and deployment phase. **Documentation and User Guides:** ● Prepare final documentation for all implemented features. ● Create user guides and tutorials to assist users in navigating the app. **CI/CD Pipeline:** ● Complete setup of CI/CD pipeline with GitHub Actions. ● Automate testing and deployment processes.

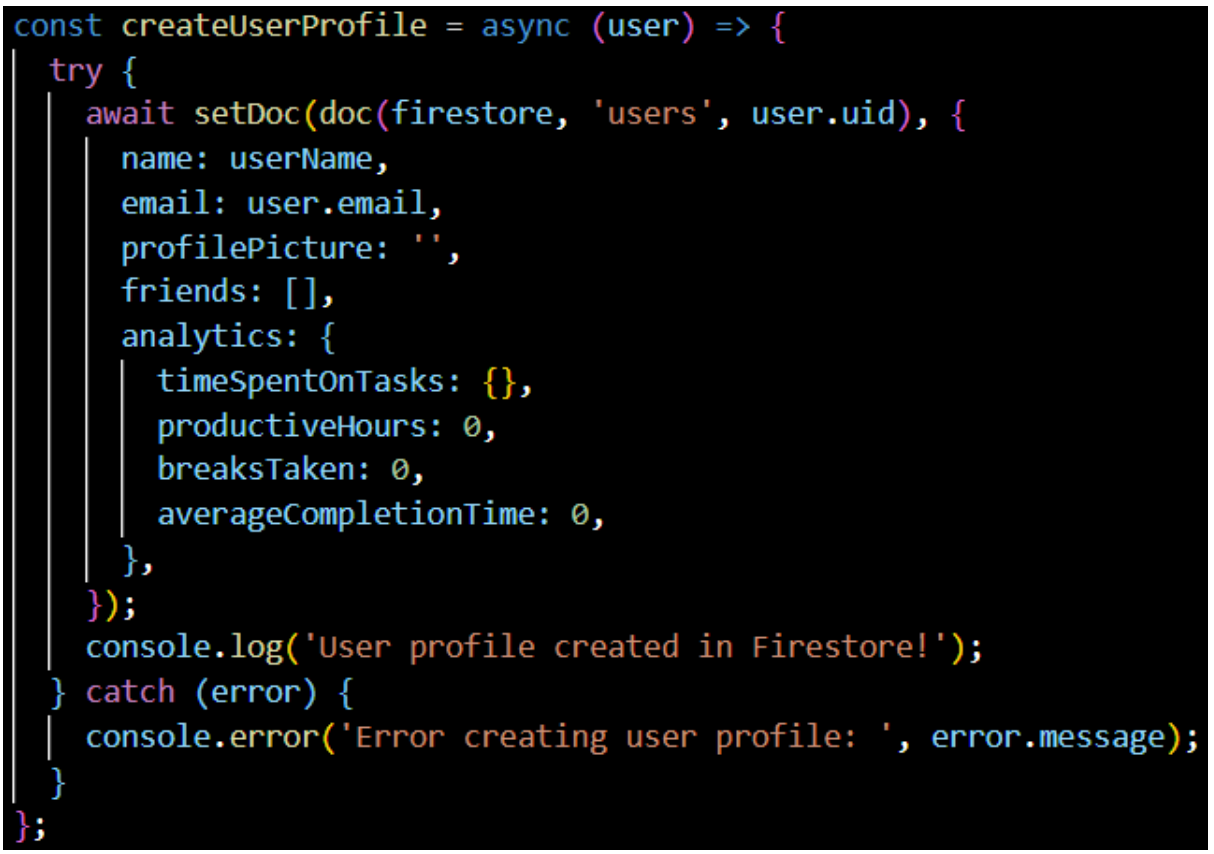
Software Engineering Principles

As we developed the app, we have also learnt to implement various software engineering principles in order to ensure the efficiency of our code and the usability of the app itself.

1. **Modularity**

   This principle refers to the idea that different parts of a program should handle different concerns or aspects of functionality, ensuring that each part is responsible for a single piece of logic. Here are specific snippets from your code that illustrate this principle:

   Code snippet showing this function is responsible solely for creating a user profile in Firestore. It does not handle any other aspect of the application.



```
const createUserProfile = async (user) => {
  try {
    await setDoc(doc(firestore, 'users', user.uid), {
      name: userName,
      email: user.email,
      profilePicture: '',
      friends: [],
      analytics: {
        timeSpentOnTasks: {},
        productiveHours: 0,
        breaksTaken: 0,
        averageCompletionTime: 0,
      },
    });
    console.log('User profile created in Firestore!');
  } catch (error) {
    console.error('Error creating user profile: ', error.message);
  }
};
```

Code snippet showing this function is responsible for fetching the user profile from Firestore based on the user ID. It does not handle user authentication or UI logic.

```
const getUserProfile = async (userId) => {
  try {
    const userDoc = await getDoc(doc(firestore, 'users', userId));
    if (userDoc.exists()) {
      console.log('User profile data: ', userDoc.data());
      return userDoc.data();
    } else {
      console.log('No user profile found!');
      return {};
    }
  } catch (error) {
    console.error('Error getting user profile: ', error.message);
    return {};
  }
};
```

Code snippet showing this function handles the authentication logic, including signing in, signing up, and logging out. It delegates profile creation to createUserProfile.

```
const handleAuthentication = async () => {
  try {
    if (user) {
      console.log('User logged out successfully!');
      await signOut(auth);
    } else {
      if (isLogin) {
        await signInWithEmailAndPassword(auth, email, password);
        console.log('User signed in successfully!');
      } else {
        const userCredential = await createUserWithEmailAndPassword(auth, email, password);
        await createUserProfile(userCredential.user);
        console.log('User created successfully!');
      }
    }
  } catch (error) {
    console.error('Authentication error:', error.message);
    switch (error.code) {
      case 'auth/wrong-password':
        alert('Incorrect password. Please try again.');
        break;
      case 'auth/user-not-found':
        alert('This email is not registered. Please create an account.');
        break;
      default:
        alert('Authentication failed. Please try again.');
    }
  }
};
```

2. Persistence

It includes mechanisms for both local and remote data storage, ensuring that tasks can be saved and synchronised even when the user is offline. Local storage of data utilises AsyncStorage to synchronise tasks.

Code snippet showing useEffect hook at the start loads any previously saved draft task from AsyncStorage when the component mounts

```
useEffect(() => {
  const syncTasks = async () => {
    const netInfo = await NetInfo.fetch();
    if (netInfo.isConnected) {
      const drafts = await AsyncStorage.getAllKeys();
      const taskKeys = drafts.filter(key => key.startsWith(`draftTask-${userId}-${scheduleId}`));
      for (const key of taskKeys) {
        const draftTask = JSON.parse(await AsyncStorage.getItem(key));
        const tasksCollectionRef = collection(firestore, 'users', userId, 'schedules', scheduleId, 'tasks');
        await addDoc(tasksCollectionRef, draftTask);
        await AsyncStorage.removeItem(key);
      }
    }
  };
  syncTasks();
}, [userId, scheduleId]);
```

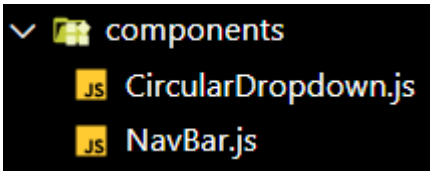
Code snippet showing the saveDraftTask function saves the current task data to AsyncStorage. This is called when the user saves the task while offline.

```
const saveDraftTask = async () => {
  const taskData = {
    title,
    dueDate: dueDate?.toISOString(),
    priority,
  };
  try {
    await AsyncStorage.setItem(`draftTask-${userId}-${scheduleId}`, JSON.stringify(taskData));
    console.log('Draft task saved:', taskData);
    Alert.alert('Draft Saved', 'Draft task has been saved locally.');
    navigation.dispatch(CommonActions.navigate('Home'));
  } catch (error) {
    console.error('Error saving draft task:', error);
    Alert.alert('Error', 'Error saving draft task.');
  }
};
```

3. Reusability

This principle emphasises creating components or modules that can be reused in different parts of the application or in different projects. Reusability helps to reduce duplication, enhance maintainability, and improve the scalability of the code.

Reusable components such as CircularDropdown Menu and the Navigation Bar were coded once and saved under the "components" folder.



Both were then referenced multiple times in the code.

```
<View style={styles.tasksContainer}>
  <View style={styles.tasksHeader}>
    <Text style={[styles.tasksHeaderText, { color: theme.colors.text }]}>My Tasks & Events</Text>
    <CircularDropdown
      icon="add"
      options={[
        { label: 'Task', value: 'task' },
        { label: 'Event', value: 'event' },
      ]}
      onSelect={handleDropdownSelect}
    />
  </View>
  {loading ? (
    <Text style={[styles.loadingText, { color: theme.colors.text }]}>Loading...</Text>
  ) : (
```

```
    </View>
  </View>
  <NavBar navigation={navigation} userId={'yourUserId'} scheduleId={'yourScheduleId'} />
</View>
;
```

4. Security and Environment Configuration Management

This principle involves managing configuration settings and sensitive data in a secure and organised manner. It helps ensure that sensitive information such as API keys are not exposed in the source code and are securely stored. Using environment variables and configuration files, you can keep your API keys and other sensitive information out of your codebase, enhancing security and simplifying configuration management.

Code snippet showing the firebaseConfig.js file referencing the API keys from a .env file.

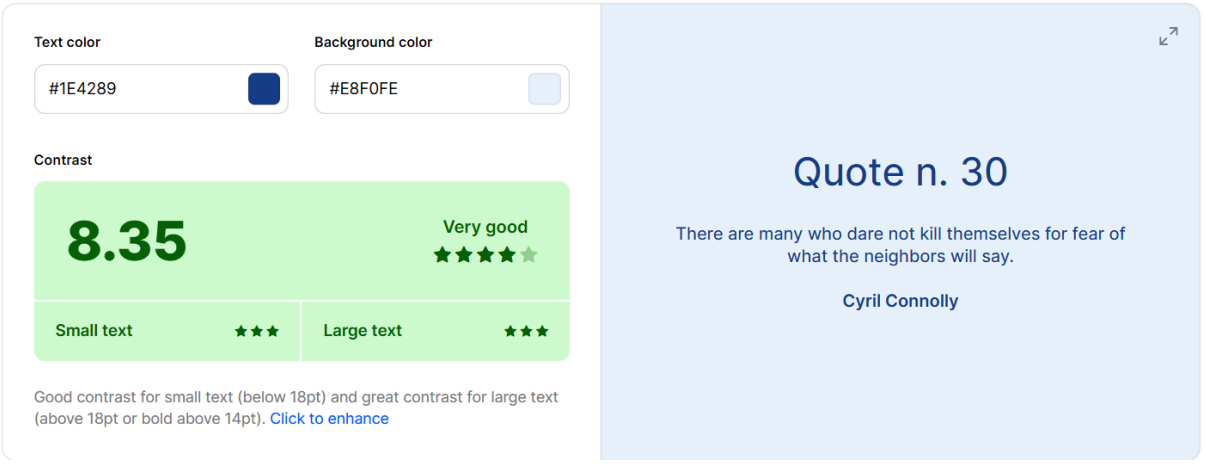
```
const firebaseConfig = {
  apiKey: process.env.REACT_APP_FIREBASE_API_KEY,
  authDomain: process.env.REACT_APP_FIREBASE_AUTH_DOMAIN,
  projectId: process.env.REACT_APP_FIREBASE_PROJECT_ID,
  storageBucket: process.env.REACT_APP_FIREBASE_STORAGE_BUCKET,
  messagingSenderId: process.env.REACT_APP_FIREBASE_MESSAGING_SENDER_ID,
  appId: process.env.REACT_APP_FIREBASE_APP_ID,
};
```

As more features are implemented more software engineering principles can also be implemented along with them. Furthermore, more practices such as Continuous Integration and Continuous Development (CI/CD) are meant to be implemented in Milestone 2 as automated testing will also be implemented. We have used Git repositories (along with Github) and Firebase (for the Firebase Analytics feature) to expedite the implementation of CI/CD pipeline in the upcoming milestones.

5. Material Design Principles

As part of my software engineering practices, we have followed Material Design principles in TaskMaster. This approach ensures a consistent and intuitive user experience, with proper contrast for enhanced visibility. By adhering to these guidelines, we have created a visually cohesive and user-friendly interface that is easy to navigate and interact with across different devices and screen sizes.

Screenshot showing the proper contrast between text and background in light mode being checked.



Testing & CI/CD

After completing user testing for the current features of TaskMaster, we are now focusing on unit, component, and integration testing. For these tests, we are using Jest to ensure that individual pieces of code function correctly and React Native Testing Library to make sure that components work together seamlessly. To streamline the testing process, we have started configuring GitHub Actions for continuous integration and automated testing, enabling efficient and reliable workflows. Additionally, we plan to use Detox for end-to-end (e2e) testing to simulate real user interactions and validate the complete functionality of the app in a production-like environment. By implementing these testing strategies, we aim to ensure that the app is robust, reliable, and user-friendly. Our tests should be fully complete in Milestone 3.

Future Plans

As previously mentioned, one of our future plans is to complete unit, component, integration and end to end testing using the technologies mentioned above. We also have plans to add the following features:

Sleep tracking using Machine Learning

We plan to integrate a simple linear regression model using TensorFlow for sleep tracking. This feature will record users' sleep start and end times based on their input, similar to how free time slots are added. The goal is to monitor sleep patterns, ensuring users get sufficient rest and identifying any anomalies in their sleep schedule.

Reminders

To alert users of upcoming tasks and events, we intend to implement a reminders feature using Firebase Cloud Messaging (FCM). FCM enables the app to send notifications to users from the server, even when the app is not active. This functionality will be instrumental in ensuring users stay informed about their schedules.

Synchronisation

Our initial approach for synchronisation involved users downloading their NUSMods timetables and uploading them to TaskMaster. However, we discovered the NUSMods API, which offers a more seamless integration. Additionally, we plan to synchronise with other platforms such as Google Calendar and Microsoft Outlook Calendar. This integration will provide users with a comprehensive view of their tasks across different platforms.

User collaboration

We aim to implement a user collaboration feature that allows users to view and add friends. This feature will enable users to view their friends' schedules and identify common free time slots. Our Firestore database structure, which includes a collection of user documents, facilitates this implementation. We also plan to use a Firebase search engine extension to enable users to search for their friends efficiently.

Time Management Analytics

We are planning to add an analytics page to provide users with an overview of their time management. This feature will involve creating additional fields in each user’s Firestore

document, which will be updated as tasks, events, and sleep are logged. The analytics will likely be displayed in the ProfileScreen, offering users valuable insights into their time management practices.

These enhancements aim to provide a robust and user-friendly experience, ensuring efficient task management and insightful analytics for our users.

Others

Furthermore, we also aim to improve on the limitations we have stated below.

Challenges Faced

Throughout the development of our app, we encountered several challenges that significantly contributed to our understanding of mobile application development. Below, we outline four major challenges and our approaches to overcoming them.

Choosing the tech stack

One of the initial hurdles was selecting the appropriate technologies for each component of the app. Given the multitude of options and our limited prior experience, we had to rely heavily on online resources to make informed decisions. After extensive research, we reached a consensus on the technologies to use. For instance, we debated the necessity of Expo, considering we could run and test the app on the Android Studio Emulator. Additionally, we deliberated between Flutter and React Native, noting Flutter's superior integration with Google services like Firebase. In hindsight, both choices proved advantageous: Expo streamlined our workflow and simplified the APK building process, while React Native provided abundant resources and support crucial for beginners.

Steep learning curve

The adoption of various technologies introduced a steep learning curve. Initially, we planned to follow tutorial videos on YouTube, but soon realised the importance of understanding the fundamentals of React Native and JavaScript before diving into coding. This preparatory phase ensured we were better equipped to handle syntax-related errors. Structured two-week sprint plans and regular video meetings were instrumental in maintaining our schedule and mitigating miscommunication. These strategies facilitated a coordinated approach and a clearer path forward.

Complexity of User Interface (UI) Design

We underestimated the complexity of designing an intuitive user interface. Choosing appropriate colours for both light and dark themes proved challenging as we adhered to Material Design guidelines, emphasising proper contrast and visibility. To address this, we examined existing apps in the market and drew inspiration from their designs. This comparative analysis helped us create a more user-friendly and aesthetically pleasing UI.

Need for Large Language Models (LLMs) in the auto scheduling feature

Initially, we considered leveraging external software for the auto-scheduling feature, doubting our ability to develop a robust algorithm in-house. We experimented with LLMs such as ChatGPT and Gemini, which provided satisfactory results. However, implementation issues

arose due to payment barriers: our credit cards were declined for OpenAI, and the Gemini API required a subscription. Consequently, we decided to develop our own algorithm for auto-scheduling. To our surprise, our solution performed better than expected, demonstrating the value of perseverance and innovation in overcoming resource limitations.

Current Limitations

Despite the significant progress in our app development, there are still some areas that need improvement. We have identified two major limitations that we are currently addressing.

Uneven Auto Scheduling

Our auto-scheduling feature is designed to efficiently sort and fit tasks into available time slots. However, it may not always distribute tasks evenly across these slots. This can result in certain time slots becoming overloaded with tasks while others remain underutilised. To address this issue, we are actively working on refining our algorithm to ensure a more balanced distribution of tasks across all available time slots.

Inefficient sign in

Another limitation is the initial sign-up or sign-in process, which currently requires users to enter their full email address. We have explored options to simplify this process, such as incorporating Google Account sign-in. However, we encountered challenges in integrating the Asynchronous Storage feature necessary for speeding up subsequent sign-ins with Google Accounts. Due to these difficulties, we had to remove the Google Account sign-in option. We are continuing to explore solutions to enhance the efficiency and user experience of our sign-in process.

Insufficient Machine Learning Model (Anticipated)

Due to our current level of expertise in machine learning, we recognize that a simple linear regression model may not be entirely adequate for accurately tracking a user’s sleep cycle. Consequently, we will inform users about the limitations of this model, emphasising that they remain responsible for managing their sleep schedules.

Project Management

We tried to follow the Agile methodology, more specifically Scrum. While we did not take up Scrum roles, we relied heavily on sprints. Our sprints were 2 weeks long, each with detailed plans. Moreover, we also adhered to Scrum practices such as daily meetings (through video call) and reviewed and tested our app at the end of the sprint.

Project logs

Please find the project logs here.