README - SyncUp

SyncUp

Milestone 3

Lim Jian Yang

Timothy Leow

Table of Contents

Table of Contents
Foreword / Preamble
Poster
Proof-Of-Concept/Deployment
    Additional Installation Instructions
Proposed Level of Achievement
Aim
Motivation

User Stories
    Extensions:
Features
    User Account Authentication / Onboarding
    Home Page
    Personal Calendar Page
    Groups Page
        Managing Groups
        Group Chat Feature
        Scheduling Meetings
    Account Page
Overall Navigation Flow
User Interface Design
Timeline and Development Plan
Entity Relationship Diagram (ERD)
Software Engineering Practices
    Architecture Pattern:
    Version Control:
        Branching:

        Pull Requests:
        Git Issues:
    Security Measures:
Quality Control
    Automated Testing
        Unit Testing
    User Testing
Tech Stack
Project Log

Foreword / Preamble

Since we started university, finding a common time to collaborate has become a relevant problem more than ever before. As such, we took Orbital 2023 as an opportunity to find and implement an optimal solution to the problem of dealing with the hectic schedules of multiple people. Manually filling up availabilities is time-consuming and we wanted to improve on existing solutions like when2meet. We brainstormed hard and `SyncUp` was the result.

`SyncUp` is a project management application that encourages a smoother collaborative experience - through seamless calendar API integration, it automatically uses event/availability data to suggest the best meeting times within a group. This is made possible through Flutter's calendar_v3 library, as well as other existing frameworks that we have made use of.

Poster

OBJECTIVE

Our app aims to transform the way collaboration and scheduling is done. Through seamless calendar API integration, SyncUp minimizes the steps needed to find a common meeting time within a group of people.

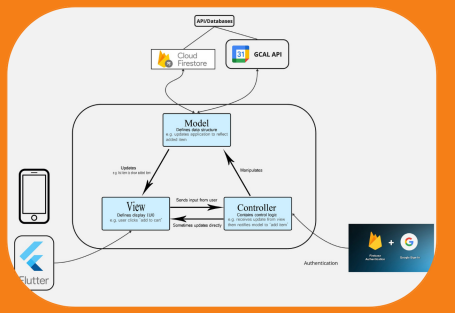
FEATURES

- Google Calendar Integration
- Group Event Calendar
- Seamless group collaboration
- Academic Database
- Group Chat

SOFTWARE ENGINEERING PRACTICES

SyncUp adheres to the MVC (Model-View-Controller) software design pattern - the code handling business logic and visual are systematically separated.

Git was used appropriately to handle version control, with branching, pull requests, and issues to trace down and bugs.



TECH STACK:

SyncUp

Collaborate and schedule like never before.
You're never filling up a when2meet ever again.

Hassle-Free Sign-In with Google Account

Automatic Availabillity-Syncing with Google Calendar

Smooth collaborative experience with creating, joining and browsing groups

Tried and tested foolproof algorithm to suggest meeting timeslots

Seamless API integration Events created added directly to gCal with invitation emails

View and create events directly from personal calendar page (gCal)

Academic database - share, view and rate cheatsheets

Customise group settings and meeting hours according to purpose

Converse with group members using the group chat feature

Proof-Of-Concept/Deployment

Refer to the video demonstration here.

Download the APK here, or try out our application on TestFlight if you are using an iOS device.

Additional Installation Instructions

▼ For Android

1. Download the SyncUp APK file from here.
2. Transfer the APK file to your Android device using any preferred method, such as email, USB cable, or cloud storage.
3. On your Android device, go to **Settings**.
4. Navigate to **Security** or **Privacy** (the exact location may vary depending on your device).
5. Enable the **Unknown Sources** option. This allows the installation of apps from sources other than the Google Play Store.
6. Use a file manager app to locate the transferred SyncUp APK file on your device.
7. Tap on the APK file to start the installation process.
8. Review the permissions required by the app and tap **Install** to proceed with the installation.
9. Wait for the installation to complete.
10. Once the installation is finished, you can open SyncUp from your app drawer and start using it to manage your schedules and collaborate with others effectively. Follow along with our walkthrough/video below!

▼ For iOS/iPadOS

1. Install TestFlight on the AppStore if you don't already have the application.
2. Accept SyncUp and help us test it through this link.

Proposed Level of Achievement

Apollo 11

Aim

Our project aims to transform the way scheduling is done, making it a breeze for everyone involved. Our platform will also allow students from different faculties to have the opportunity to connect and collaborate on exciting projects together.

Motivation

As university students, we understand the importance of collaboration when it comes to achieving academic success. However, the process of coordinating schedules with peers can be a major obstacle that often leads

to frustration and wasted time. **The traditional method of coordinating schedules through multiple exchanges of texts and emails is not only time-consuming but also prone to errors and misunderstandings.**

Furthermore, the current situation of remote and hybrid learning has only added to the complexity of syncing up schedules. With students located in different time zones due to overseas exchanges, juggling different class schedules and extracurricular activities, finding a common meeting time has become an even greater challenge.

That's why our app is here to help. By providing a centralized platform for students to schedule and manage their meetings, the app streamlines the coordination process, reducing the time and effort required to find a suitable meeting time. With features like real-time availability updates and schedule-conflict management, students can quickly and easily find a time that works for everyone. Our app not only saves time but also promotes better communication among students, making the collaborative process hassle-free, be it for casual study groups, project groups, or even CCA interest groups.

User Stories

1. As a student, I want to be able to form specialised groups with their friends, group project members, or interest group committee members.
2. As an individual with a hectic schedule, I want to be able to to easily identify mutual availability with other individuals of interest and schedule meet-ups easily.
3. As an inquisitive and sociable student, I want to be able to effortlessly network with others from different faculties and form groups on the platform.
4. As a student looking for others with common interests, I want groups to indicate their intention of recruiting members of certain interests, making it easy for me to connect with like-minded individuals.
5. As a user with multiple groups, I want to be able to know when a certain group is having a meeting and how it lines up with my personal schedule.

Extensions:

6. As a student, I want to be able to access/share cheatsheets or course summaries made by others and know which are the best ones to use.
7. As a long-term user, I want to be able to keep track of the correlation between the meetings had by the group and how good the output is seen to be.

Features

User Account Authentication / Onboarding

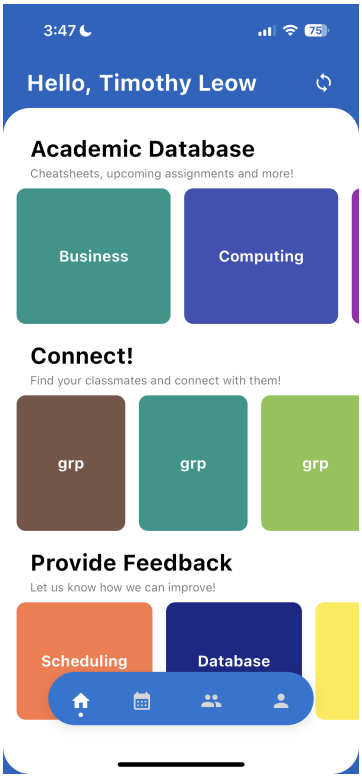
▼ Description

As we are mainly working with google sign-in / google calendar API, we have made the decision to reduce the sign-in methods to only via Google Accounts for now. The considerations made are simple:

- We had previously allowed sign-ins with any emails, but we decided to focus on google calendar API to make things more streamlined.
- The application will only work with google accounts as we have not implemented calendar API integration with other services (e.g., outlook calendar, apple calendar).
- This simplifies the sign-in process as the authentication is done through the google-sign-in API, and reduces the need to manage accounts.

The google-sign-in package in flutter still works with Firebase authentication, so a successful login will result in the account being reflected in the list of accounts in our Firebase console.

Home Page



▼ Description

The HomePage exists mainly for quick access to the other main features, such as Academic Database, where users can view and rate cheatsheets, “Connect”, where users can quickly request to join groups they are interested in, and “Provide Feedback”, where users can send their feedback directly to us developers regarding any of the features we have implemented.

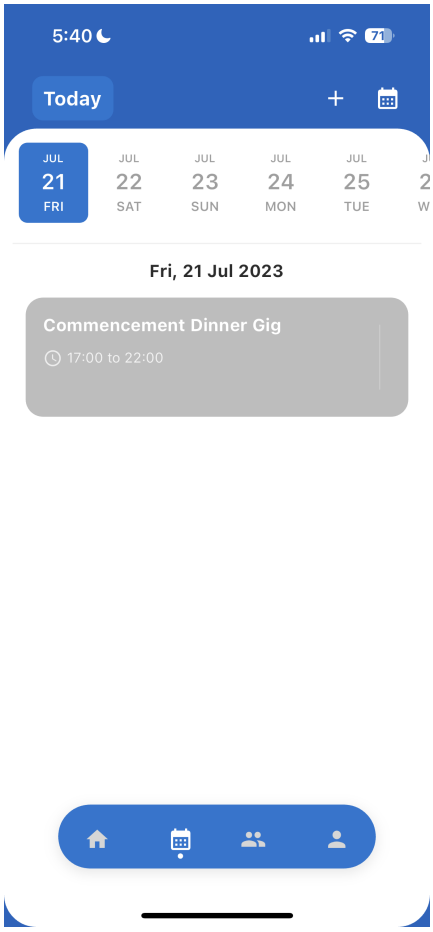
▼ Implementation Philosophy

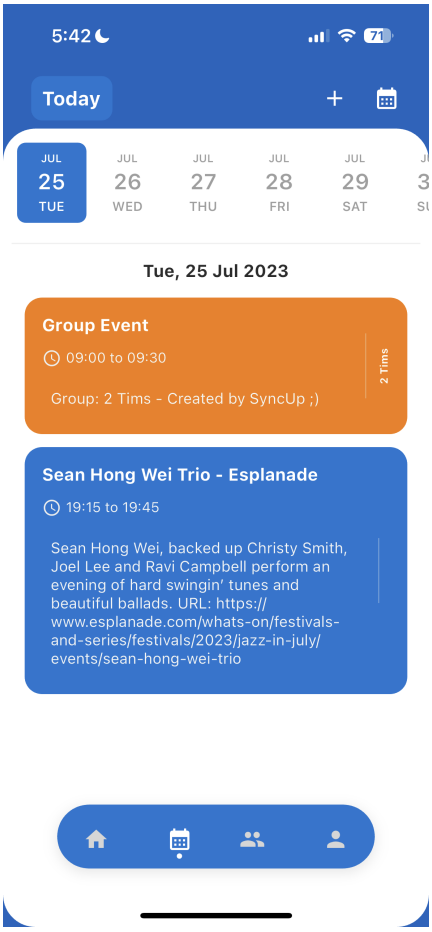
While there is a navigation bar that allows users to easily access the other essential features, we thought that the Home Page would be a good way to welcome the user and provide an additional option of quick access.

▼ Implementation Challenges

The challenges of this page lie mostly in the design aspect - we have chosen on an infinite horizontal scrolling section for each segment of the homepage as it is a clean option. However, this means that as the number of groups/categories of database scale to a very large number, our interface becomes limited - in future, we are considering implementing a search filter for each segment to avoid excessive scrolling.

Personal Calendar Page

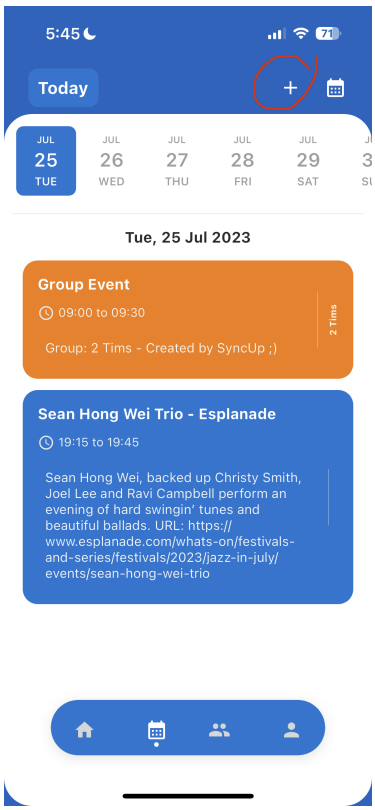

5:40
Today
JUL 21 FRI
JUL 22 SAT
JUL 23 SUN
JUL 24 MON
JUL 25 TUE
Fri, 21 Jul 2023
Commencement Dinner Gig
17:00 to 22:00

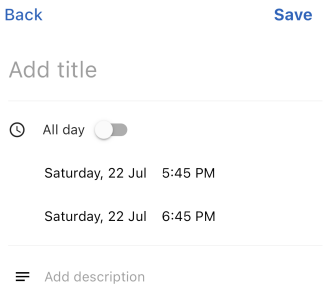

5:42
Today
JUL 25 TUE
JUL 26 WED
JUL 27 THU
JUL 28 FRI
JUL 29 SAT
Tue, 25 Jul 2023
Group Event
09:00 to 09:30
Group: 2 Tims - Created by SyncUp ;)
2 Tims
Sean Hong Wei Trio - Esplanade
19:15 to 19:45
Sean Hong Wei, backed up Christy Smith, Joel Lee and Ravi Campbell perform an evening of hard swingin' tunes and beautiful ballads. URL: https://www.esplanade.com/whats-on/festivals-and-series/festivals/2023/jazz-in-july/events/sean-hong-wei-trio

- Description

  The Personal Calendar page pulls event data through the google calendar API and formats them accordingly based on the event data - events created using `SyncUp` will be in bright orange whereas personal events will be in blue. A new UI enhancement is that events that are in the past are greyed out.

  It could arguably be a substitute for other calendar apps, since there is the option to create events which are added directly to the user's google calendar by clicking on the "+" at the top right corner.





▼ Implementation Philosophy

We wanted users to be able to cross-check group events against personal events within our app, and so we decided to create this page to make it easy for them to check their calendar.

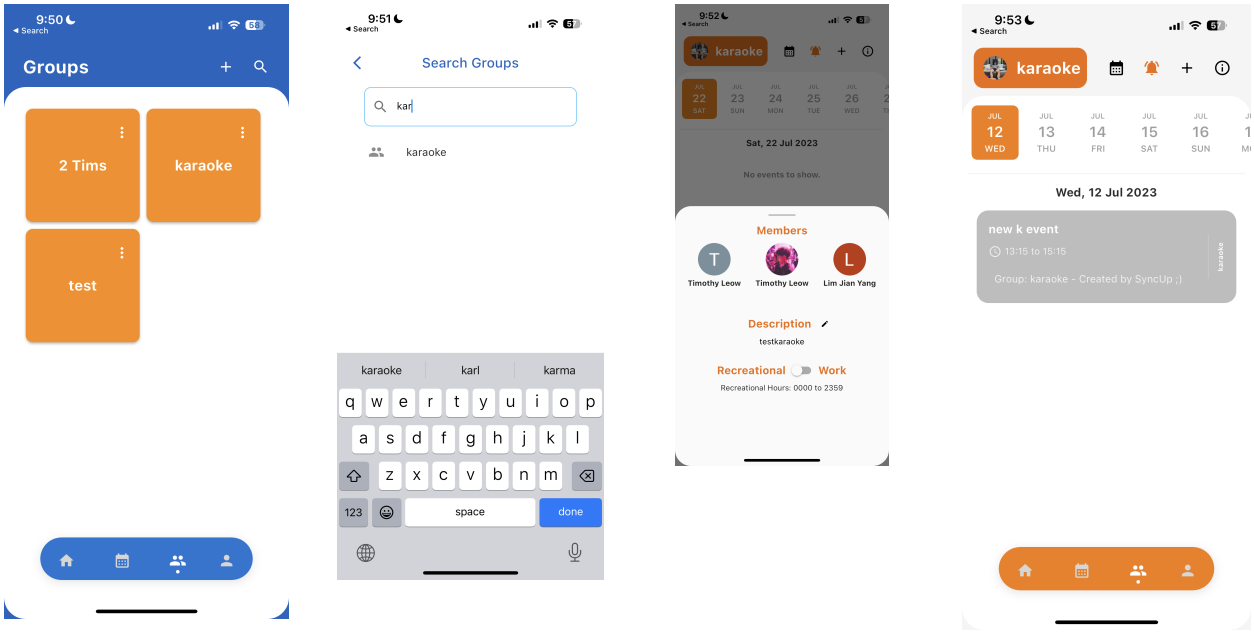
From there, we expanded on the features by allowing users to create and add their own events to their google calendar too.

▼ Implementation Challenges

The main challenge was to correctly query the data from google calendar API - we had to make sure that the code was universal, and worked across different timezones such that this page was able to accurately display and manipulate calendar data using appropriate API calls.

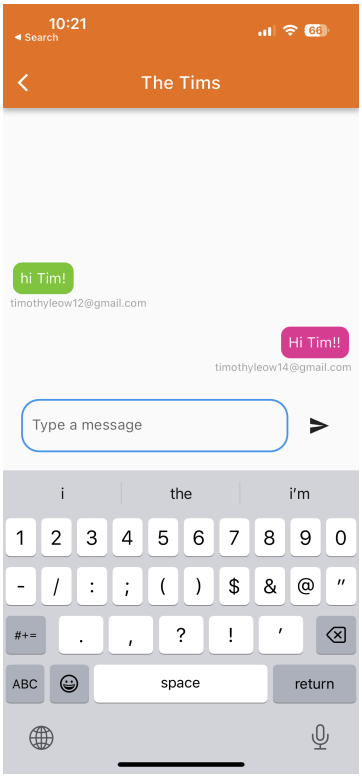
Groups Page

Managing Groups



- We implemented a “Groups” infrastructure such that users can:
  - Create groups and add users to them
  - Request to join other groups
  - Leave a group
  - Manage the members in the group
  - Create/edit the group description
  - Specify the type of group (Recreational or Work) so that the meeting time window is appropriate
  - Create and see group events based on the availabilities of the members in the group

Group Chat Feature

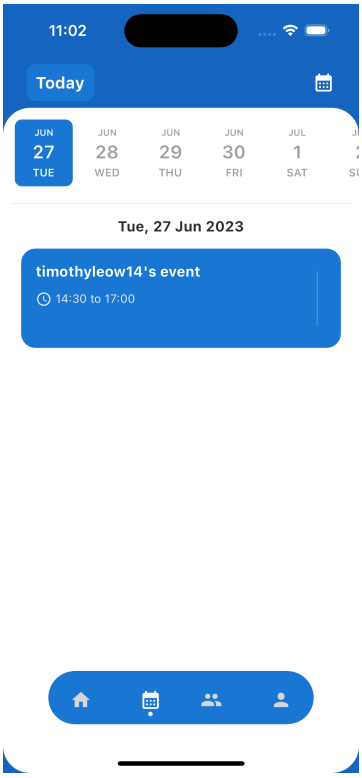


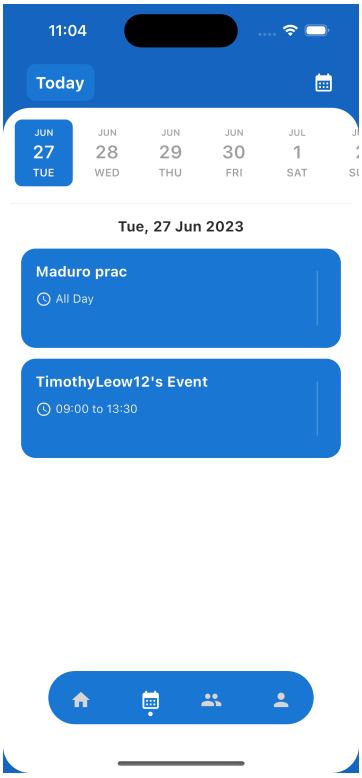
Members of a group can communicate with each other using the handy group chat feature too.

Scheduling Meetings

Description:

To explain this feature, we will use an example. The 2 accounts in the group are timothyleow12.com and timothyleow14@gmail.com, and we are trying to schedule a meeting between them on 27 June 2023, within working hours (0900-1700).

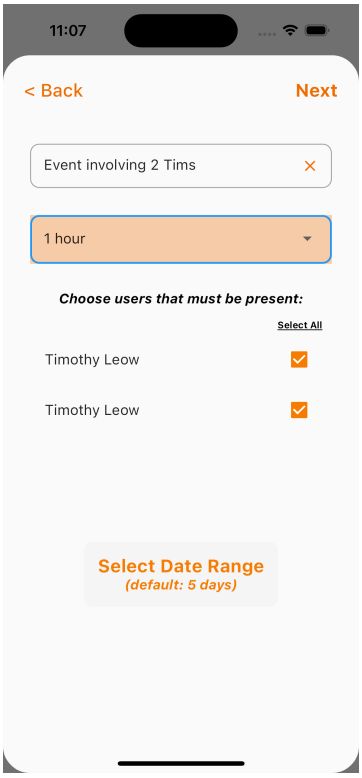


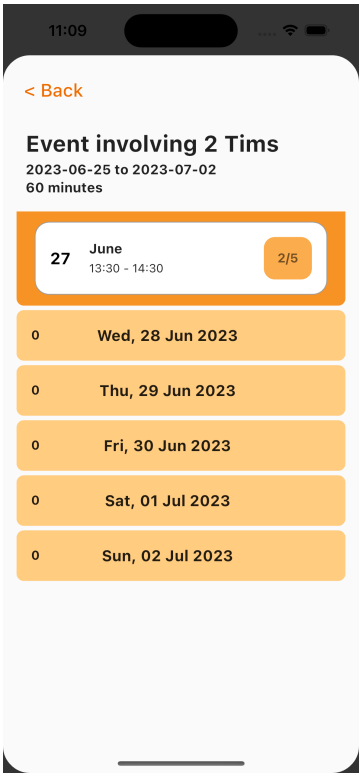


timothyleow14 is occupied from 1430-1700

timothyleow12 is occupied from 0900-1330

Hence, the only free timing between them from 0900-1700 is 1330-1430. Our algorithm will suggest this timing for them to meet for one hour as seen in the screenshots below:





The user gives a few general details about the event such as name, duration, and selects a desired date range for the event to be held in.

The app performs the “Free-Slot-Finding” algorithm, by comparing it against the availabilities of the group members (which are automatically retrieved and synced with the members’ google calendars). It then lists out all the possible slots, along with the attendance rates listed at the side. In this case, the only available meeting timeslot between these 2 people is 27th June 1330-1430.

After a meeting is scheduled, invitation emails are sent out on gmail, and the event should show up in the participants’ google calendar (they can accept or decline the invitation from there).

▼ Implementation Philosophy

In apps like when2meet, users fill out their availabilities manually on a calendar grid, which is time-consuming. SyncUp automates this process (provided the user’s google calendar is updated), and handles the availabilities of group members in a blackbox fashion, only suggesting to the users the meeting times.

For recreational groups, the meeting hours should generally be more flexible than for working/official groups. We wanted to make it as hassle-free as possible for users to schedule an events between their desired group participants. As such, we minimised the number of steps needed to schedule an event, only needing the duration, name, and desired users.

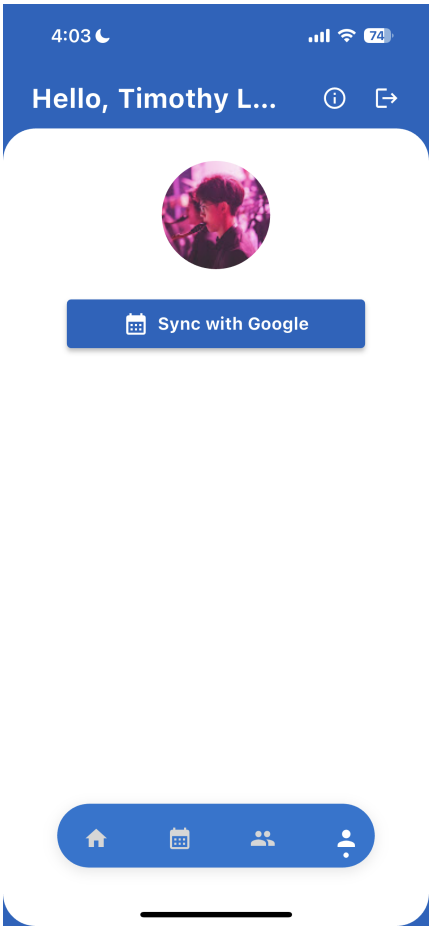
▼ Implementation Challenges

An easier solution to implement this feature was to store all google event data in the database - however, Firestore does not allow this as the documents in the collections are restricted to certain data types. As such, we had to process the event data from google calendar API and parse them as strings to be stored in

the Firestore database, so that we could still perform the algorithm of finding free timings between members.

This made the design of our code less than optimal; we had to covert DateTime objects into strings that represented the length of time occupied (e.g., “22-07-2023 1000-1100”) and vice versa - we had to process the strings to give sensible DateTime suggestions when the algorithm demanded for it. A possible solution might be to use an SQL database - which might break up the already carefully planned integration in our app between Flutter and Firebase. Hence, we decided to go ahead with this compromise as the algorithm still works as can be seen in our unit testing (see below).

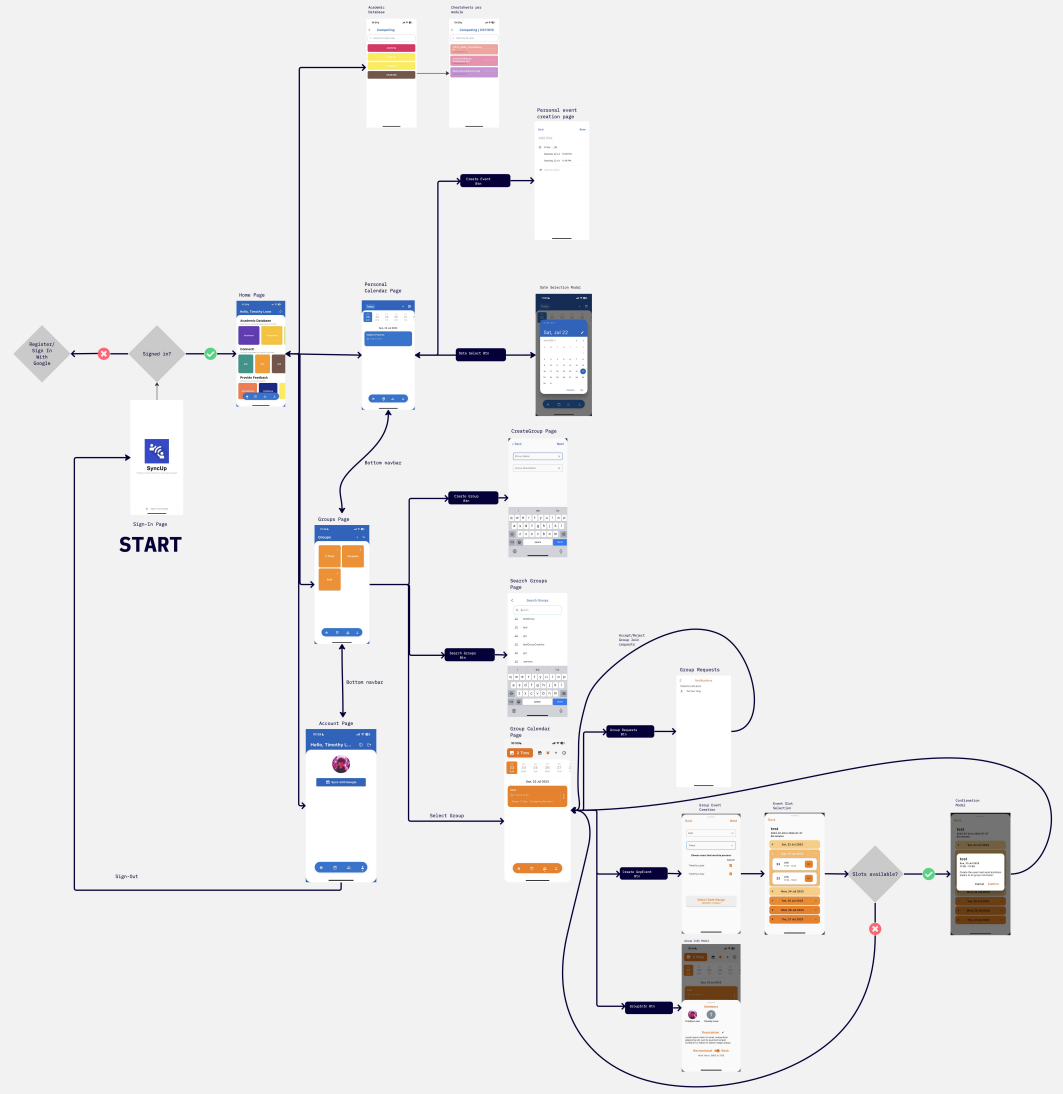
Account Page



▼ Description

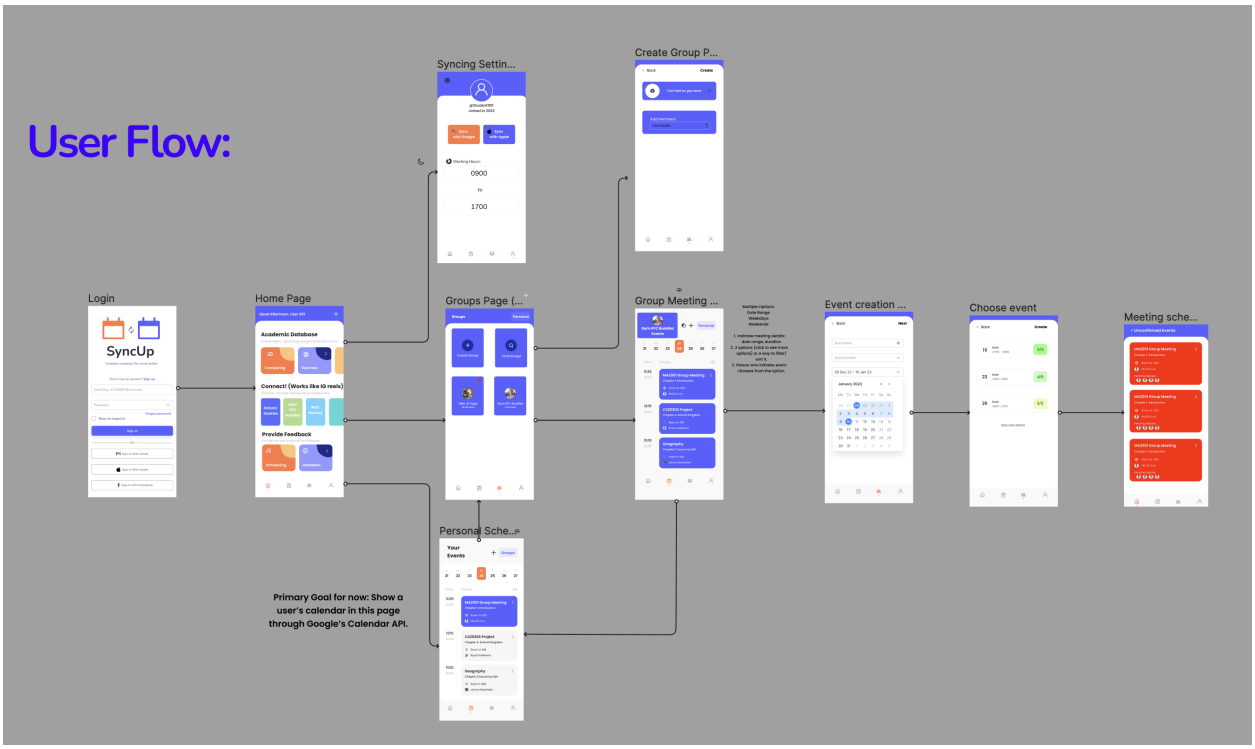
The main purpose that this page serves is to allow users to trigger the syncing of their google calendar to update their availability accordingly. It also allows the user to sign out and log-in with a different account, and indicates clearly to the user which account is currently logged in, with a greeting message and a profile picture taken directly from the user’s google account.

Overall Navigation Flow



View the clearer, full miro board here.

User Interface Design



Above are our original Sketches of UI Design on Figma.

Timeline and Development Plan

S/N	**Tasks**	Description	Intended Execution Period	I/C	Completion Status
1	Acquiring knowledge and technical skill set	Pick up technical skills for flutter, firebase, git and google api integration.	12-28 May	Tim, JianYang	Completed.
2	Develop Barebones Application, implementing register/login before Milestone 1	Integrate firebase authentication with a skeleton flutter app.	20-29 May	Tim, JianYang	Completed.
3	Focus on UI design and generation	Learn to create reusable components and compartmentalise code where appropriate.	30 May - 12 June	JianYang	Completed
4	Integrate UI with google calendar API integration, and connect to backend	Find out the correct flutter packages to use to make the front-end and back-end integration seamless	12 - 28 June	Tim, Jianyang	Completed

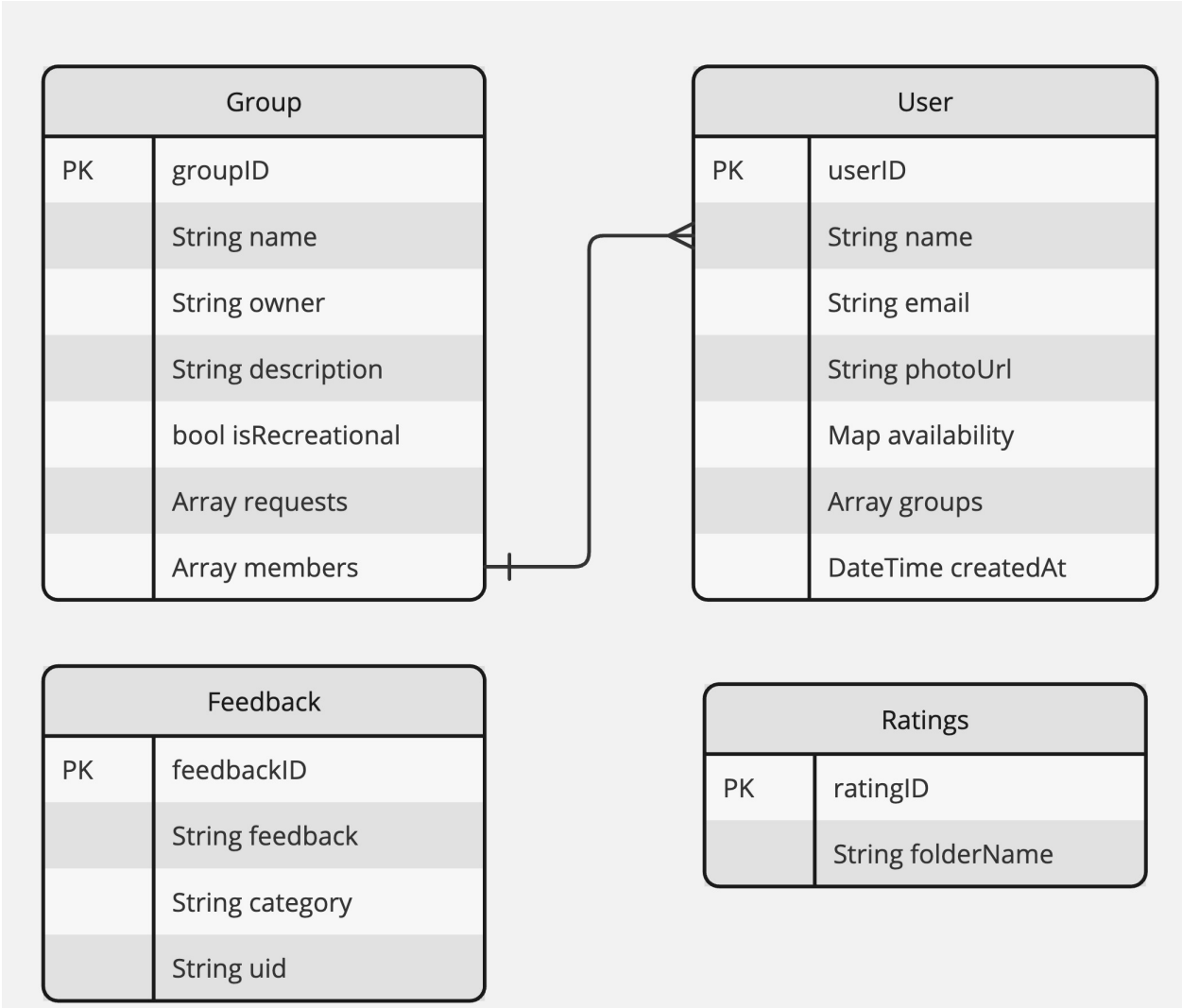
	with Firestore database				
5	Focus on optimising Groups and Scheduling Feature	-	28 June - 14 July	Jianyang, Tim	Completed
6	Implement Additional features such as Academic Database, and feedback system	-	14 July - 24 July	JianYang	Completed
7	Allow integration with other calendar API as well (e.g., apple calendar, microsoft exchange)	-	14 July - 24	Tim	Incomplete due to lack of time.

Entity Relationship Diagram (ERD)

We used Firestore, which is a NoSQL database and is limited in the types of data that we can store. As we had to model many-to-one and one-to-many relationships, we took strongly to the approach of modelling our Firestore database as a relational database.

This means that we were storing references to documents between collections using document IDs, to avoid unnecessarily storing object copies.

Below is the ERD to describe the relationships within our Firestore database.



By querying events on demand from google calendar on demand, we are able to save space by not storing the full event data. This also makes sense because it would be too inefficient to have our data matched to the actual event google event data at all times. As such, we do not have to worry about event data in the ERD. Instead, we use array and string manipulation to store the availability of the members, which is automatically updated when they restart the application, or trigger the syncing calendar function.

Software Engineering Practices

Architecture Pattern:

💡 Note: The examples below are from the file `lib/components/common_slots_tile.dart`. It is where the callback for group event creation is handled - which is the core feature of our application.

Model:
In SyncUp, models are classes that represent the data and logic needed to render specific screens and components. For instance, the `CommonSlotsTile` model encapsulates essential information about a group event, such as the event name, selected period, start date, end date, user IDs, and member count. This model not only holds data but also serves as an intermediary for data conversion between the application and Firestore.

Within each model class, there are methods like `getSlotsSuggestionsHelper`, `getStartEndTime`, and `getUserEmails` that handle data manipulation, formatting, and retrieval from Firestore. These methods ensure that the models used by the application are independent of the database responses and provide a convenient way to access and manipulate the data throughout the app.

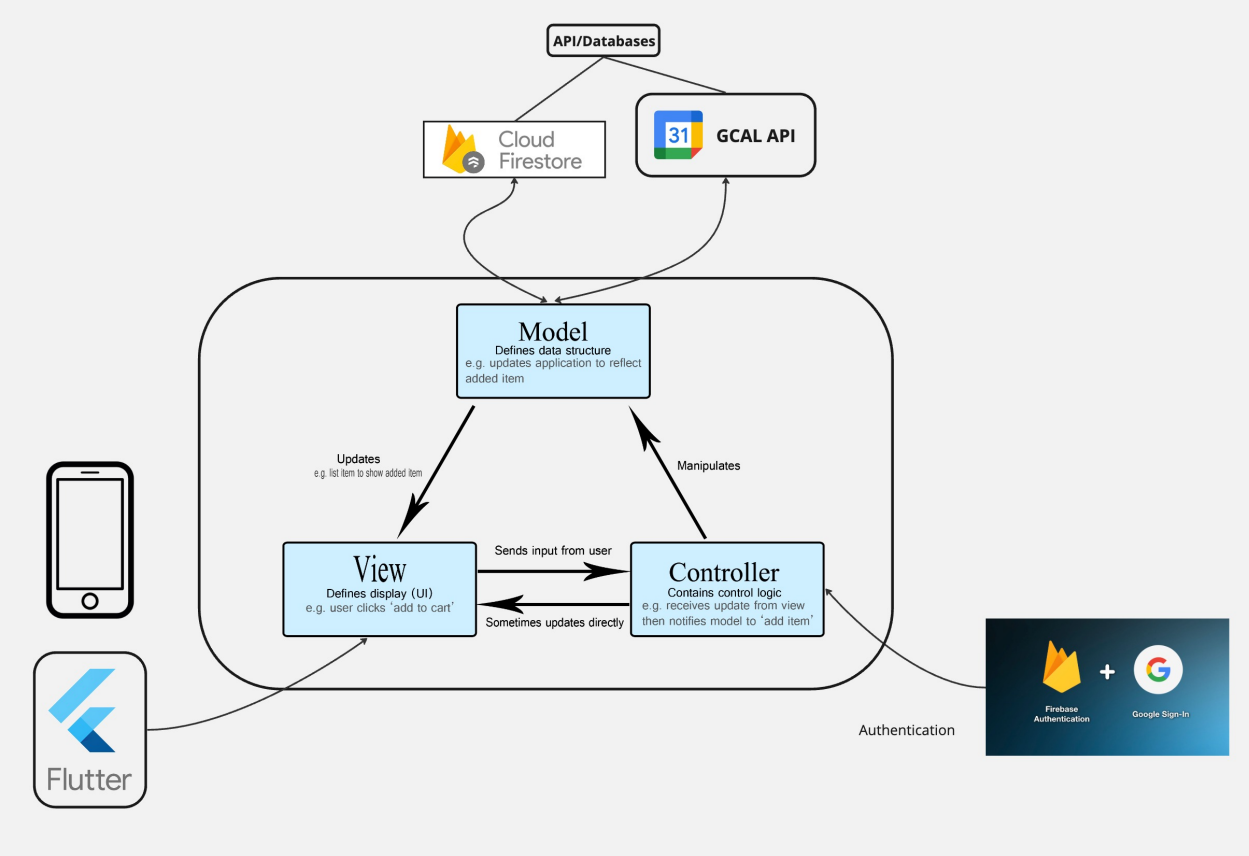
View (Flutter Widgets):
In SyncUp, views are represented by Flutter widgets responsible for displaying the user interface. The `CommonSlotsTile` widget is an example of a view that presents information about the group event time slots on the screen. It utilizes various Flutter widgets such as `Text`, `ListView`, `ExpansionTile`, and others to construct the user interface and show relevant data to the users.

Controller (SyncUp Controllers):
Controllers in SyncUp act as the bridge between the Models and Views, handling business logic and managing interactions between the user interface and data models. In `lib/components/common_slots_tile.dart`, the `_CommonSlotsTileState` class serves as a controller for the `CommonSlotsTile` view. It contains state variables, like `selectedDate` and `now`, which are directly used within the view and might change during the app's runtime.

The controller is responsible for processing user interactions, such as creating and inserting events into the Google Calendar, and updating the view accordingly. For instance, the `createEventAndBackToGroupPage()` method handles the logic to create a calendar event and then navigates back to the group events page.

Furthermore, the controller can communicate updates to the view by calling `setState()`, which triggers a widget rebuild to reflect the updated data.

Overall, the SyncUp application adheres to the MVC pattern by effectively segregating business logic and data handling (Models), user interface rendering (Views), and interaction management (Controllers). Embracing this design pattern helps maintain a clear and organised structure for the codebase, enhancing the application's maintainability and scalability as it grows in complexity.



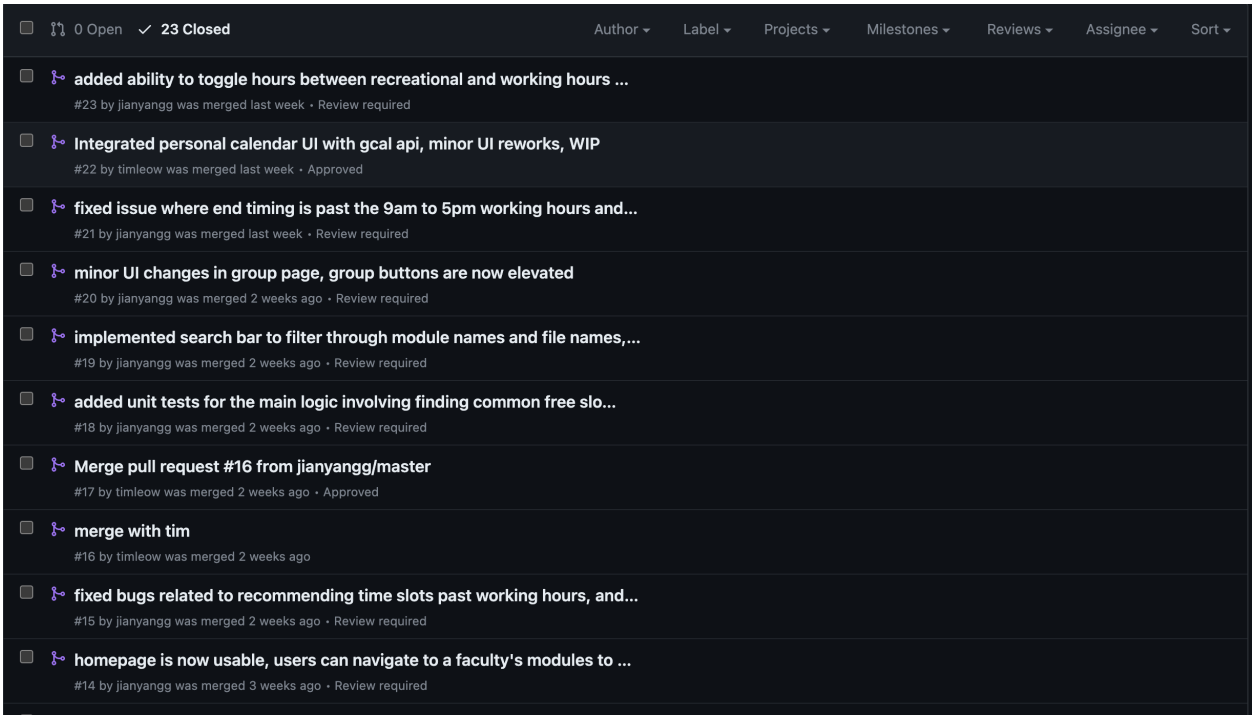
Version Control:

Branching:

We used git for version control and collaboration on our code. It was always made sure that the code on the master branch was deployable and working at all times. To experiment, expand or work on new features, we branched off into our individual branches. As we worked on multiple features at once, we did not see the need

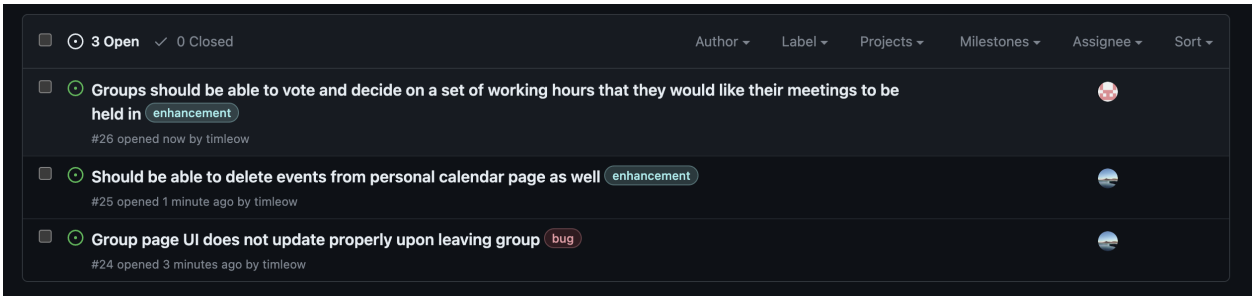
to create one branch for one new feature, and each of us mainly worked in our individual branches *(Timothy worked in tim_branch, while Jian Yang worked in jy_branch).*

Pull Requests:



After we completed and tested the features created in our individual branch, we mainly made use of pull requests to merge the code with the master branch. We performed code reviews on each other's code and handled merge conflicts, so that we were sure that the application would still work, before merging the branches with the master branch.

Git Issues:



With git issues, we made use of the assign and tag function to keep track of any bugs or potential enhancements within our project. We would close the issue when the solution has been implemented.

Security Measures:

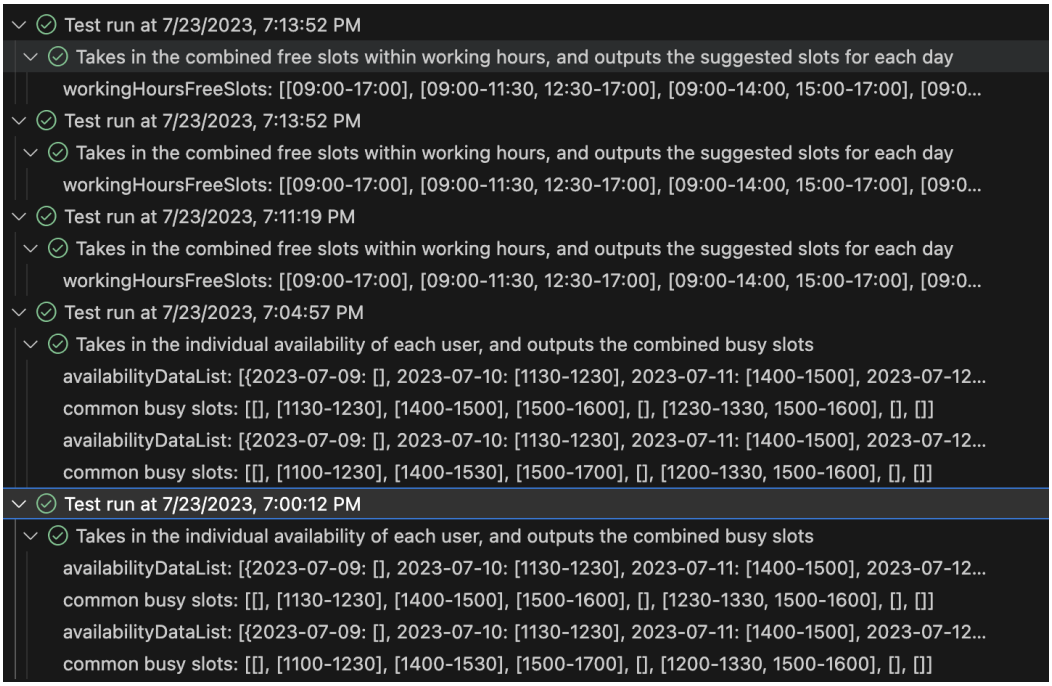
API, SHA keys, tokens were left out of the files and never hardcoded directly and committed to GitHub, preventing any unwanted authentication requests/changes to our firestore database

Quality Control

Automated Testing

Unit Testing

The main feature in our app that required a high level of correctness is the “Free-Slot-Finding Algorithm” which finds all the possible meeting timings based on the availabilities of group members, within a specified set of hours (e.g., working hours 0900-1700, so the time-slots suggested will be within this time). As a result, we decided to test the functions that compose this algorithm.



Above is a screenshot of our application passing the unit tests. The summary of the test cases can be seen below - this gave us the assurance that the app was suggesting sensible and accurate meeting timings as it is the main selling feature of SyncUp.

Test S/N	Testing Objective	Steps Taken	Expected Results	Actual Results
1	Finding the common busy slots in a group with only one user should output the time-blocks of that user.	1. Pass in the free slots that are within working hours in this format: `{ "2023-07-14": ["1230-1330", "1500-1600"], … }` 2. Call the function `GetCommonTime.findCommonBusySlots` with a time-period of 7 days	The busy slots that are in the output are just arrays of the slots without the date. e.g., for 14 July, we are expecting to see `["1230-1330", "1500-1600"],` within the list.	Pass

2.	Finding the common busy slots in a group with only one user should output the time-blocks of that user.	1. Similarly pass in the free slots, however, now in the availabilityDataList, there are 2 sets of availabilities belong to the 2 users. 2. Call the function `GetCommonTime.findCommonBusyslots` with a time period of 7 days.	The common busy slots between the 2 users should be the ranges of timings that overlap in the original set of availabilities provided.	Pass
3.	Testing the function that finds the combined workingHoursFreeSlots from the commonBusySlots	1. Pass in the `commonBusySlots` for the day (i.e., timeslots where all members in the group are occupied) 2. Call the function `GetCommontime.findWorkingHoursFreeSlots` to get the common free slots within working hours.	The output should just be a nested array of strings indicating the available free slots within working hours based on the `commonBusySlots` of the members provided.	Pass
4.	Test the function that returns the suggested slots for each day from the combined `workingHoursFreeSlots`	1. Pass in the `workingHoursFreeSlots` . 2. Call the function `getSlotsSuggestionsHelper` with a specified meeting duration (`selectedPeriod`).	The output should indicate the start timings of all possible slots.	Pass

User Testing

User testing was conducted with some students who were mostly friends of ours. User testing is the process of having end users test and evaluate the application. iOS users used TestFlight to run our app while Android users were provided our APK file.

We had set up a google spreadsheet and collated the feedback. We were able to correct most of the relevant/crucial issues raised by our users:

S/N	Feedback/Suggestion	Response taken/Clarification/Remark	Status
1	Greeting on homepage overflows if name is too long	Used `Expandable` widget to make the dimensions dynamic, so that `...` is displayed in case the content exceeds the dimensions allocated to the widget	Completed
2	Google calendar events sometimes do not show up on Personal Calendar page, only shows up after a few refreshes.	Use a `FutureBuilder` to contain the tiles displaying the events, so that the rendering is only complete after the query to google calendar API is	Completed

		complete (basically block rendering until the `async` call is done)	
3.	Sometimes group events should be scheduled outside of the 0900-1700 window, especially if the group is between friends, for recreational purposes.	Added the `isRecreational` boolean field to `Group` documents in our Firestore Database so that the meeting hours are instead 0000-2359 for recreational groups, and 0900-1700 for working groups.	Completed
4.	Meeting hours should be dynamic and be able to be set/voted upon by the users in the group, so that they get meaningful meeting time suggestions	To be completed by splashdown	In progress
5.	Groups should be able to set a group icon for easy identification	Implemented group icon feature	Completed
6.	Group meetings are sometimes not displayed immediately after creation, and only display after refreshing the page	Re-arranged `async` function calls when creating a group event such that the user is only brought back to the group calendar page after the API call is complete and the group event has been generated on google calendar. Also made sure that syncing is triggered upon creation of event to update availability.	Completed
7.	Since the app is heavily dependent on the google ecosystem, google contacts should be able to be easily added to groups, to make the collaboration process even more seamless	To be completed by splashdown	In progress

Tech Stack

1. Flutter and Dart (Cross-Platform Mobile App Development)
2. Firebase and Firestore (Account Authentication and Backend Services)
3. Google-Sign-In and Google Calendar API (For interaction between our application and actual Google Account Data)
4. Git and Github (Version Control)

Project Log

Refer to this link.