Smart Personal Assistant (SPA)

Wong Ja Yee
Chan Zhi Yang

Proposed Level of Achievement: Apollo 11

Table of Contents

Introducing

SPA, Your Smart Personal Assistant

Wong Ja Yee, Chan Zhi Yang

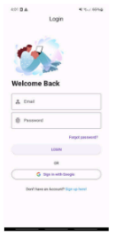
Motivation and Aim

Ever feel clustered and in a mess with too many mobile applications responsible for your work and productivity? Well our Smart Personal Assistant (SPA) is the solution just for you.

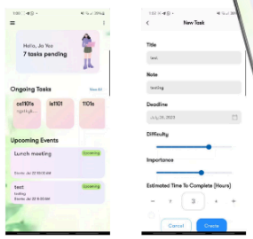
Whether you're a busy professional, a student, or just looking to stay organized, a personal assistant app can help take a breather and make everyday feel like a SPA.

User Interface

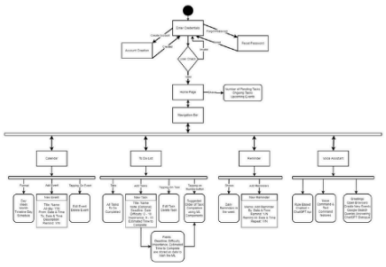
Authentication



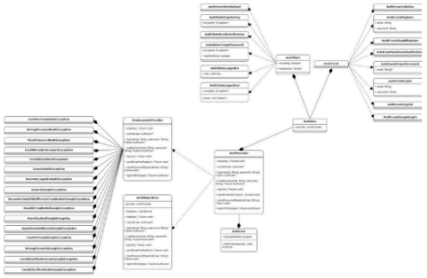
Core Features



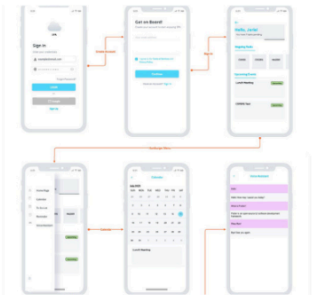
Diagrams



Activity Flow Diagram



BLoC Management



Wireframing

System Design

Features

1. Calendar
2. To-do list
3. Task sequence suggester
4. Reminders & Notifications
5. AI Assistant

To-do List

To-do List help users keep track of their tasks. Tasks created have the fields: title, note, deadline, difficulty, importance and estimated time required. Using a sequential model, it will recommend the order in which users should complete their tasks

Calendar

The calendar help users keep track of upcoming events. In the calendar page, users can swap to different types of calendar view such as day, week, month and more to view their schedule

Reminder

Users can create reminders for themselves. They can set a deadline for a reminder and reminders would be sorted by deadline in the reminders view. They can also choose the remind function, where a notification would be sent at the indicated timing. In addition, notifications can be repeated daily, weekly and monthly.

AI Assistant

Provides users web search, information gathering, and communication through voice or text inputs. It is also able to create events for users that would be reflected in the calendar

SWE Practices

- Version Control and Project Management with GitHub
- Abstraction and separation of frontend and backend
- Reusable UI Elements

State Management

In this application, 3 types of state management were used: BLoC, GetX and Provider. They were chosen for specific parts of the application that they best suit.

- BLoC - authentication.
- GetX - navigation
- Provider - update widgets

Testing

User, unit and widget testing were used. Unit and widget testing comprised of a mix of manual and automated testing using Flutter's inbuilt packages

Motivation

In today's digital age, individuals often find themselves overwhelmed with numerous mobile applications responsible for work and productivity. Switching between applications, remembering various logins, and managing multiple interfaces can lead to confusion, forgetfulness, and decreased efficiency. SPA aims to address this problem by streamlining the user experience, reducing cognitive load, and providing a comprehensive solution for everyday tasks.

Aim

The Smart Personal Assistant (SPA) project aims to create a mobile application that integrates multiple daily-use applications into a single platform.

By providing a unified and convenient solution, SPA will help users enhance their productivity, improve organization, and reduce the need for managing multiple applications simultaneously. With features like voice commands and natural language processing, a personal assistant app can also make it easier for users to access information and complete tasks on-the-go.

Core Features

1) Voice Assistant
2) To-do List
3) Calendar
4) Reminders

Tech Stack

1) Flutter (Frontend)
2) Firebase (Backend)
3) Python (Machine Learning)
4) TensorFlowLite (Machine Learning)

User Stories

1) As a student with multiple assignments and tasks, I want to be reminded of upcoming deadlines and efficiently manage my workload in a single application.

2) As a student with many deadlines and assignments, I want to make deciding the order in which to finish these easier as I spent quite some time figuring out which task I should complete first

3) As a professional, I rely on several productivity applications. I desire a consolidated solution where I can access all the necessary tools, automate repetitive tasks, and stay organized without the hassle of switching between multiple apps.

4) As a user who values convenience, I want to utilize voice commands to accomplish tasks quickly and effortlessly. Being able to search for information using natural language and receive accurate and relevant results would greatly enhance my productivity.

Project Scope

The project can be broken down into 2 parts: Setup and Core

Setup: User registration and login
Core: Main page and core features, including calendar, to-do list, reminders and a voice assistant.

Software Engineering Process

The development of SPA will involve the following key steps: Requirement Gathering & Planning, Design, Coding and Testing

Requirement Gathering & Analysis

Process of identifying requirements of SPA from start to finish:

We identified that our main stakeholders will be students and working adults who would have to increasingly rely on productivity apps to manage their busy daily lives. Our initial plan was to develop a one-stop application with calendar, to-do list and reminder functions. However, after we gathered feedback from students and working adults, we recognised the critical need to improve our application by integrating additional features such as a task suggestion manager, notifications and a voice assistant to perform tasks for the user and allow them to communicate their needs. By combining these features into a single productivity platform we believe that our application will be an indispensable tool for users seeking to improve their goals efficiently.

System Design

The following are the project requirements that make up our mobile application:

Frontend Development:

1) Log-in Page
2) Calendar Function
3) To-Do List Function + Tasks Suggestions with Machine Learning
4) Reminders Function + Notifications
5) Rule Based Voice Assistant + ChatGPT

UI/UX Design:
Using Flutter to create an intuitive and user-friendly interface that combines elements from various applications while maintaining consistency and ease of use. At the same time we researched on principles of design in order to appeal to our users.

Backend Development:
Using Firebase with Flutter to implement the core functionality, including task and deadline management, integration of voice commands, and saving of user data.

Deployment and Release:
Prepare the application for deployment on major mobile platforms on Android and release it to the Google Play Store.

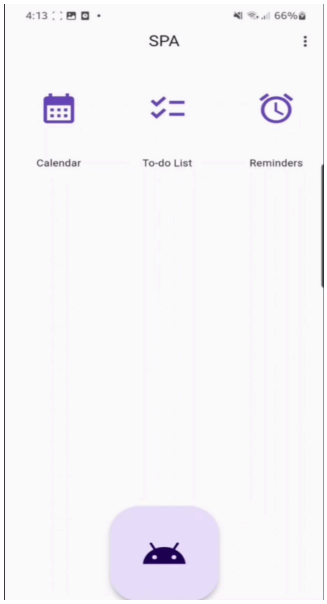
Task Management integrated with Machine Learning :
SPA will combine a to-do list with a calendar, enabling users to record and manage tasks and deadlines seamlessly. Users will receive reminders when a deadline approaches, ensuring they stay on track. Our application also integrates machine learning components in order to provide users with suggested optimal sequence of tasks to complete.

Voice Assistant:
SPA will incorporate voice commands to assist users in performing tasks. Through a natural language processing (NLP) model, the application will process user queries, and provide relevant information via a rule-based chatbot. Users will also be able to use ChatGPT as a chatbot for assistance.

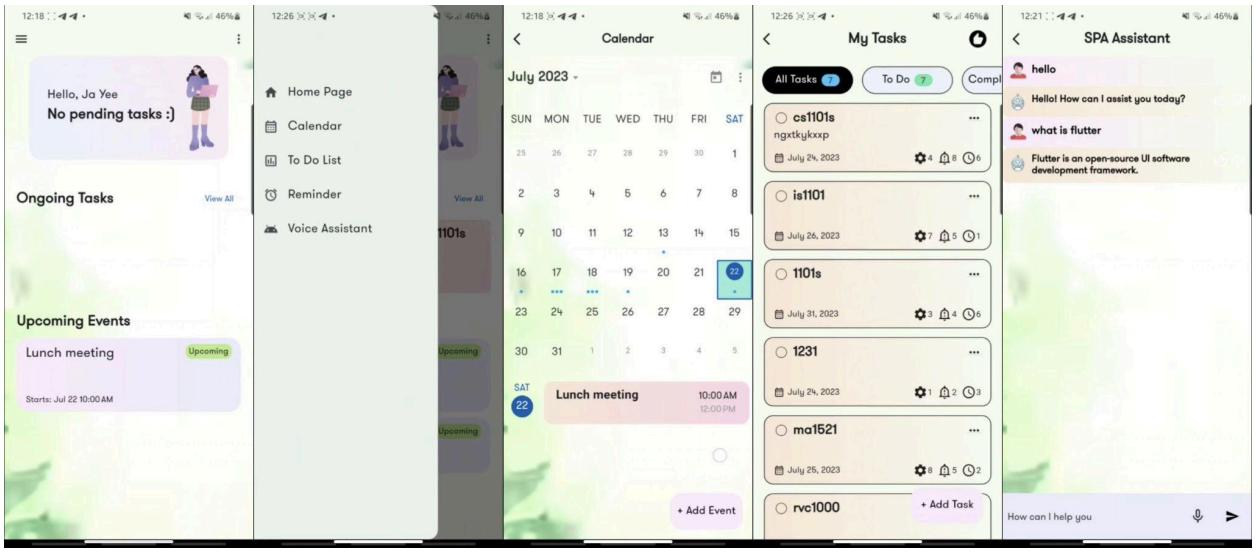
Testing and Refinement:
Conduct thorough testing to ensure the application's stability, reliability, and usability. Collect user feedback to identify areas for improvement and implement necessary refinements. Testing includes; unit testing, widget testing, user testing, system testing, and automation.

Frontend Design
We envisioned the project through the eyes of the intended users to ensure that our software is feasible and understands the user's concerns and needs.
Our homepage has 3 main functions, calendar, to-do list and reminders, whereas the voice assistant function is also easily accessible at the bottom.
We planned for a light blue, purple and green colour scheme as based on research, both colors allow the user to feel relaxed. (Like a SPA).

Our initial design has three icons on the top of the page, and the voice assistant is easily accessible at the bottom.



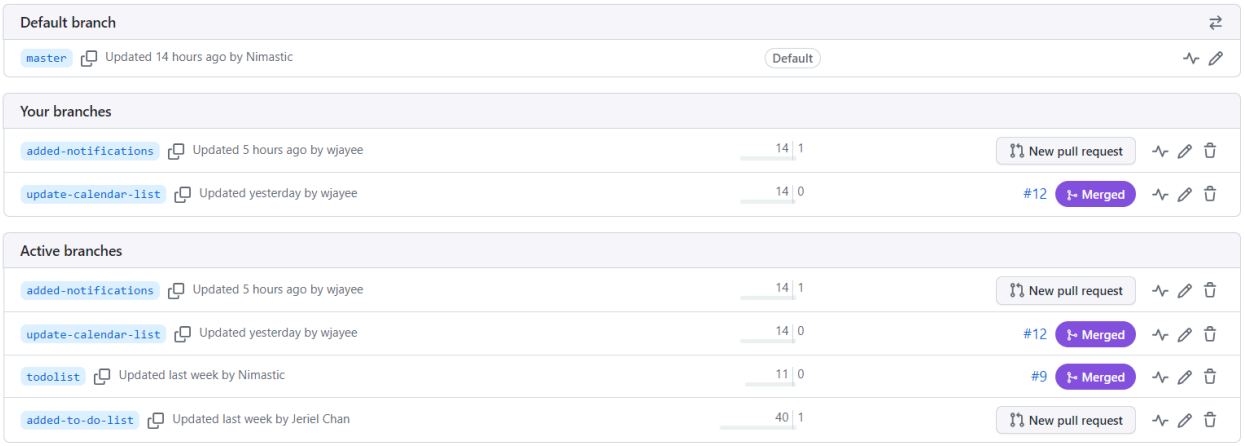
After Milestone 2, we decided to improve on the UI design based on feedback. Upon pressing the hamburger icon on the top left of the page, a navigation drawer will slide out revealing the features that our application has to offer. Tapping on any of the features will direct the user to the intended pages of the feature.



Software Engineering Practices & Implementations

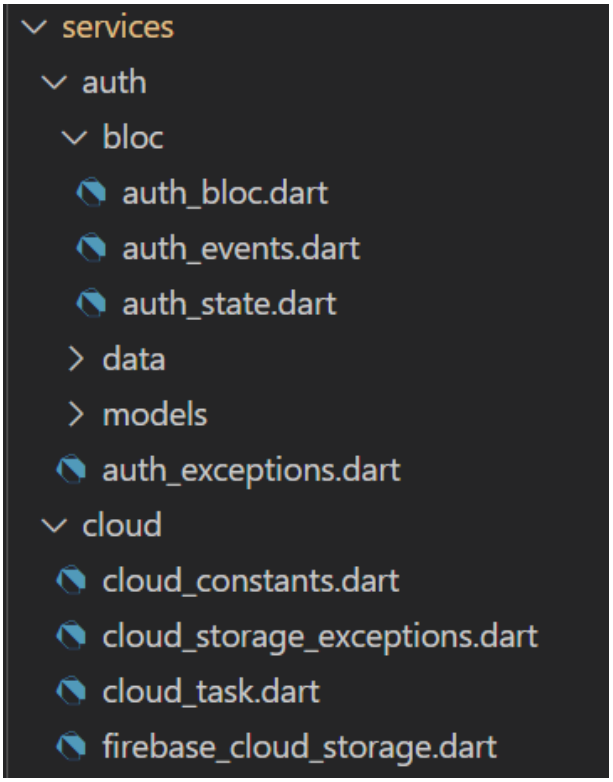
1) **Version control and project management with Github**

Using Github's branching feature, we worked on our own branches instead of pushing directly to the master branch to avoid any code from being deleted by accident. Using pull requests, we would update the master branch from our own branches.



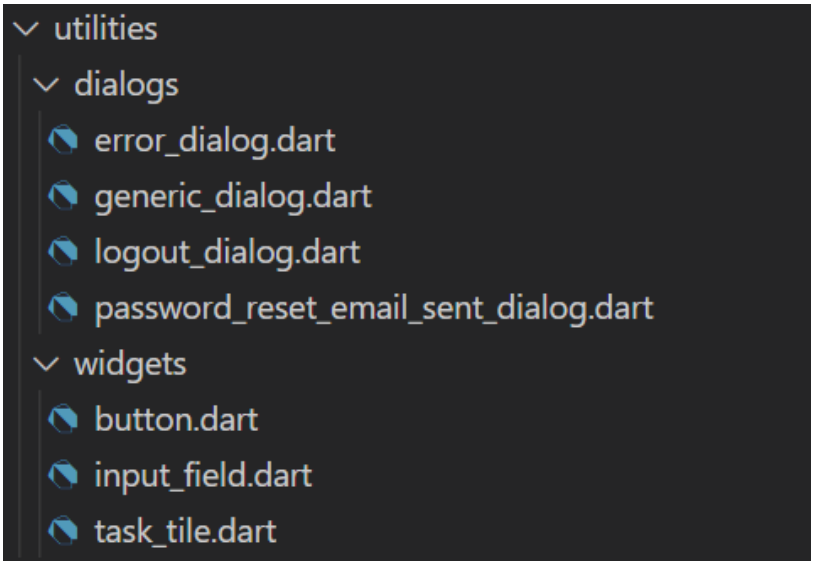
2) Abstraction and separation of frontend and backend

Using Flutter libraries such as GetX and BLoC, this helps to abstract the backend away from the user interface. BLoC is used to abstract the authentication logic away from users. GetX is used to manage the states of the application after authentication. Since Firebase is used for its cloud storage, separate files for any calls to the database were created to further abstract and separate backend from the frontend.



3) **Reusable UI elements**

Since Flutter is built using widgets, certain widgets can be reused in the applications, making the code cleaner and easier to read.



Testing

User, unit and widget testing were carried out. Some unit and widget tests are tested using flutter's test package, flutter_test to automate testing while others are tested manually.

One example of automated unit testing that we did is to test calls to firestore, such as creating a new document, getting, updating existing documents and deleting existing documents. (CRUD)

```
Run | Debug
test('Successfully created a task and return a CloudTask', () async {
  // Arrange.
  when(mockCollection.add(dataFields)).thenAnswer((_) async
      => mockDocumentReference);
  when(mockDocumentReference.get()).thenAnswer((_) async
      => mockDocumentSnapshot);
  // Simulate the data returned from Firestore when the document is fetched.
  when(mockDocumentSnapshot.id).thenReturn('test_document_id');
  when(mockFirebaseCloudTaskStorage.createNewTask(ownerUserId: 'test_owner_user_id'))
      .thenAnswer((_) async => const CloudTask(
                                documentId: 'test_document_id',
                                ownerUserId: 'test_owner_user_id',
                                title: 'test_title',
                                text: 'test_text',
                                deadline: 'test_deadline',
                                difficulty: 0,
                                importance: 0,
                                timeRequired: 0,
                                isCompleted: 0,
                               )
  );

  // Act
  CloudTask cloudTask = await mockFirebaseCloudTaskStorage.createNewTask(ownerUserId: 'test_owner_user_id');

  // Assert that the created CloudTask has the expected values.
  expect(cloudTask.documentId, 'test_document_id');
  expect(cloudTask.ownerUserId, 'test_owner_user_id');
});
```

In this example, we are testing the creation of a document in Firestore. Since calls to Firestore cannot be made in testing, we used a flutter package, Mockito, to mock several classes in our application, such as Firestore, our class that contains all the calls to Firestore, FirebaseCloudTaskStorage, and many more. Using Arrange, Act and Assert to test the function, we first mock the function we are testing: createNewTask. Next, we call the actual function and assert that the created value has the expected values.

One example of automated widget testing is to test the text shown on the main page of our application.

```
Run | Debug
testWidgets('Test tasks in Ongoing Tasks ListView', (tester) async {

  const ownerUserId = 'test_owner_user_id';
  const task1 = CloudTask(
    documentId: 'task_1',
    ownerUserId: ownerUserId,
    title: 'Task 1',
    text: 'Task 1 description',
    deadline: 'Task 1 deadline',
    difficulty: 0,
    importance: 0,
    timeRequired: 0,
    isCompleted: 0,
  );
  const task2 = CloudTask(
    documentId: 'task_2',
    ownerUserId: ownerUserId,
    title: 'Task 2',
    text: 'Task 2 description',
    deadline: 'Task 2 deadline',
    difficulty: 0,
    importance: 0,
    timeRequired: 0,
    isCompleted: 0,
  );

  Iterable<CloudTask> todoTasks = [task1, task2];

  await tester.pumpWidget(
    MaterialApp(
      home: MainPageTasksBar(
        context: createContext(),
        todoTasks: todoTasks,
      ) // MainPageTasksBar
    ) // MaterialApp
  );

  final task1Tile = find.text('Task 1');
  final task2Tile = find.text('Task 2');
  final noTile = find.text('Task 3');

  expect(task1Tile, findsOneWidget);
  expect(task2Tile, findsOneWidget);
  expect(noTile, findsNothing);

});
```

In the main page, there is a widget that shows the ongoing tasks that the user has. In this test, we are testing the function that returns a ListView Widget to ensure that it returns the correct tasks that are being passed into the function. By creating 2 tasks named task1 and task2, we passed both tasks into the function. We then test if there exists a widget that contains the title of these 2 tasks. We also test to make sure there is no other widget that exists that contains the title of any other tasks that was not passed into the function.

Frontend Testing

Authentication

Test Type	Test	Pass/Fail
Widget	“Login” button leads to the Main Page	Pass
Widget	“Sign in” button leads to Sign In Page	Pass
Widget	“Forgot Password” button leads to Forget Password Page	Pass
Widget	“Submit” button leads to Email Verification Page	Pass
Widget	“Log in here” button leads to Login Page	Pass
Widget	“Back to login page” leads to Login Page	Pass
Widget	“Click here” leads to Login Page	Pass

Main Page

Test Type	Test	Pass/Fail
Widget	“Log Out” button leads to Login page	Pass
Widget	Hamburger Button opens the navigation drawer	Pass
Widget	Calendar button leads to Calendar List	Pass

	Page	
Widget	To-do List button leads to To-do List Page	Pass
Widget	Reminder button leads to Reminder List Page	Pass
Widget	Voice Assistant Button leads to Voice Assistant Page	Pass
Widget/Unit	Header widget in Main Page displays the correct username and to do tasks	Pass
Widget/Unit	Task bar in Main Page displays uncompleted tasks	Pass
Widget	View All button leads to To-do List Page	Pass
Widget/Unit	Event bar in Main Page displays upcoming events or events that just ended	Pass
Widget	Pressing tasks/events in the Main Page leads to Edit Task/Event page	Pass

Calendar Page

Widget	Add Event Button leads to Add Event Page	Pass
Widget	Users can scroll through the calendar list sideways	Pass
Widget	Event display the title and description written by users	Pass
Unit	Events with specific dates only appear on those dates on the calendar	Pass
Unit	Deleted events are not displayed	Pass

Widget	Clicking on events opens bottom sheet (only for month view)	Pass
Widget/Unit	Clicking on edit event leads to Edit Event View	Pass
Widget/Unit	Clicking on delete event deletes the event from the calendar	Pass
Widget	“Close” button only closes the bottom sheet	Pass
Widget	Clicking the kebab menu opens dropdown menu of calendar views	Pass

To Do List Page

Widget	Add Task Button leads to Add Task Page	Pass
Widget	Tasks display its title, note, deadline, difficulty, importance and time required	Pass
Widget/Unit	Tasks are marked as completed when the checkbox is pressed	Pass
Unit	Deleted tasks are not displayed	Pass
Widget	“Close” button only closes the bottom sheet	Pass
Widget	Clicking the kebab menu opens dropdown menu of actions	Pass
Widget/Unit	Clicking ‘Edit’ leads to Edit Task View	Pass
Widget/Unit	Clicking ‘Delete’ deletes the task from the page	Pass
Widget/Unit	Clicking on the different tabs(all tasks, to do, completed) shows the respective events	Pass

Reminder

Widget	Add Reminder Button leads to Add Reminder Page	Pass
Widget/Unit	Users can scroll through the calendar sideways	Pass
Widget	Reminders display the title and note written by users	Pass
Unit	Reminders are sorted by date	Pass
Widget/Unit	Reminders are marked as completed when the checkbox is pressed	Pass
Unit	Deleted reminders are not displayed	Pass
Widget	Clicking on the reminder display leads to Edit Reminder View	Pass

Voice Assistant View

Widget	Microphone Button glows when users hold it	Pass
Widget/Unit	Clicking the send displays the typed message on the chat	Pass
Widget	Loading icon appears when the assistant is formulating a response	Pass

Backend Testing

Authentication

Test Type	Test	Pass/Fail
Developer/Unit	Users are able to register an account/log in with valid credentials	Pass

Developer/Unit	Email verification is sent when users register/log in without verifying their email	Pass
Developer/Unit	Request to reset password sends a password reset link to the email	Pass
Developer/Unit	Request to verify email sends an email verification link to the email	Pass
Developer/Unit	Users are able to log in with existing Google accounts	Pass

Main Application

Test Type	Test	Pass/Fail
Developer/Unit	Only tasks/events/reminders that are created by the user is displayed	Pass
Developer/Unit	Creating a task/event/reminder updates the database	Pass
Developer/Unit	Editing a task/event/reminder updates the database (Includes completing tasks)	Pass
Developer/Unit	Deleting a task/event/reminder updates the database	Pass
Developer/Unit	Reminders/Events send a notification at the remind at time	Pass
Developer/Unit	Reminders with notifications set to repeat send a notification at the set time every day/week/month	Pass
Developer/Unit	Editing a reminder/event with notifications reset the notifications when reminder/event is saved	Pass
Developer/Unit	Voice commands given to the assistant are displayed	Pass

Developer/Unit	Responses by the assistant are displayed	Pass
Developer/Unit	Past conversations with the assistant are displayed unless application is refreshed	Pass
Developer/Unit	Responses by the assistant are played out loud	Pass

User Testing

For User Testing, we gathered feedback from students currently studying in NUS, NTU & SUSS. We provided them with the following prompt:

- Explore the application and provide feedback on its usability, accessibility and ease of use.
- Test out the creation of Events, Tasks, and Reminders functions
- Talk/Type to the chatbot (greetings, queries, creation of events; date must be specified)
- Explore the suggested completion of task sequence feature
- Bugs/Errors

These users generally agreed that the application is easy to use and is user friendly. However, some users did face a few difficulties with the voice assistant due to the irregular response of the voice assistant. Nevertheless, the voice assistant generally works in the creation of events and the answering of queries. As for the other features, all users did not find any errors or bugs while testing them out. Below are the specific responses of 2 users that participated in our user testing:

User 1: Jun Rong
I am able to log into the application really smoothly after creating an account. I was also able to create a task and event reminder. However, for the chatbot, I am able to use the voice message the first time, and after that I'll have to do manual typing. Only works again once I restart the application. Application is very easy to navigate.

User 2: Jiayi
To Do List is easy to navigate and adding tasks is simple. A little confused on where to add events, but realised it was at Calendar tab afterwards. Reminder is a cool idea, thought it will show up on homepage, but was told it's meant for notifications. Clock to adjust time is small, overall design is really nice. Would be nicer if we're able to choose the colours for our own tasks as I like to colour code. Description function is good as it helps me remember important stuff. Chatbot sounds robotic but works well. When I try to talk to the bot, it will tell me to please type a message. Navigation wise it's pretty easy but I might be a bit lost if I did not know how to use the sidebar.

Milestone 1 - Setup

1) **Frontend page routing**

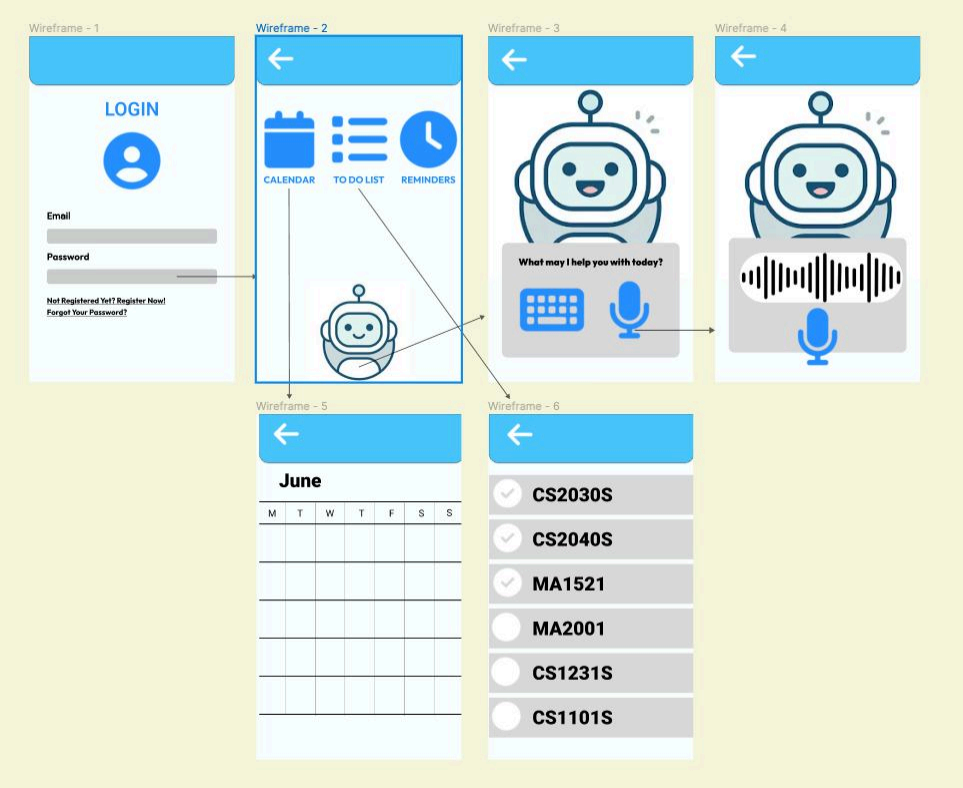
Route users between different authentication pages using BLoC, a Flutter state management library.

2) **Backend Database**

Implement calls to Firebase through BLoC for user registration and log in.

3) **UI Planning**

Used Canva to plan the layout of the user interface and color schemes to be used



Milestone 2 - Core

1) Main Page

After successfully logging in, users are brought to the main page where it is mainly used to navigate between the different features of the application.

2) Voice Assistant Feature

Using the flutter libraries speech-to-text and url launcher, users can give voice commands and depending on the command given, they can be directed elsewhere.

3) To-do List Feature

Since the tasks collection is shared by both to-do list and calendar, this feature is used to layout all the tasks that the user has created in a single page for easy viewing.

4) Calendar Feature

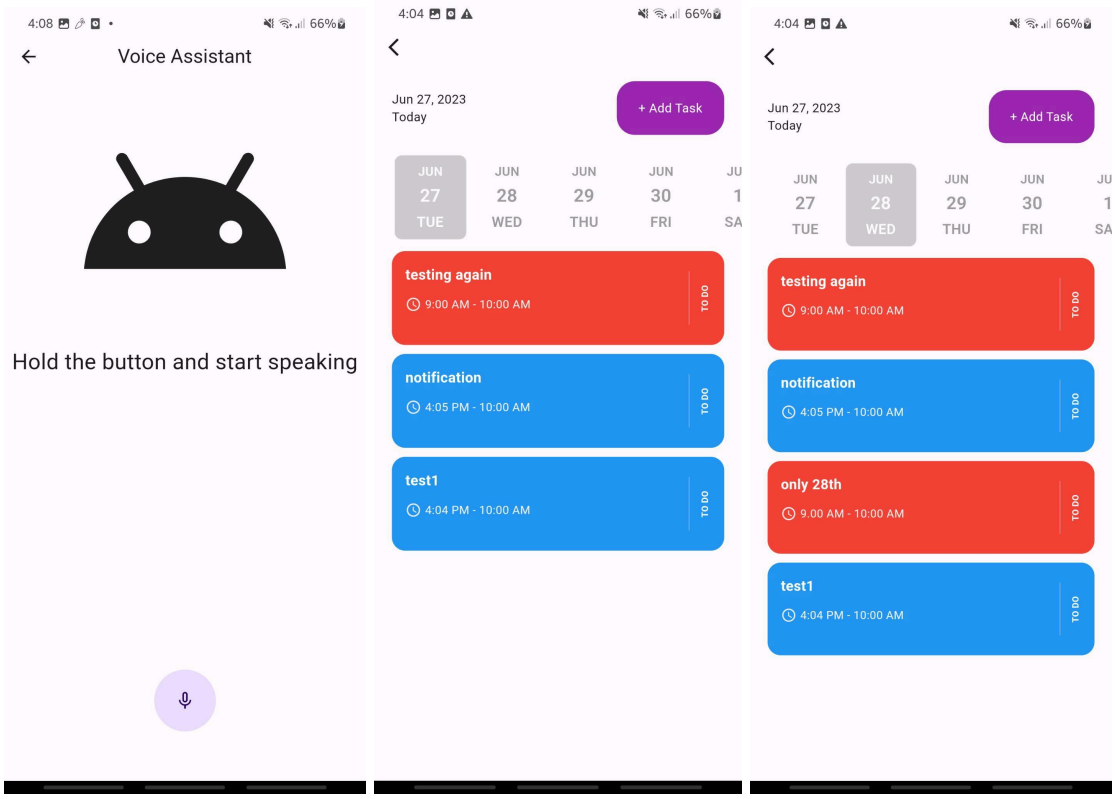
Implement a calendar in the form of a list view for users to view their tasks according to the date of the task. Users can only view the tasks that they have created with the logged in account rather than all tasks in the database, which could be created by different users.

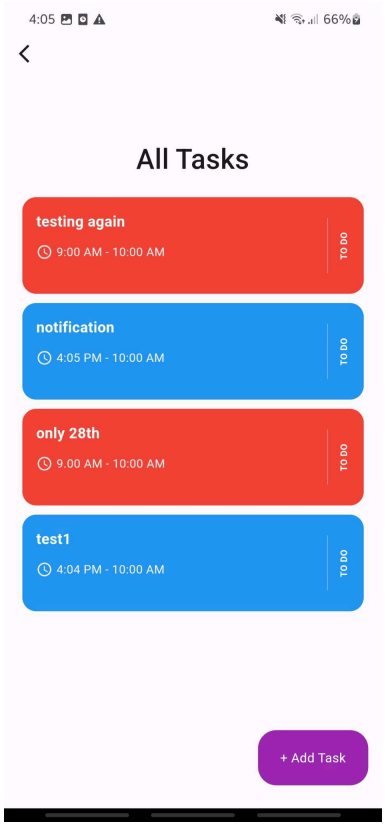
5) Authentication UI

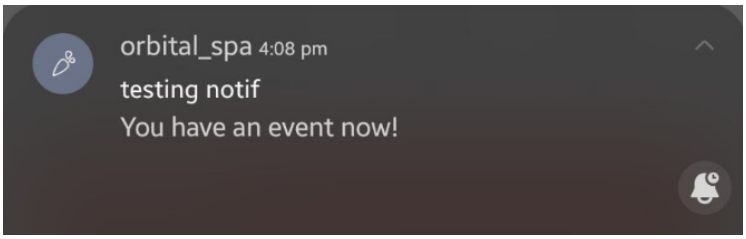
Update the authentication user interface to include more information for users and to

6) Notifications (Reminder) Feature

Using flutter libraries flutter_local_notification, flutter_native_timezone_updated_gradle and more, notifications can be sent based on the stated timing on the task as a reminder to users. Users can choose to send daily reminders that would send notifications at the same time everyday.







Milestone 3 - Core Improved

1) Main Page

The main page interface was redesigned to display current tasks and events and help users take note of their pending tasks easily.

2) Calendar

More calendar views were added to the calendar page: day, week, month, timeline day and schedule. The design of the calendar page is also improved for users to easily view their upcoming events.
Events were edited to include new fields such as all day event and remind.

3) To Do List

The to do list view was updated to allow the user to view all tasks, to do and completed tasks to prevent the page from getting too cluttered with undeleted but completed tasks. Uncompleted and completed tasks are also color coded for

easier viewing. We chose not to delete completed tasks as it can help some users to keep track of what they have completed and hopefully give them a sense of accomplishment to further motivate them to finish all their tasks.

Tasks now have more fields and these include: title, note, deadline, difficulty, importance, and estimated time required. More fields were added for our next feature, task suggestion.

4) Task Suggestion

This feature suggests the order in which users should complete their task. Using tflite_flutter, a package by Tensorflow in Flutter, we trained our own model and used this package in our application.

When deciding on a model to use, we were deciding between a sequential model or a recurrent neural network (RNN) model by tensorflow. After a few tests, we decided that a sequential model was more suitable as it better suited the type of input, output and process that we were looking for as compared to a RNN model.

Using Google Colab, we first started to train the model using our own training data.

```
import pandas as pd
import tensorflow as tf
# import ImageClassifierDataLoader
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import MinMaxScaler
from google.colab import files
# uploaded = files.upload()
```

```
# Step 1: Load and preprocess the data
data = pd.read_csv('test3.csv')
```

```
# Extract the relevant attributes (e.g., time required, time left to deadline) as features
X = data[['time_left', 'time_required', 'difficulty', 'importance']].values

# Create a MinMaxScaler object
dataScaler = MinMaxScaler()

# Apply Min-Max normalization
X = dataScaler.fit_transform(X)

y = data['order'].values
```

To create the training data, we randomly generated numbers from 1 to 10 (for difficulty, importance) and 1 to 20 (for time left and time required).
We then ordered the tasks based on these criteria:

- Deadline (time left) was prioritized.
- For tasks of the same deadline, importance was prioritized next.
- Based on our research, we could not fully prioritize difficulty over time required or vice versa

- We ordered these last 2 fields by weighing difficulty against time required. For example,
    - If a task has a high difficulty but a short amount of time required, we would prioritize this task first as there may be an underestimation by the user in terms of time required.
    - If there are multiple tasks (same deadline and importance) with the relatively similar difficulty, we would prioritize time required in that case

We then split the training data in a 80:20 ratio, where 80% is used to train the model and 20% is used to evaluate the model.

```
[ ] # Step 2: Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    X_train.shape
```

```
(32, 4)
```

```
[ ] # Step 3: Build the TensorFlow model
    model = tf.keras.Sequential([
        tf.keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dense(1)
    ])
```

```
[ ] # Step 4: Compile the model
    model.compile(optimizer='adam', loss='mse')
```

```
    # Step 5: Train the model
    model.fit(X_train, y_train, epochs=100, batch_size=32)
```

```
[ ] # Step 6: Evaluate the model
    mse = model.evaluate(X_test, y_test)
    print(f"Mean Squared Error: {mse}")
```

```
1/1 [==============================] - 0s 139ms/step - loss: 0.0022
Mean Squared Error: 0.0022123078815639 02
```

```
[ ] # Run predictions on the training data
    y_test_pred = model.predict(X_test)
    order = orderScaler.inverse_transform(y_test_pred)
    # .reshape(-1, 1)).flatten()

    # Print the predictions
    print("Training Data Predictions:")
    print(y_test_pred)
    print(order)
    print(orderScaler.inverse_transform(y_test.reshape(-1, 1)).flatten())
```

```
1/1 [==============================] - 0s 42ms/step
Training Data Predictions:
[[0.78646874]
 [0.34952074]
 [0.86584103]
 [0.83444595]
 [0.65304816]
 [0.20749411]
 [0.45983884]
 [0.5094864 ]]
[[31.672281]
 [14.631309]
 [34.7678  ]
 [33.543392]
 [26.468878]
 [ 9.09227 ]
 [18.933714]
 [20.869968]]
```

To use the model, we would pass in an array of tasks and the output would be an array of numbers.

```
[ ]  X_test1 = pd.read_csv('test1.csv')
```

```
[ ]  # Extract the relevant attributes (e.g., time required, time left to deadline) as features
     X_test1 = X_test1[['time_left', 'time_required', 'difficulty', 'importance']].values

     # Apply Min-Max normalization
     X_test1 = dataScaler.fit_transform(X_test1)
```

```
[ ]  # Run predictions on the training data
     y_test1_pred = model.predict(X_test1)
     order = orderScaler.inverse_transform(y_test1_pred)
     # .reshape(-1, 1)).flatten()

     # Print the predictions
     print("Training Data Predictions:")
     print(order)
```

```
1/1 [==============================] - 0s 41ms/step
Training Data Predictions:
[[ 1.2801281]
 [13.140207 ]
 [ 4.2487316]
 [40.747375 ]
 [ 7.95508  ]
 [24.005514 ]
 [39.269127 ]]
```

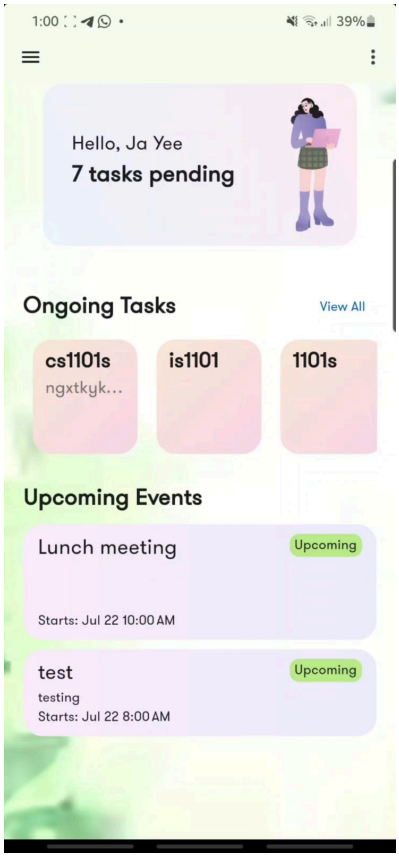
The numbers would then be sorted in ascending order and converted back to the task's title, to obtain the suggestion order of task completion.

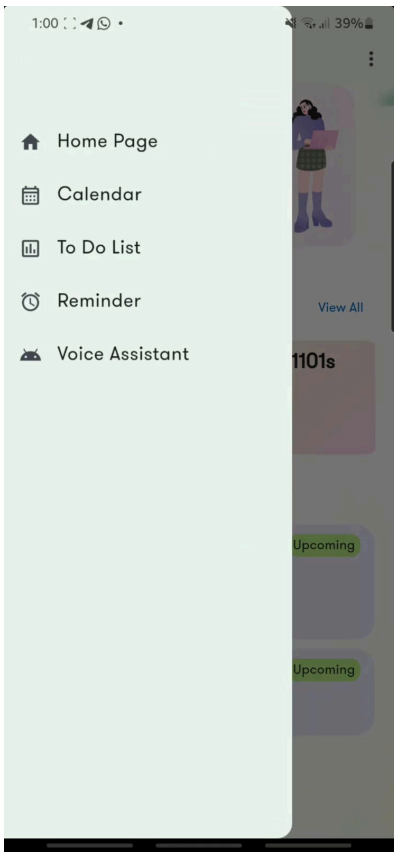
5) Reminders/Notifications

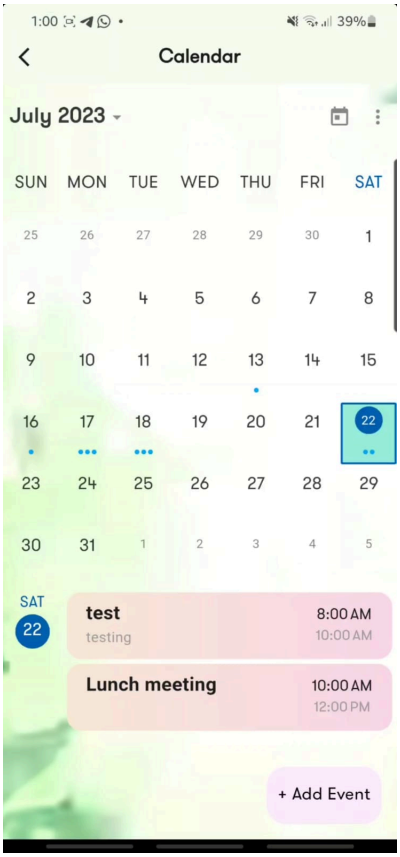
The reminder list was updated to sort reminders in ascending order (based on deadline) and notifications were updated to repeat weekly and monthly.

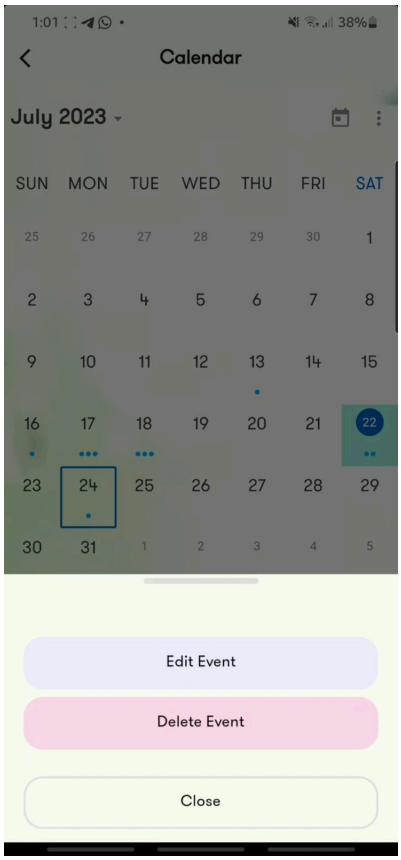
6) Voice Assistant

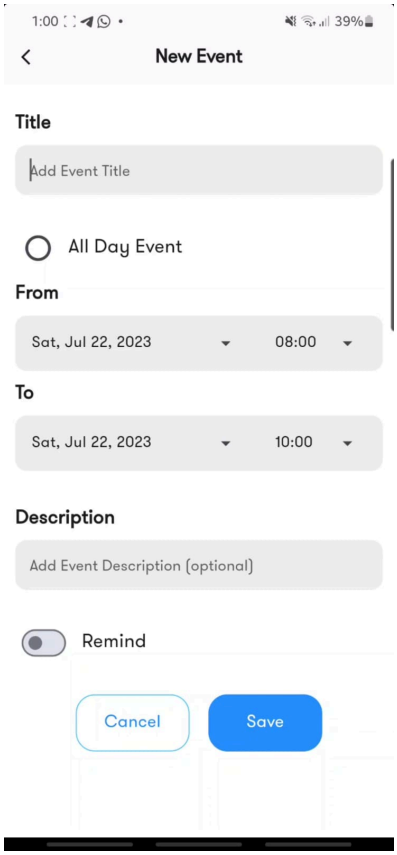
Switching away from a full rule-based voice assistant, we combined it with the use of ChatGPT to answer queries. In addition to voice input, we added a text input for users. Users will also be able to communicate with the bot, and request to perform tasks such as adding events. We also edited the user interface to remember previous conversations with the voice assistant.

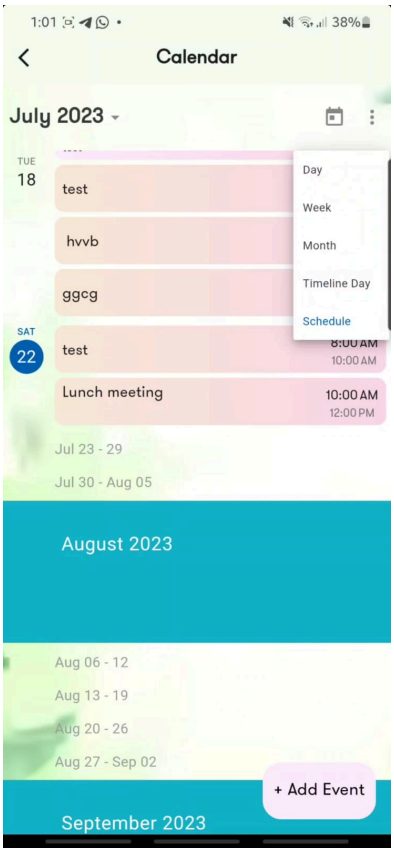

Hello, Ja Yee
7 tasks pending
Ongoing Tasks
View All
cs1101s
ngxtkyk...
is1101
1101s
Upcoming Events
Lunch meeting
Upcoming
Starts: Jul 22 10:00 AM
test
Upcoming
testing
Starts: Jul 22 8:00 AM

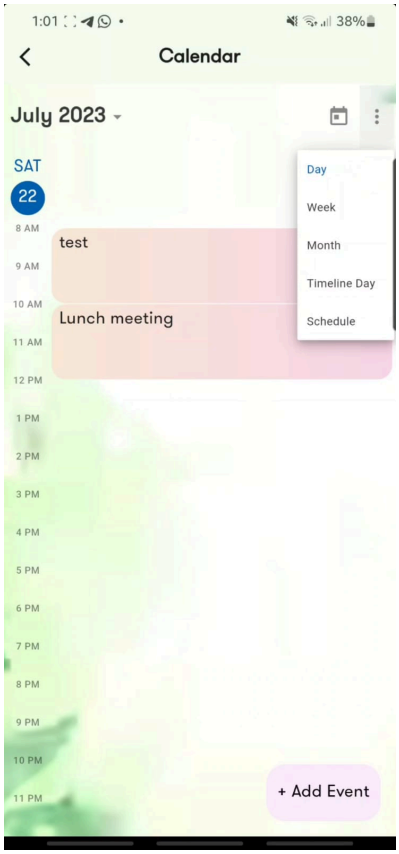

Home Page
Calendar
To Do List
Reminder
Voice Assistant
View All
1101s
Upcoming
Upcoming

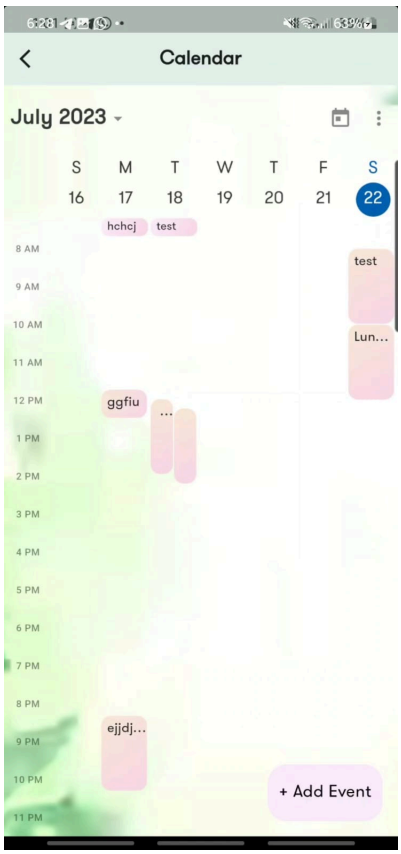

Calendar
July 2023
SUN MON TUE WED THU FRI SAT
SAT
22
test
testing
8:00 AM
10:00 AM
Lunch meeting
10:00 AM
12:00 PM
+ Add Event

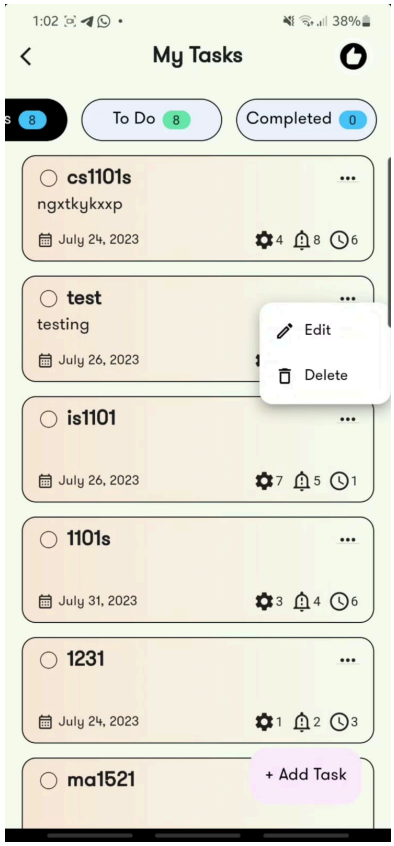

Calendar
July 2023
SUN MON TUE WED THU FRI SAT
Edit Event
Delete Event
Close

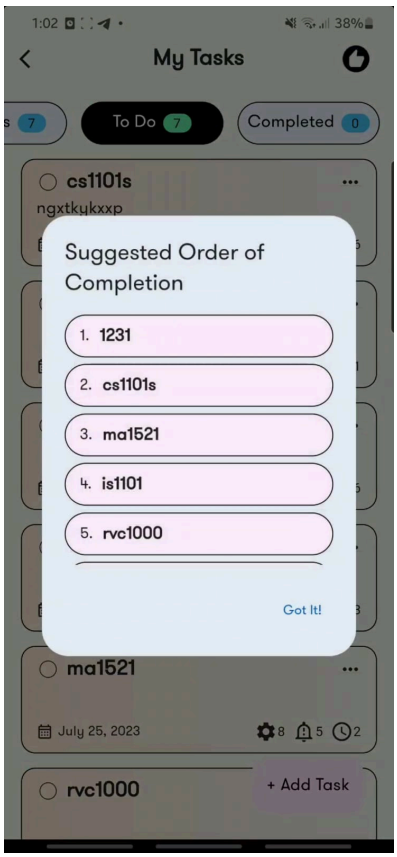

New Event
Title
Add Event Title
All Day Event
From
Sat, Jul 22, 2023
08:00
To
Sat, Jul 22, 2023
10:00
Description
Add Event Description (optional)
Remind
Cancel
Save

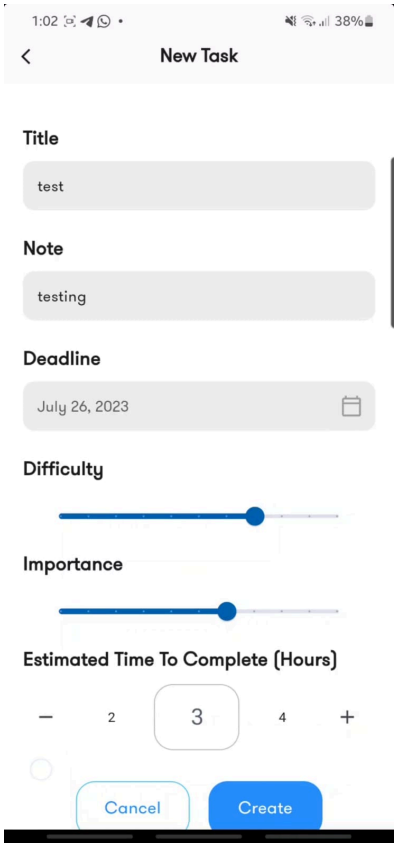

Calendar
July 2023
TUE
18
test
hvvb
ggcg
Day
Week
Month
Timeline Day
Schedule
SAT
22
test
10:00 AM
Lunch meeting
10:00 AM
12:00 PM
Jul 23 - 29
Jul 30 - Aug 05
August 2023
Aug 06 - 12
Aug 13 - 19
Aug 20 - 26
Aug 27 - Sep 02
+ Add Event
September 2023

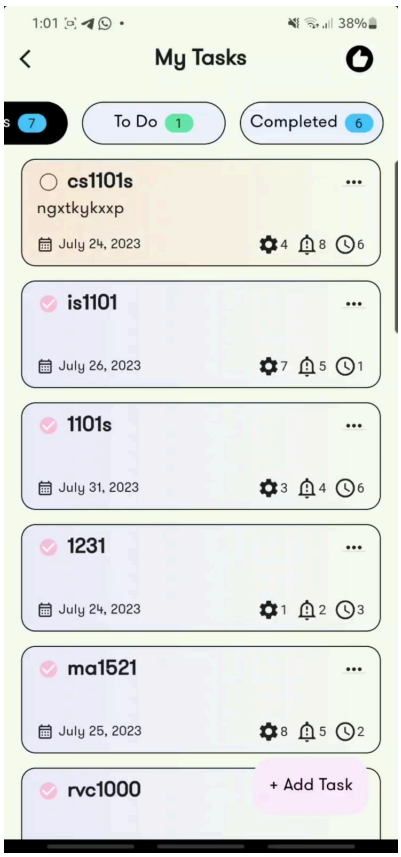

Calendar
July 2023
SAT
22
Day
Week
Month
Timeline Day
Schedule
test
Lunch meeting
+ Add Event

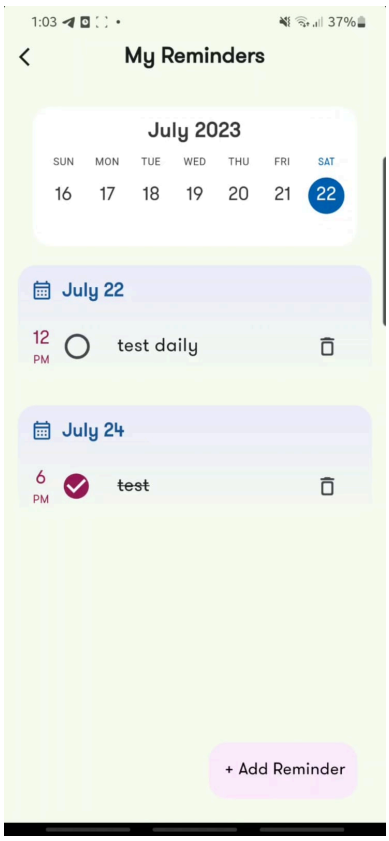

Calendar
July 2023
S M T W T F S
16 17 18 19 20 21 22
hchcj
test
ggfiu
ejjdj...
+ Add Event

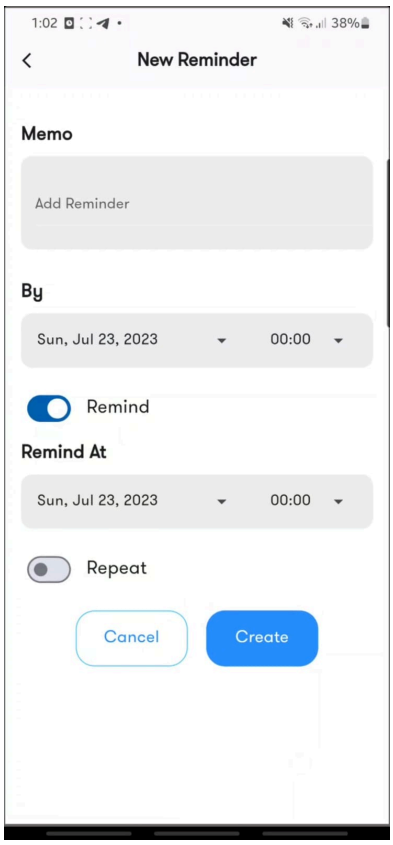

My Tasks
To Do 8
Completed 0
cs1101s
ngxtkykxxp
July 24, 2023
test
testing
July 26, 2023
Edit
Delete
is1101
July 26, 2023
1101s
July 31, 2023
1231
July 24, 2023
ma1521
+ Add Task

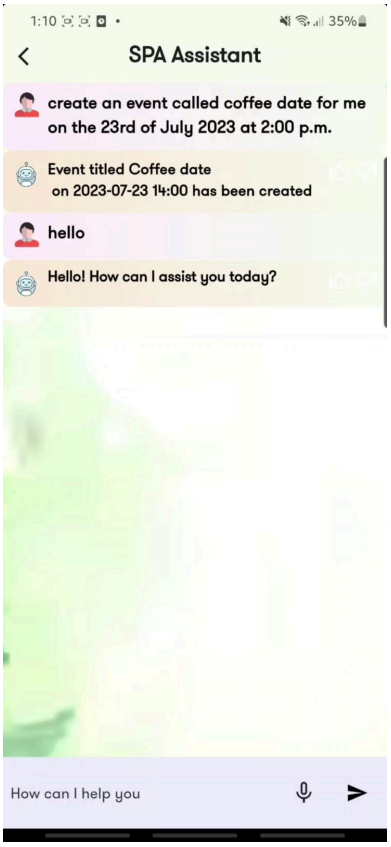

My Tasks
To Do 7
Completed 0
Suggested Order of Completion
1. 1231
2. cs1101s
3. ma1521
4. is1101
5. rvc1000
Got It!
ma1521
July 25, 2023
rvc1000
+ Add Task


New Task
Title
test
Note
testing
Deadline
July 26, 2023
Difficulty
Importance
Estimated Time To Complete (Hours)
2
3
4
Cancel
Create

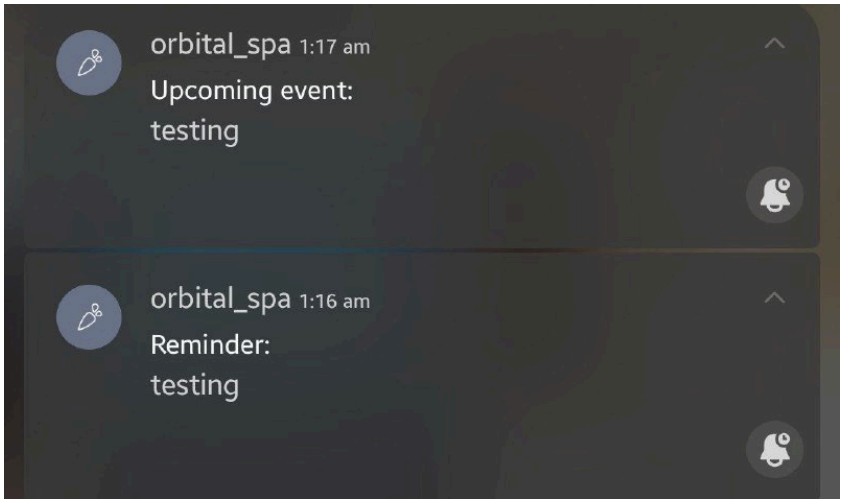

My Tasks
To Do 1
Completed 6
cs1101s
ngxtkykxxp
July 24, 2023
is1101
July 26, 2023
1101s
July 31, 2023
1231
July 24, 2023
ma1521
July 25, 2023
rvc1000
+ Add Task


My Reminders
July 2023
SUN MON TUE WED THU FRI SAT
16 17 18 19 20 21 22
July 22
12 PM test daily
July 24
6 PM test
+ Add Reminder

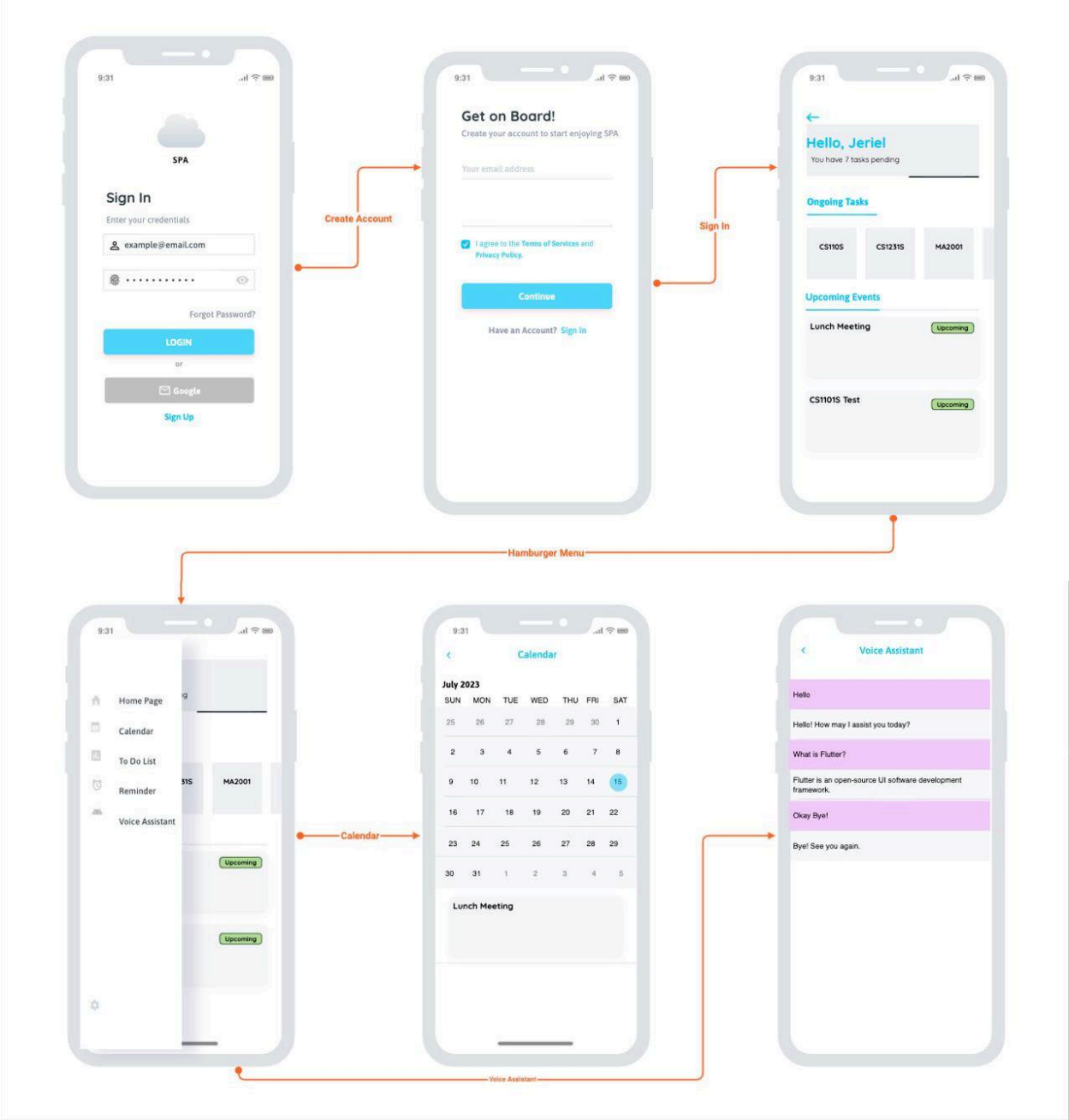

New Reminder
Memo
Add Reminder
By
Sun, Jul 23, 2023
00:00
Remind
Remind At
Sun, Jul 23, 2023
00:00
Repeat
Cancel
Create


SPA Assistant
create an event called coffee date for me on the 23rd of July 2023 at 2:00 p.m.
Event titled Coffee date on 2023-07-23 14:00 has been created
hello
Hello! How can I assist you today?
How can I help you


SPA Assistant
what is flutter
Flutter is an open-source UI software development kit created by Google for building mobile applications.
How can I help you


orbital_spa 1:17 am
Upcoming event:
testing
orbital_spa 1:16 am
Reminder:
testing

Updated Wireframing Mockup



Development Timeline

By Milestone 1: Complete Setup

1) Frontend Routing
2) Database Setup

By Milestone 2: Complete Core Features

1) To-do List
2) Calendar
3) Reminders (Notifications)
4) Voice Assistant

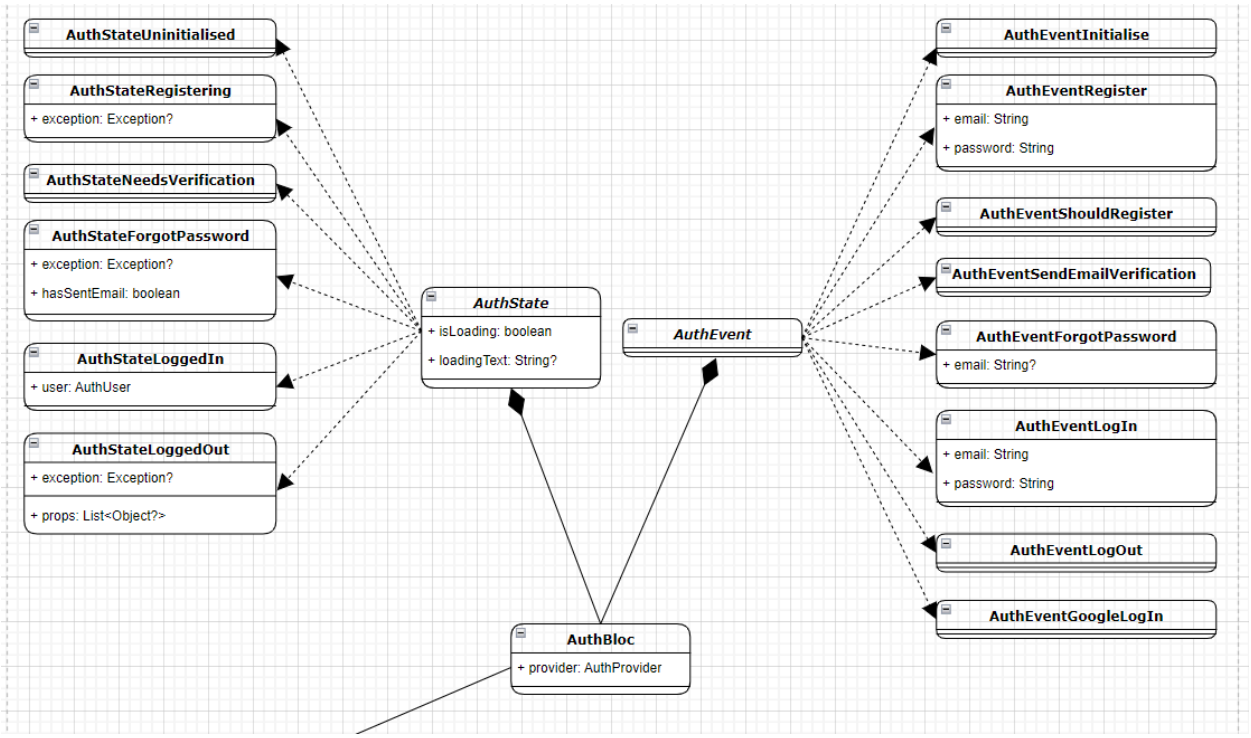
By Milestone 3: Complete User Interface and Fix Core Features

1) User Interface for the main application and features
2) Finalize core features that may not be fully completed
3) Fix any bugs that may be present in the core features implemented
4) Extension: Possible extensions include profile pages, customisation of the application and a chatbot powered by ChatGPT
5) Add task suggestion feature
6) Testing

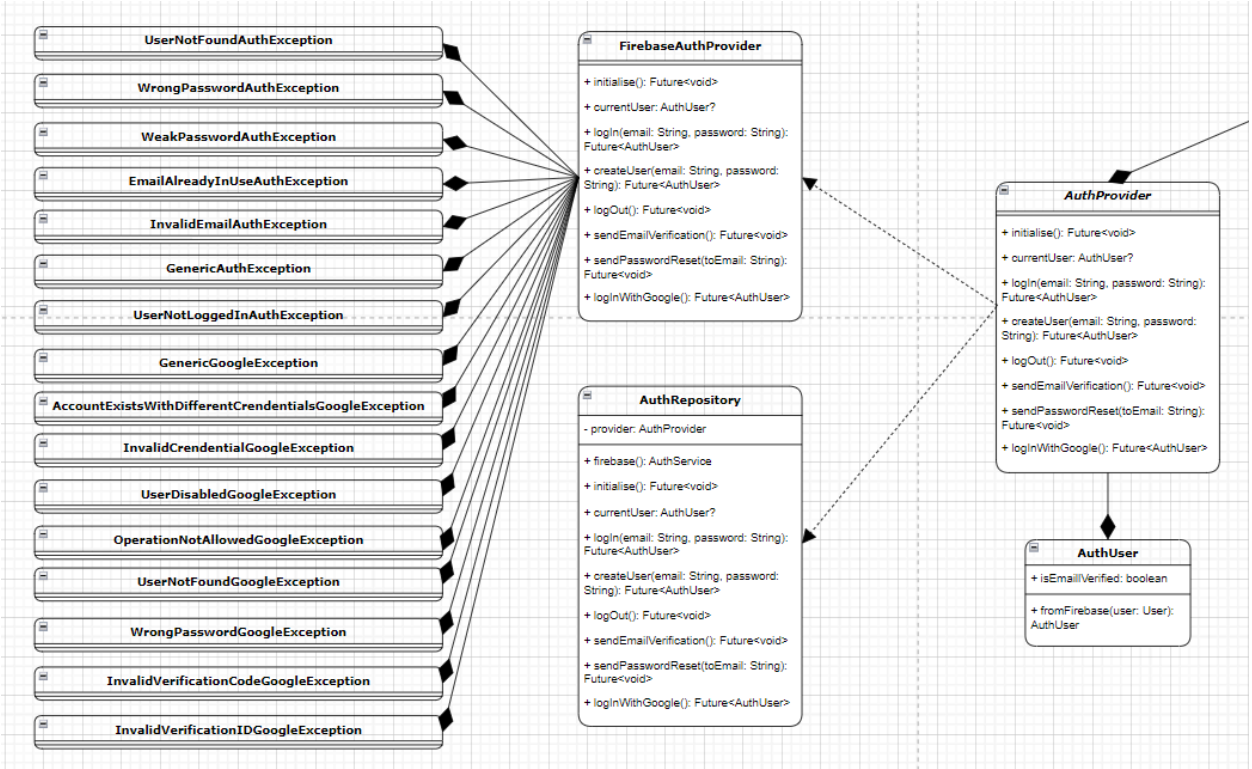
Unified Modelling Language (UML) Diagram

We created UML Diagrams to provide a better understanding on how our mobile application is structured. View the full UML Diagram: https://drive.google.com/drive/folders/12nyZTON7BUj2OAyQkmwrUOLyOpxGJE2e?usp=drive_link

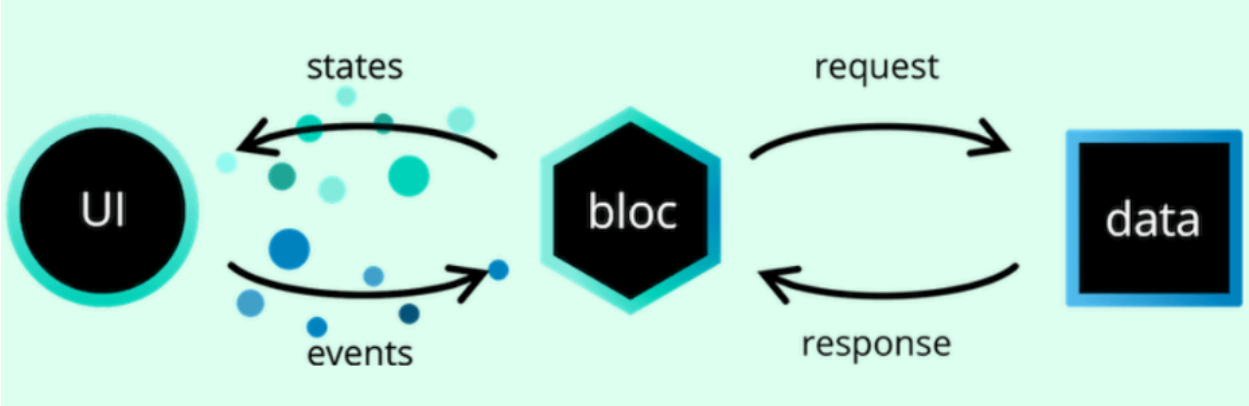
1) **BLoC Management**



BLoC uses events and states to manage the application by taking in a stream of events.



From the stream of events received by BLoC, it makes a request to the provider, which provides the data required back to BLoC. BLoC then emits a stream of states as output back to the User Interface.

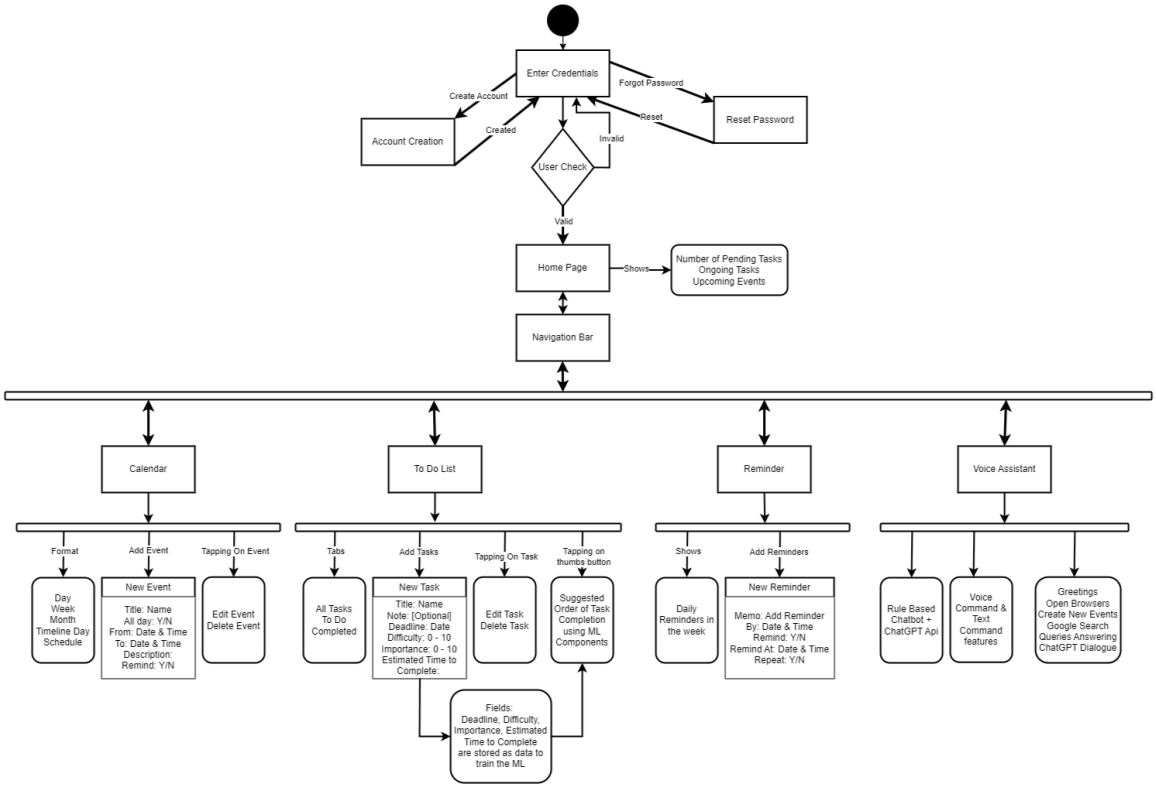


2) Application Flow

Using the state management library Bloc, the authentication process is controlled through the Business Logic Components design patterns. This separates the business logic from the User Interface. HomePage listens for the states emitted and pushes the respective authentication pages to authenticate the user.

The main page is used to navigate to the various features such as the calendar, to-do list, reminder list and voice assistant. Instead of BLoC, GetX is used to manage the state of the application. Provider, another state management class, is also used for the voice assistant to update widgets.

The application flow is shown in the activity diagram below:



Limitations

1) ChatGPT

- Rate Limit: OpenAI may impose a rate limit on the usage of ChatGPT to manage server load and ensure fair usage for all users. This means that if you send a large number of requests within a short period, you might encounter delays or restrictions in accessing the service.
- Varying Response: ChatGPT generates responses based on patterns it has learned from its training data. While it can often provide accurate and helpful information, it may sometimes produce varying responses to the same query, leading to potential inconsistencies.
- Unreliable Event Creation: When asked to create events based on keywords like 'today, tomorrow, next week,' the model might not reliably generate specific event details due to its inability to access real-time

information. Hence, to create an event with a specific date using ChatGPT, you need to specify the date in the format "dd/mm/yyyy." If the user doesn't follow this format, the model might not correctly interpret the date and could lead to inaccurate responses.

- Date Issues: For some reason, the current year produced by the API is 2022.

Project Log

The full project log can be found at
https://docs.google.com/spreadsheets/d/1w_gvhXAnq7-gayPEXQykMVZGMjC372eGZgcxFUja1Z0/edit?usp=sharing

APK Release File

The application is only suitable for Android currently
Download the APK file here to test the application:
https://drive.google.com/drive/folders/1TiZmBrQ2Pi4nx1tBLJsDbchIwAGeZlN6?usp=drive_link

View the demo video here:
https://drive.google.com/file/d/1bUP0FhdOdVgLDfax6BodzkRDaj12Pgnu/view?usp=sharing

View the full poster here:
https://drive.google.com/file/d/1xzG1cf4p-ncjK8VRggbeXlm2KlafjbcY/view?usp=sharing