IM3080 Design and Innovation Project

(AY2021/22 Semester 1)

Project Report

Title: CoHab - The One Stop Platform for Communal Living

Github: https://github.com/JiXiangChua/Cohab

Video: https://youtu.be/3NvfqOknjGY

Submitted by: Group 6

Supervisor: Mohammed Yakoob Siyal

Content

Section 1: Background and Motivation

During the beginning stages of discussion, the team had differing ideas on what existing application to choose from and had different suggestions for improvements. However, one particular application caught our attention—Roommate. This application helps to ease communication among roommates to make cohabitation an easier process. As university students who live on campus with fellow students, Roommate was relevant and of interest to us. Yet, upon further exploration, we discovered it fell short of satisfying our needs.

There is a very limited number of applications using a similar concept to Roommate. Among all of them, the best application overall was Roommate. However, Roommate has a messy user interface, is confusing to navigate and has limited functionality.

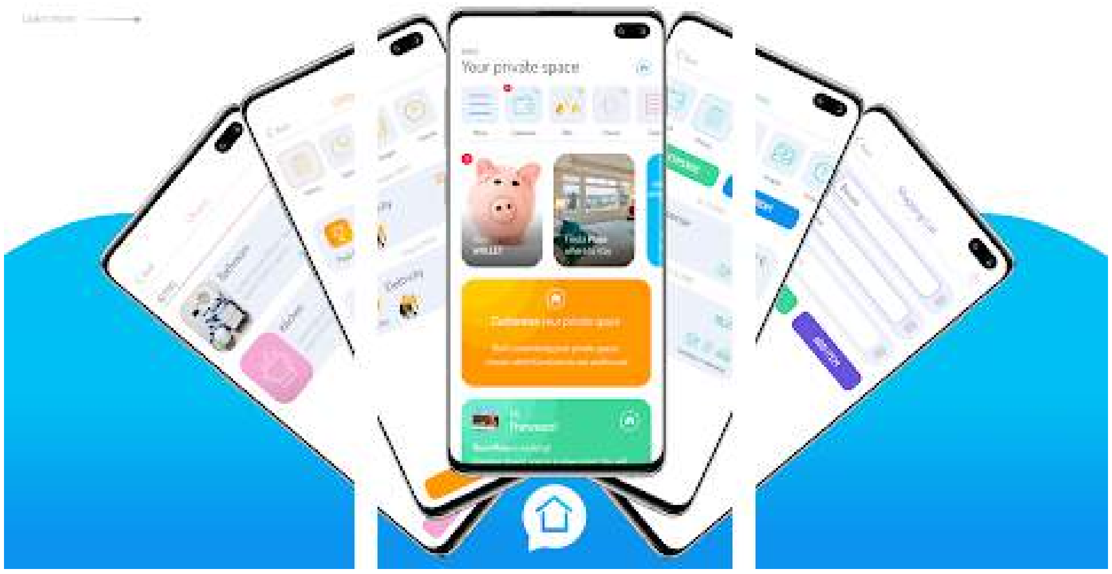


Figure 1.1. User Interface of Roommate

Surveys were sent out by the team to investigate the user demands and desired functionalities of an application based on the concept of cohabitation. After receiving various responses, we discovered that among respondents who are part of the co-living community, they experience problems that would make applications like Roommate, reinforced with added functionalities, relevant and useful to them. Thus, we decided to choose Roommate as our reference, inspiring the creation of CoHab.

Section 2: Objective

The purpose of CoHab is to serve as a one-stop platform for all things related to communal living. It serves to provide convenience for cohabitants by allowing them to organise household and financial matters or life events in a centralised application, with the main target audience of CoHab being roommates who live together.

This project aims to develop an improved version of an existing app—Roommate—by adding more functions while maintaining a simple and intuitive design. In addition to creating an avatar to serve as CoHab's mascot and guide for users, we have 3 core pillars of functions for the application:

- Finance—where expenses and bills among roommates are tracked. This feature records the total expenditure on shared resources and connects roommates' credit cards to CoHab for transactions among them.

- Chores and Tasks—which allow planning of shared household chores and tasks. Roommates can plan out chores, such as laundry duties and cleaning, as well as collate a to-buy list. Users can create chores or tasks for anyone to take up or allocate a specific individual to carry it out, which will then be available for everyone to view.

- Calendar—which coordinates roommates' personal schedules. This allows cohabitants to update any new plans they might have in CoHab and keeps all roommates informed of all upcoming plans together.

Section 3: Review of Literature and Technology

In this section, the justifications behind various software and developmental tools used in creating CoHab will be given.

3.1. User Interface & User Experience (UI/UX) Design Technologies

Adobe Illustrator (AI) is an industry-standard vector graphics editor and design program developed by Adobe Inc. For IM3080, AI was chosen as the tool to create 2D graphics for the CoHab application.

Adobe After Effects (AE) is an industry-standard motion graphics and visual effects software developed by Adobe Inc. For IM3080, AE was chosen as the tool to create video animations for the CoHab application.

Adobe Media Encoder (AME) is a video media transcoding utility developed by Adobe Inc. It is tightly integrated with video applications like AE to form a seamless workflow. For IM3080, AME was chosen as the tool to compress and reduce media file size in order to optimise the performance of the CoHab application.

Blender is a free and open-source 3D graphics software. For IM3080, Blender was chosen to create 3D modelled graphics for home furniture, buildings and CoHab's avatar. Additionally, it was used to implement animations for a dynamic UI.

Figma is an online user interface design and prototyping tool that allows collaboration. It was chosen to be used for wireframing and user-interface design prototyping as it allowed our team to brainstorm, visualise and discuss different possible ideas in the initial stages of ideating and conceptualising the CoHab application.

3.2. Front End Development Technologies

React Native is a front end development framework developed by Facebook and released on March 26, 2015. It is a subsidiary following after React and the purpose was to meet its growing mobile needs. The application is written using a mixture of JavaScript and XML-esque markup language, known as JSX. React native focuses significantly on the native UI elements and additionally leverages existing means of rendering views regardless of the platform used.

Expo Command Line Interface (CLI) is a framework and a platform for developing React applications. It is an alternative way to run developing applications as compared to using React Native CLI. The Expo CLI allows us to run our app on a website, an android emulator with Android Studio or an iphone simulator with Xcode with the same exact code. Expo also provides many third party libraries that are ready to be used and integrated into the mobile app.

3.3. Back End Development Technologies

Tomcat Server and MySQL were respectively used as a "pure Java" HTTP web server and database management system.

When a client makes a request, the application queries the database and produces a response in JSON format. This JSON data can be parsed with most programming languages. Ngrok proxy was used to easily access the server remotely. We implemented the API and test functionality in Postman, an API platform for building and using APIs. Postman simplifies each step of the API lifecycle and streamlines collaboration so the front end team can create APIs faster and the back end team can develop and test APIs without the front end part. Hence it provides a flexible timeline for back end and front end.

Regarding front end integration, HTTP methods were used for RESTful Services (GET/POST/PUT/DELETE) to call the APIs, which require JSON format as the POST data. Each request returns a JSON object as a response.

Section 4: Design

This section contains the thinking process behind the various user-centric design decisions made for the CoHab application.

4.1. Overall Design Considerations

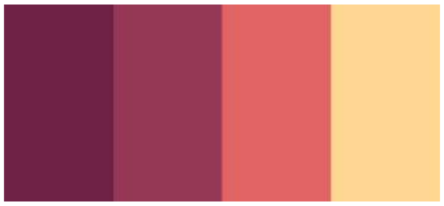


Figure 4.1. Colour palette featuring #6E2142, #943855, #E16363 and #FFD692

One of the design goals behind CoHab was to create a welcoming, cozy and homely atmosphere for users to enjoy. Therefore, the colour palette as shown in Figure 4.a is generally followed throughout CoHab as it consists of warm colours and a balanced mix of colour values.

Another goal was for the graphics in CoHab to follow a mix between isometric vector graphics and 3-dimensional (3D) rendered graphics. This would help CoHab stand out as a unique application and allude to gaming applications, which commonly use 3D graphics, to imbue a sense of fun and excitement. In the following subsections, the details of the design considerations for CoHab's logo, avatar and individual screens will be shared.

4.1.1. CoHab Logo & App Icon

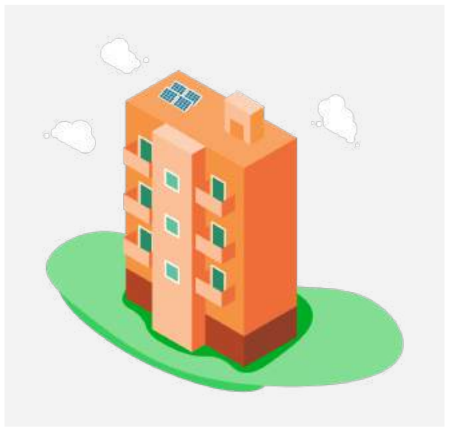


Figure 4.2. CoHab logo

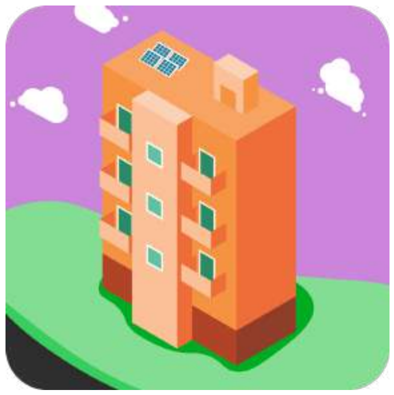


Figure 4.3. CoHab Application Icon

Design Inspiration
As cohabitation is the quintessential concept behind CoHab, the logo (Figure 4.2) consists of a shared building housing a group of home units. It is surrounded by a natural environment of grass and clouds to create a welcoming and relaxing atmosphere. In the application icon (Figure 4.3), a black patch by the green grass resembles a road to fill up the environment.

Overall Art Style
The logo is designed in a vector art style and adopts an isometric perspective to match the rest of the graphics in CoHab. Simplicity, in the form of solid colours, minimal hues

and organic shapes, is retained.

Colour Palette

The colours used are balanced between pastels and saturated hues to keep the design vibrant and lively while maintaining a soft and comforting look. The building is made up of warm orange-y hues and the purple background of the CoHab application icon closely follows the colour scheme in Section 4.1.

4.1.2. Navigation Icons

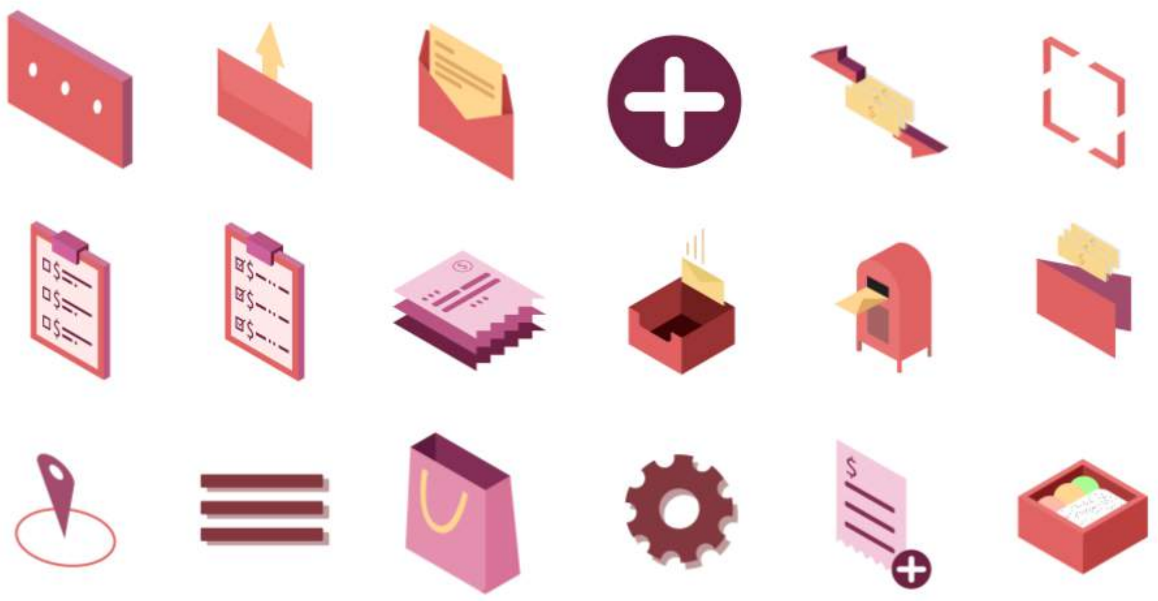


Figure 4.4. Customised CoHab Navigation Icons

Design Inspiration

Customised navigation icons were created to bring CoHab a sense of originality and to better suit the overall colour scheme of the application. The icons are designed in a vector art style and are in an isometric perspective to match the rest of the graphics in CoHab.

4.1.3. Avatar

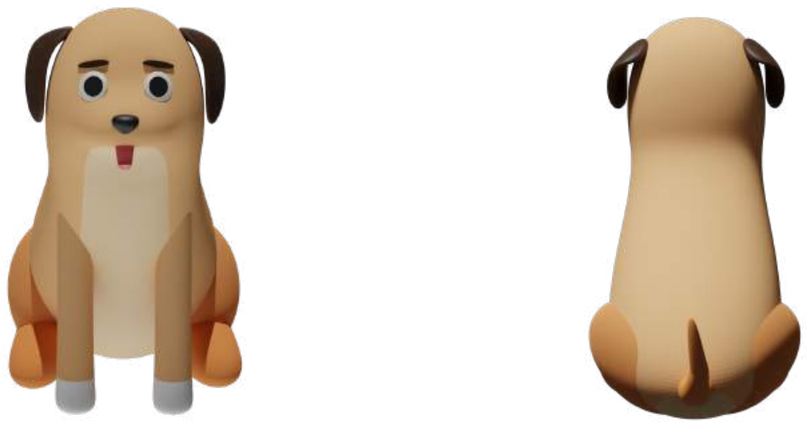


Figure 4.5. CoHab Avatar Model Front and Back view

Design Inspiration
The avatar was created with the intention of providing users with an initial contact point to introduce them to the rest of the application. Similar approaches can be seen in games such as Dave in Plants VS Zombies, or Nook in Animal Crossing.

As for CoHab, the avatar is named Carl, to match the first letter in the name of the application, as well as to give the avatar a more defined identity. Carl is not a common name for dogs, hence it would stand out to users as unique.

The design of the avatar is a dog because it is a common household pet that conveys the notion of warmth and comfort. The avatar will appear to be befriending its new owner upon their arrival, helping users to treat the room as their home.

Colour Palette
Lighter shades of orange are used to resemble a golden retriever, due to the species' traits of being gentle, friendly, and trustworthy. The ears were made brown in order to frame the avatar such that it would not be hard to see above light backgrounds. Finally, white socks were given as it is determined by most to be a very cute feature, and visually appealing avatars tend to lead to higher user retention.

4.1.4. Splash Screen

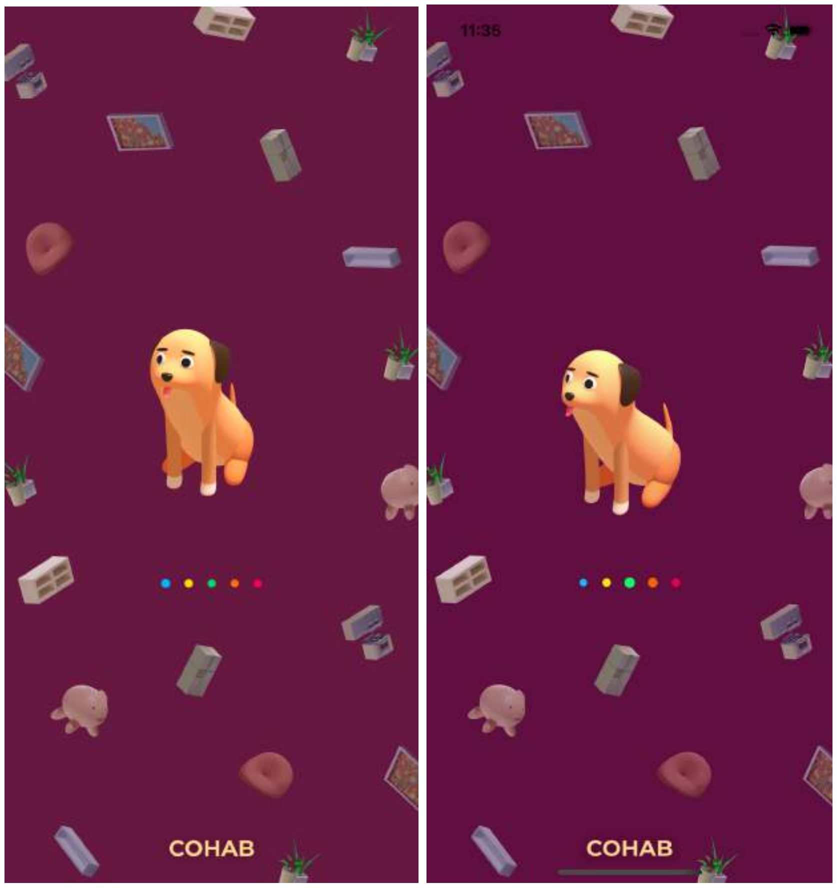


Figure 4.6. Splash screen animations

Design Inspiration
CoHab's animated splash screen was designed to create a positive first impression to set the tone for the rest of the application. The gentle bobbing animation of CoHab's avatar, Carl, greets the user accompanied by animated dots moving in waves as the application loads. Furniture and household items in the background surround Carl to fit CoHab's theme of home-living.

Overall Tone
The endearing and familiar appearance of Carl and the surrounding furniture creates a cozy, homely atmosphere to welcome the user. The gentle animations induce a soothing sensation while also introducing a sense of dynamism and liveliness to paint CoHab as a fun and relaxing application.

Colour Palette
The splash screen follows the colour scheme in Section 4.1.

4.1.5. Login/Signup Screen

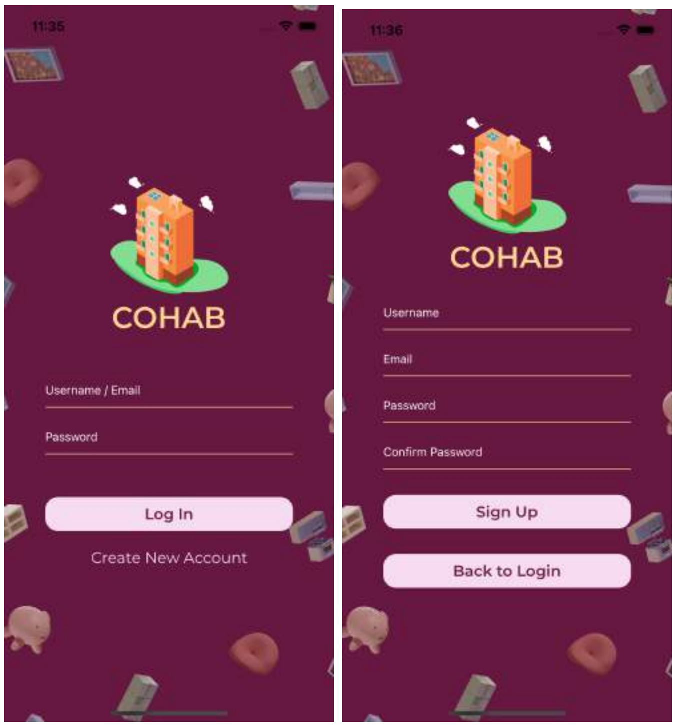


Figure 4.7. CoHab Login and Sign up screens

Design Inspiration

As the login and signup screens serve as the "title pages" of an application, the CoHab logo and name are displayed. The same background as the splashscreen is used for a sense of continuity. The login screen follows the common standard of allowing users to access their account by a unique username or email and password. When creating a new account, users are to key in their credentials and to confirm their password. These fields are included on the screen. The bare minimum is displayed on these screens so as to not overwhelm the user in the beginning.

4.1.6. Group Screen

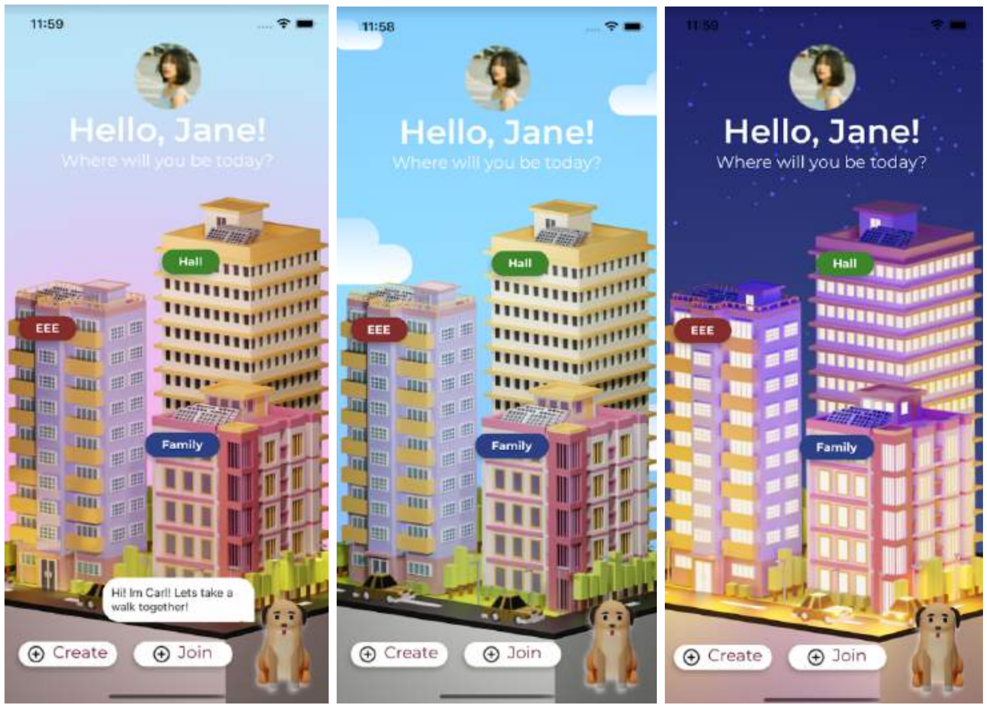


Figure 4.8. CoHab's Group Screens

Design Inspiration

As the aim for CoHab's design was to create a welcoming, cozy and homely atmosphere as well as engage users on a long-term basis, the group screen and home screen were designed to simulate a virtual semi-realistic world that the users can personalise and immerse themselves in.

Thus, the group screen simulates reality by representing the groups as buildings. When users first log in to the application, they will see an empty plot of land. As they create or join groups, a new building will appear, parallel to a real-life addition of a new address. In order to make the group screen more realistic, there are three versions of the group screen simulating different environments at different times of the day - morning, afternoon and night. The group screen will then update with different times of the day.

We felt that users would grasp a better understanding of our app by doing an interactive walkthrough. Hence, we incorporated an in-app tutorial in the group screen with the intention of providing interactivity, convenience, and efficiency. As the avatar is our bridge in connecting the user and our CoHab app, we decided to make the avatar our start button of the tutorial.

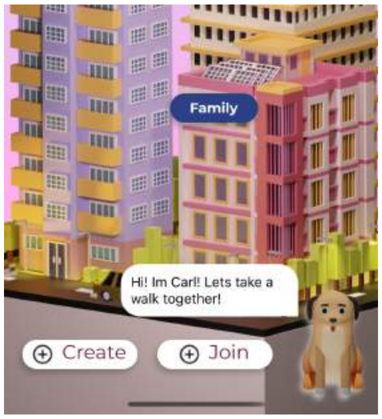


Figure 4.9. Group Screen Avatar

Subject Matter
The buildings were designed with inspiration from HDB buildings and NTU halls. The pastel colours of the buildings were also inspired by how HDB buildings around Singapore were painted. The animated cars and surrounding trees were then added to create a homely environment, allowing the user to experience a sense of familiarity as the scenery parallels a typical Singaporean neighbourhood.

To make the avatar more noticeable, it was placed at the bottom right of the screen with an added speech bubble that would blink every few seconds.

As one of the selling points of the app was being able to create and join multiple groups such as family and hall, the dog avatar is used to guide users on how to navigate through the app.

Overall Tone
The overall tone for the group screen is to invoke a sense of familiarity for the user and allow them to feel at home and at ease when using CoHab.

To make the tutorial more cosy and lively to fit the theme of the app, we decided to utilize a dog's perspective when crafting the content of the tutorial guide.

4.1.7. Home Screen

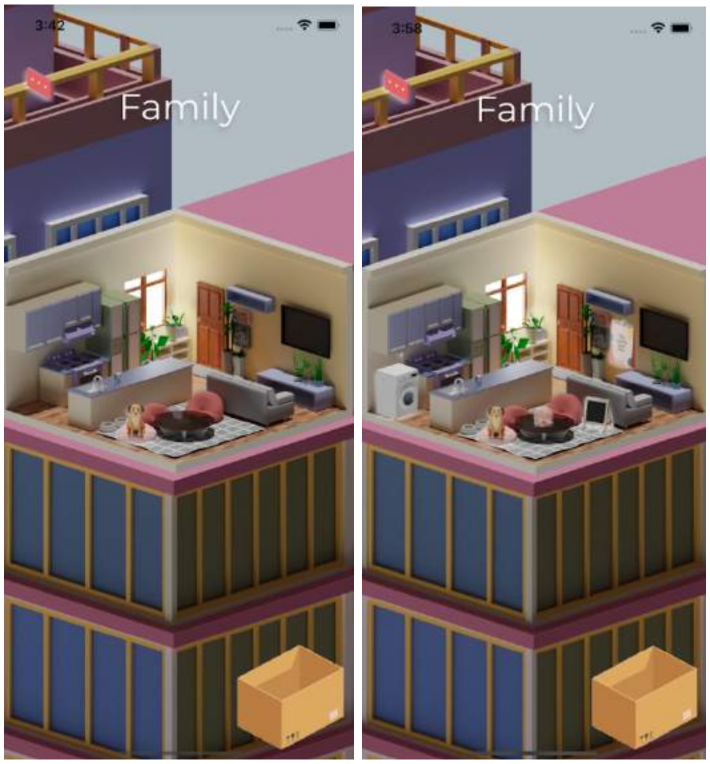
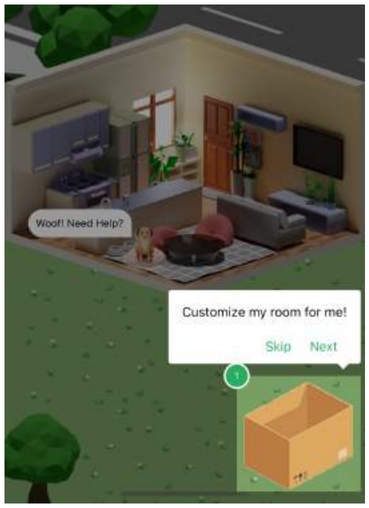


Figure 4.10. Home Screen Before and After Furniture Is Added


Woof! Need Help?

Customize my room for me!

Figure 4.11. Home Screen Tutorial

Design Inspiration

Following the theme of the group screen, the home screen is designed to simulate the environment of shared spaces in a house, specifically the living room and the kitchen. The furniture is also designed to encapsulate the main functions of the app, so that users will be able to attach a default function to the added furniture if they do not wish to personalise. For example, the table with the piggy bank represents the Wallet function and the calendar represents the Calendar function. For the surrounding environment of the room, it is linked to the building the current group is represented by in order to enhance the cohesiveness of the user experience, i.e. if the group was represented by the pink building in the group screen, the room will also be situated in the pink building. The colour tone of the furniture and room also follows the pastel colour scheme of the group screen.

Subject Matter

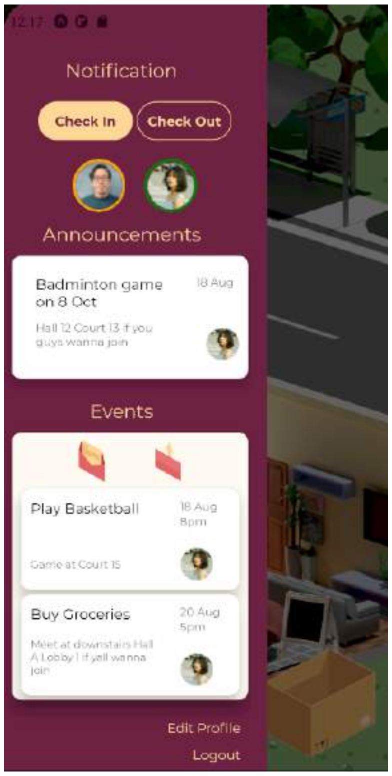
The home screen is designed to be more engaging, intuitive to use, and personal to users. The interface design resembles a game scene, prompting users to scroll, while the highlights and tutorial would assist users in recognizing the input controls. Finally, the room would act as a virtual space that mimics their actual abode, allowing users to experience the interface the same way they would in their real residence.

As the home screen is also one of our main focus of the app, we wanted users to grasp a better understanding by doing an interactive walkthrough. The avatar which is our bridge between the users and the app would also be inside the room on the home screen page.

Overall Tone

For the content of the tutorial guide, we took it from the perspective of a dog. As our unique selling point of the app is the customisable furniture, we decided that it was necessary to include that in the in-app tutorial and highlight that feature.

4.1.8. Notification Screen



Badminton game on 8 Oct
18 Aug
Hall 12 Court 13 if you guys wanna join

Play Basketball
18 Aug 8pm
Game at Court 15

Buy Groceries
20 Aug 5pm
Meet at downstairs Hall A Lobby ! if yall wanna join

Figure 4.12. Notification Screen Menu

Subject Matter
The point of CoHab is primarily to aid communication between people living together. All main features within the group tab are condensed here for easy access. The check in and check out function is also here, which serves as a key method of communication between roommates.

Overall Tone
The notification screen summarises all the major events and information in the group, including containing the key announcements from other people. As such the screen is compact and fits into a menu at the top left of each group.

Colour Palette
The splash screen follows the colour scheme in Section 4.1.

4.1.9. Finance Screen

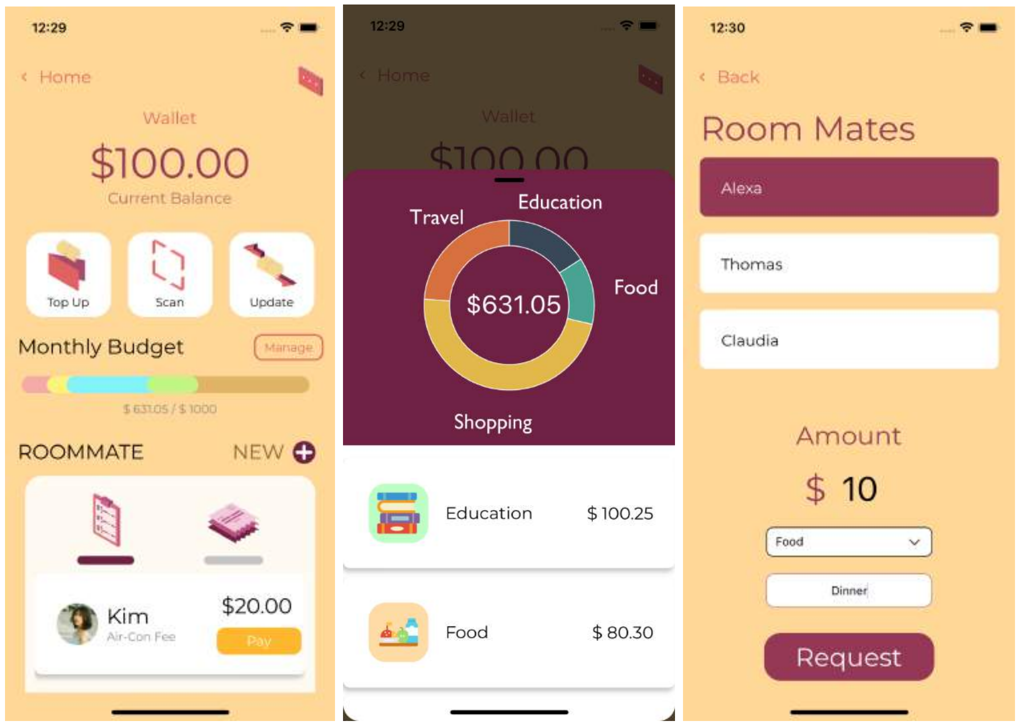


Figure 4.13. Finance Screen

Subject Matter
For the finance screen, we designed it to show the user's wallet balance and the important buttons to manage finances at first glance. The finance screen is split into 3 sections, the monthly budget section, roommate section and the group section with the use of scrollview.

Colour Palette
With the intention of making the monthly budget section appear more visually appealing to the user, we colour coded the expenses to express the different categories of expenses and used a progress bar to display. When the progress bar is clicked, a bottom sheet view will appear which would contain a donut chart for users to see their expenses. We chose to use a donut chart as donut charts have a simple structure and are reader-friendly.

Overall Tone
We chose to use a donut chart for the budget analysis as donut charts have a simple structure and are reader-friendly.

When paying their roommate, a pop-up modal will appear where users can key in the amount as well as select the item they are paying for with the use of a drop-down picker.

The group section is different from the roommate section as it allows users to split with other people. We got the inspiration for this section from the “Bill splitting” function in Google Pay which provides convenience for users. We decided to implement this as we found out from our survey that splitting shared bills is often something people want to be able to do without much trouble. When creating a bill to split within a group, users will be directed to another page where they can select the payee as well as the amount and category of expenses.

4.1.10. Chores Screen

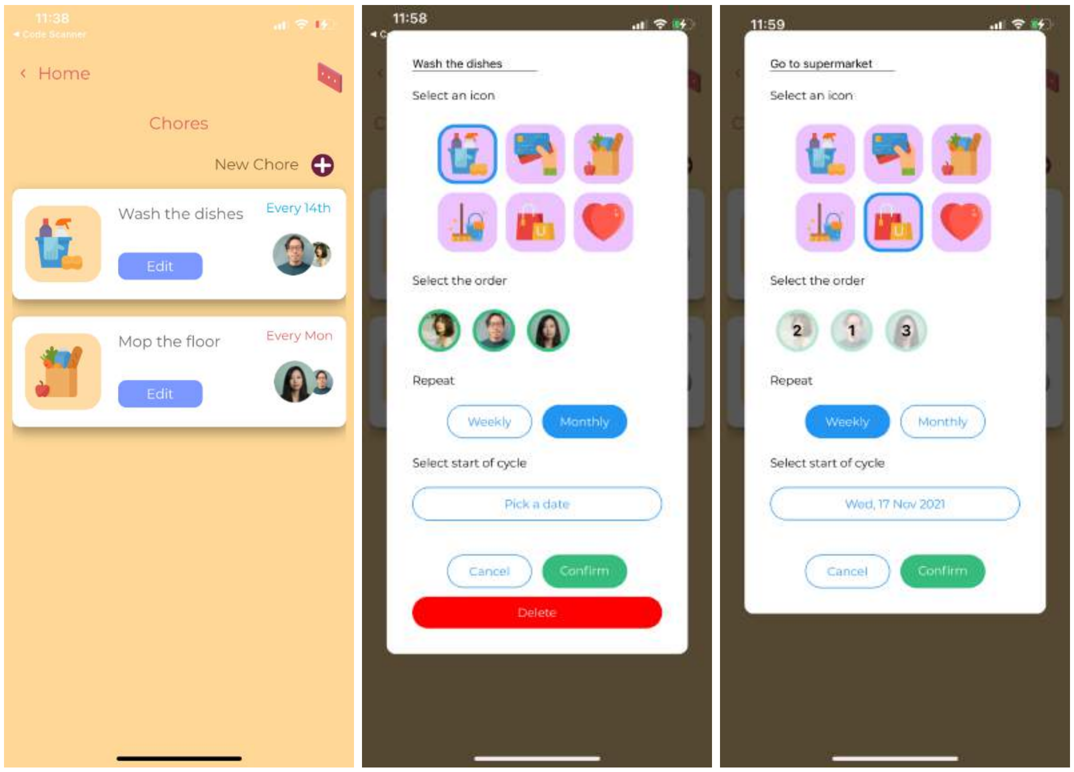


Figure 4.14. Chore Screen

Subject Matter
For the chores screen, the goal is to set up a task, usually a chore, that automatically repeats for the users. The users should be able to clearly tell how often the chore repeats, when it is due, what kind of chore or task it is, and most importantly, who is currently responsible for doing it. Adding and editing chores is also meant to be hassle-free for the users.

Overall Tone
For it to be easy for the users to look at it quickly and understand, the chore cards are designed to be neat, with the title and icons being its priority.
As editing and adding chores is meant to be simple, icons and short tabs are being shown instead. Thus users can simply pick the icon displayed and the order without any

scrolling. It is also easy to change the order of selection, simply by tapping the icon again.

Colour Palette
This screen also uses the same colour scheme as the rest of the screens. The buttons and icons use milder colors that would be easy on the eyes.

4.1.11. Tasks Screen

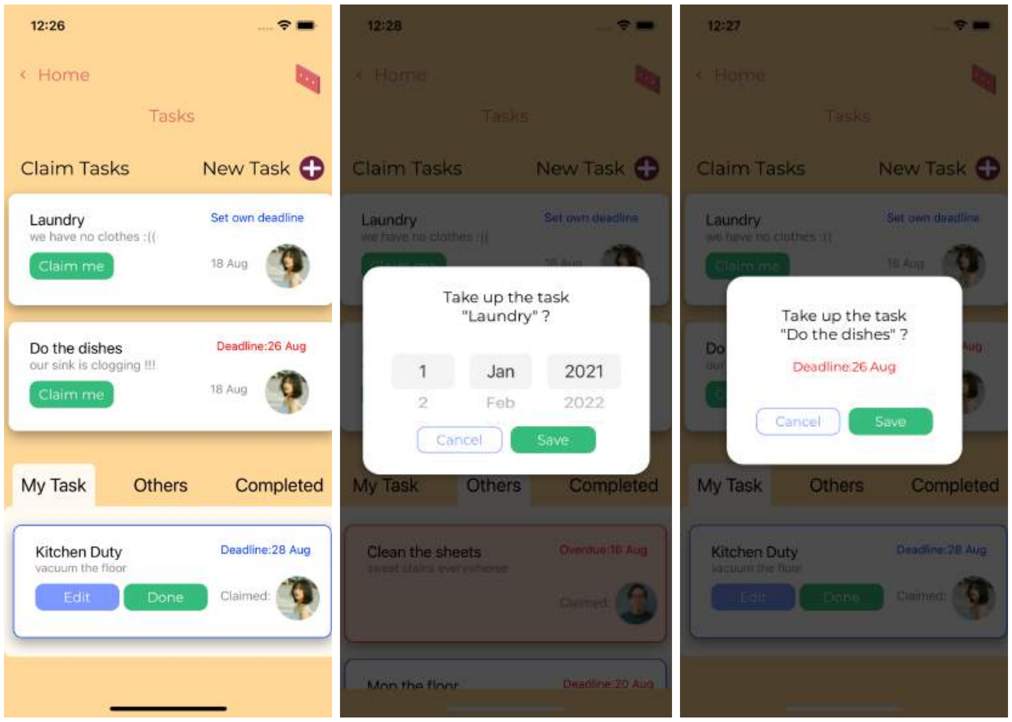
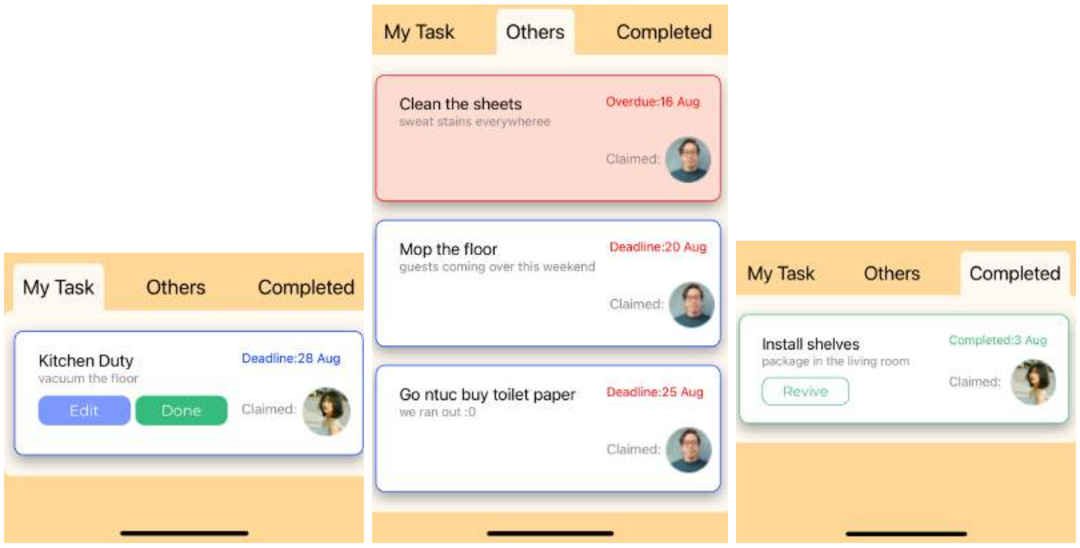


Figure 4.15. Task Screen

The main feature that separates the task screen from the chores screen is that chores screen is used for repetitive weekly or monthly routines whereas tasks screen is used as a reminder for single, unrepeated events.

Subject Matter
For the task screen,the page is divided into 2 different sections. Mainly, a section for users to claim tasks and another section to keep track of tasks.
When adding the new tasks, a pop-up modal will appear where users can select if they would like to fix the deadline for their roommate or allow them to set their own deadline.
For the “Claim Tasks” section, users can claim their respective tasks. If the deadline is not set, they can edit their deadline with the use of a date picker and the date will be updated. If the deadline is set, their task can be seen in the “My task” section below. Users can view their own tasks in the “My Tasks” section, their roommate’s task in the “Others” section and finished tasks in the “Completed” section.



Kitchen Duty
vacuum the floor
Deadline:28 Aug
Claimed:

Clean the sheets
sweat stains everywheree
Overdue:16 Aug
Claimed:

Mop the floor
guests coming over this weekend
Deadline:20 Aug
Claimed:

Go ntuc buy toilet paper
we ran out :0
Deadline:25 Aug
Claimed:

Install shelves
package in the living room
Completed:3 Aug
Claimed:

Figure 4.16. Viewing Task Tabs

One of the main selling points of this app is that there is a section to view missed/overdue tasks. The missed task card has a prominent red background as compared to others so as to serve as a reminder and notify the user.

Colour Palette

The task screen also uses the same colour scheme as the rest of the function screens.

4.1.12. Calendar Screen

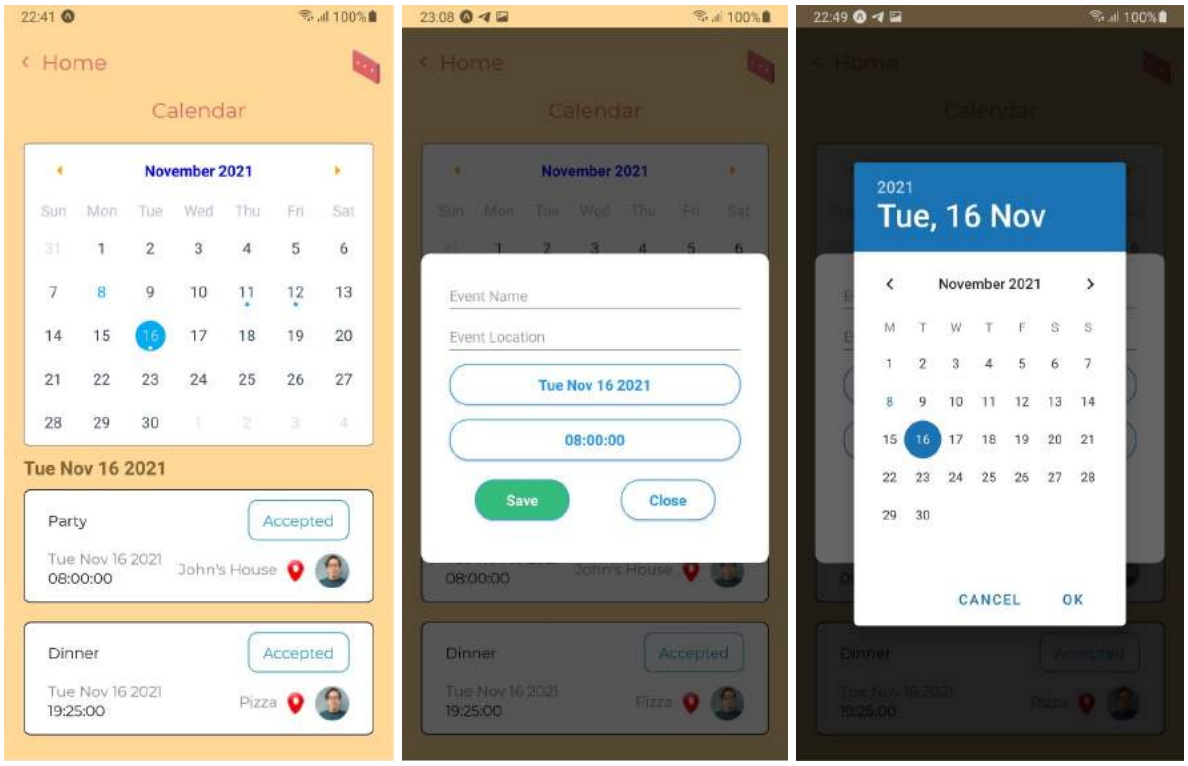


Figure 4.17. Calendar Screen

Subject Matter
The goal for the calendar was to be as simple and intuitive for users as possible. Since most people have used calendars on their mobile phones before, an external library that replicated a standard calendar was used so that it would feel familiar. Any date that has an event scheduled will have a small blue dot under it so that users can see at a glance if any events are coming up.

Although most modern mobile phones already have a calendar function, they generally do not support sharing a calendar and events among multiple people. For people who are sharing a living space, it is common to have to plan events using that same space. Even if the user does not plan on attending an event, if the event will be held in their shared living space, it is important to keep track of it. As such, the main goal of the calendar screen is not to reinvent the calendar, but merely to allow multiple people to edit and use the same calendar and events function.

Overall Tone
The main calendar screen consists of a large calendar at the top of the screen and a list of events below it. The list of events will change depending on which date the user selects in the above calendar. To add a new event, the user has to tap on the same date

twice, which will open a pop-up modal screen where the user can input the details of their new event.

The list of events below the calendar shows all the events that are happening on a selected date. If a different date is selected on the calendar, the events shown below will also update. The most important events are generally the ones that are coming up soon, so we opted to show those upon first entering the calendar screen. The current date is the default selection, so the user can immediately see the events they have planned for that day.

4.2. Overall Design

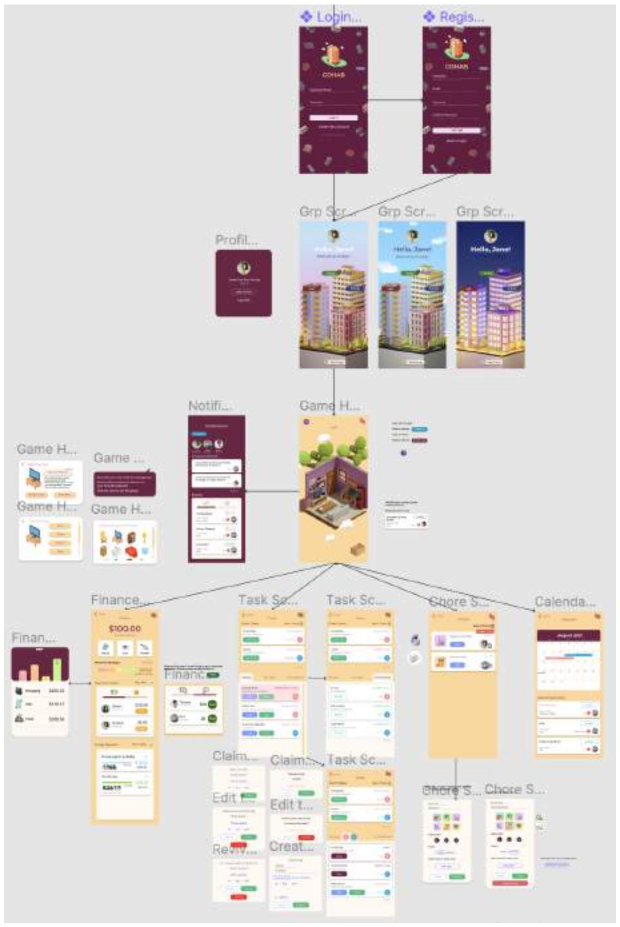


Figure 4.18. Our Application Design in Figma

The user first starts with the splash screen (not shown here) and then automatically proceeds to the login page. They can create an account to login, or login with their already-registered credentials. They then are logged in to a group screen where they can select the group they wish to manage.

Afterwards, they are navigated to the home page, where they can slide to the right to check the notifications and online/offline status of the roommates or people in their set-up group. They can also choose to navigate to the finance, task, chore or calendar pages.

Section 5: Implementation

5.1. Graphics

5.1.1. 3D

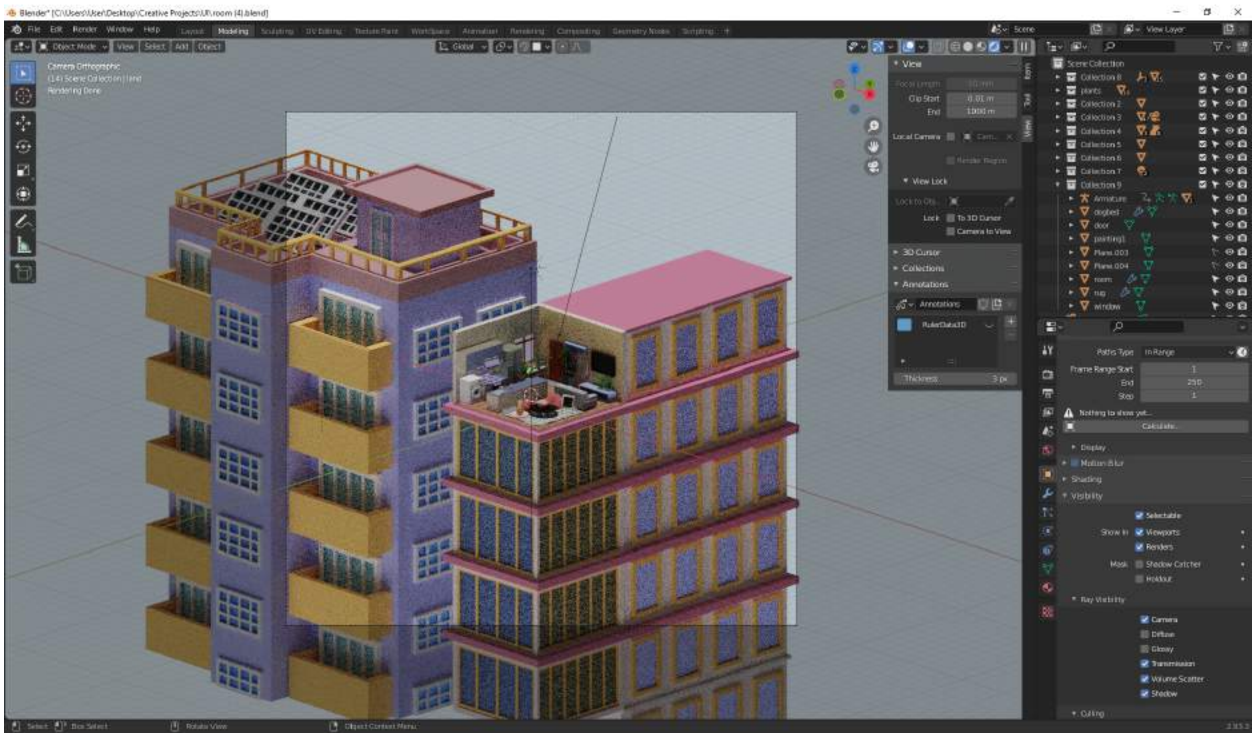


Figure 5.1. Blender

Blender was used for the designing and rendering of the 3D assets. For an open source software, Blender is very versatile and users are able to create and manipulate multiple 3D objects easily to create meaningful scenes.

Due to the graphic style of the app, models were made to look like simple geometric shapes, and colours were flat as far as possible, while retaining the identity of the object. For the group screen, features like “Array” were used to duplicate the floors of the buildings so as to hasten up the creation process. The car animation was also created in blender and rendered as a standalone animation to be later added onto the scene during the coding process.

For the home screen, the furniture was created with heavy use of the “Bevel” function to smooth out edges and make the scene look more realistic. “Subdivision Surface” was also used for furniture like the taps and sofa that require smoother edges. To achieve a rounder and cloth-like texture of the beanbag, a sphere was first sculpted with the “Sculpting” function and then dropped onto the plane, making use of the physics simulation in Blender.

All the assets in the home screen were created in a single scene to ensure cohesiveness and then rendered individually. Due to the requirement of rendering assets individually, objects needed to affect each other as little as possible, so light ray diffusion, shadows, and reflection were not used on some objects. Each object was then rendered as image sprites.

5.1.2. 2D

Adobe Illustrator was used for the graphical illustrations such as icons and logos, as it produces precise editable vector graphics that stay sharp when scaled to any size. It is a useful application to create clean, graphical illustrations.

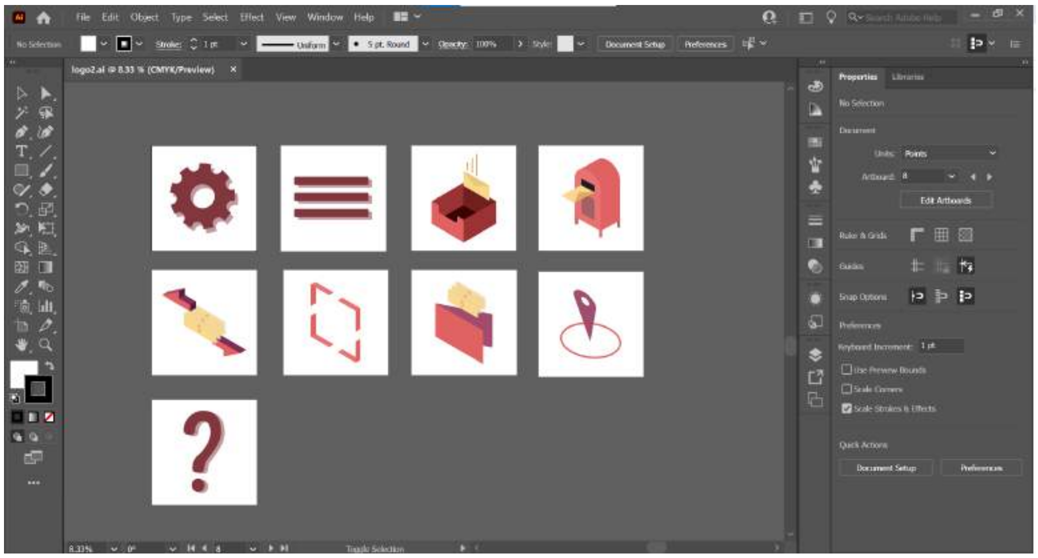


Figure 5.2: Adobe Illustrator

The illustrations were all drawn with the same colour theme and in an isometric view to ensure coherence. The grid tool was used to produce the same isometric grids as a guide for all the illustrations. The colour theme was edited slightly from the original colour theme, lighter and darker tones were added to emphasize the isometric view on the illustration. After the graphical illustrations are completed on Illustrator, they are exported and added into the database, so to be retrieved and displayed on the mobile application. Some were also coded into the application as well.

5.2. Front End Development

5.2.1. React Native Components

React Native provides a few general native components that can be used without importing any external libraries. The most commonly used in our project are View, Touchableopacity, Text, Image, and TextInput.

The View component is essentially the building block of React Native. They are essentially empty containers, much like the <div> tags of HTML that can be moved around at will. They are used all over our app, mainly for formatting purposes.

The TouchableOpacity component is the main way we make buttons, or anything that users will interact with. It works similarly to the View component, able to contain other components inside it, but with an additional feature: when clicked on, it becomes translucent for a brief moment, as users have come to expect from mobile interfaces. We use buttons on almost every screen, so the TouchableOpacity component is a necessity.

The Text component is, as it sounds, a component that displays text. It is required to display text because of the way React Native's rendering engine works, and so is used to display text within our app.

Similarly, the Image component is how React Native displays images. It makes it easy to import images, as well as change their height and width, but otherwise works exactly like one would expect.

The TextInput component is used to implement text fields. They are similar to the Text component in that they both display text, but only TextInput allows for user input. We use the TextInput component on a few screens where we have to enter names or other information.

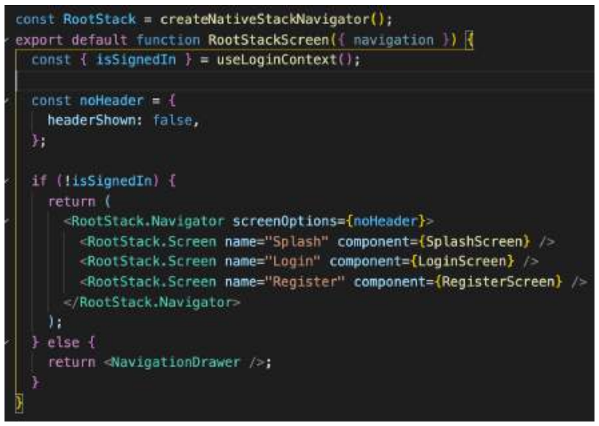
5.2.2. React Hooks

Throughout our project, we used a lot of useState which is a hook provided by React Native. A hook allows us to use state and other react features without writing a class. A hook is a function that "hooks into" the react state and lifecycle of a web or mobile application. The useState hook allows us to handle reactive data and any other data that might change in the application as a result of user interaction. Data values that change are referred to as state and whenever a state changes, React will re-render the entire user interface shown on the screen. As such, in our project, a lot of the data is created using a useState hook.

The next hook that is widely used throughout the application is the useEffect hook. Prior to the introduction of hooks, React developers have been creating classes to manage the application's lifecycle. Developers would create a class and have methods such as constructor(props), componentDidMount(), componentDidUpdate() to manage the lifecycle. The useEffect hooks allow us to perform “side effects” in function components and implement the lifecycle hooks from within a single function. We can also write functions inside this hook and get executed again whenever a specific state changes in the application. For example, useEffect hook was used to fetch and send data from the mobile app to the database.

Lastly, the useContext hook was used in login and registration authentication codes as well as declaring global variables. This hook allows us to work with the React’s Context API. It provides a mechanism to share data easily within its component tree without passing through props. For instance from a child component view, instead of accessing data from its parent component using props, the child can access the value directly when placed within a useContext component. This provides an easy alternative especially when the child and parent component are multiple layers apart and accessing through props will be tiresome.

5.2.3. React Native Navigators



```
const RootStack = createNativeStackNavigator();
export default function RootStackScreen({ navigation }) {
  const { isSignedIn } = useLoginContext();

  const noHeader = {
    headerShown: false,
  };

  if (!isSignedIn) {
    return (
      <RootStack.Navigator screenOptions={noHeader}>
        <RootStack.Screen name="Splash" component={SplashScreen} />
        <RootStack.Screen name="Login" component={LoginScreen} />
        <RootStack.Screen name="Register" component={RegisterScreen} />
      </RootStack.Navigator>
    );
  } else {
    return <NavigationDrawer />;
  }
}
```

Figure 5.3. React Stack Navigator

A popular way to implement navigation in React Native is through the react-navigation library. React Navigation provides a few tools that make the job of navigation significantly easier.

In our project, we used a stack navigator as the root navigator, to handle user authentication. Once authenticated, the user will be brought into the main functions of the app. For the main functions, we implemented a drawer navigator as the main navigation system. The main difference between the stack and drawer navigators is that the drawer navigator allows for a ‘drawer’ to slide out from the left side of the screen. It would simplify our programming since we would not have to implement the drawer ourselves.

5.2.4. Modal

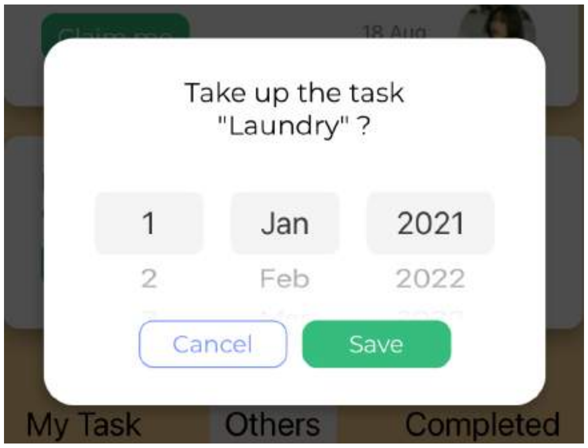
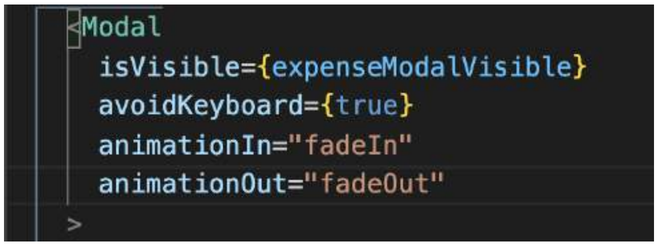


Figure 5.4. Modal Screen

Modals are a great tool for improving user interfaces. Essentially, a modal is a screen that appears above another screen, directing a user's attention towards critical information or guiding them to the next step in a process. An active modal disables the original screen, which is still visible, prompting the user to interact with the new one.

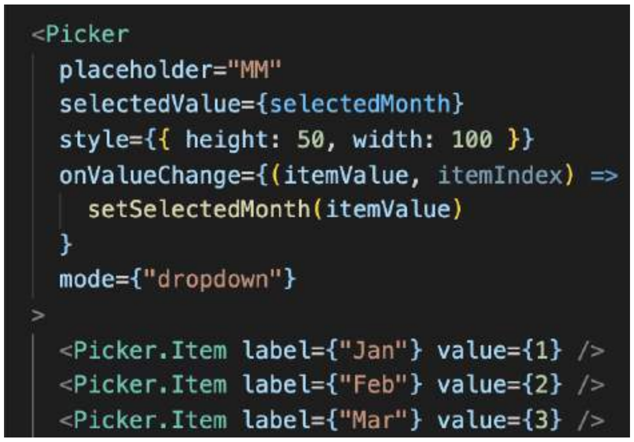


```
<Modal
  isVisible={expenseModalVisible}
  avoidKeyboard={true}
  animationIn="fadeIn"
  animationOut="fadeOut"
>
```

Figure 5.5. Modal component

The properties that we need to set for using a modal screen is shown in the figure above. The "isVisible" property as the name suggests sets the visibility of the modal visible or not visible by passing in the useState variable that contains a boolean value. "avoidKeyboard" property allows the modal to shift itself whenever the keyboards show up for users to input their description. Lastly, the "animationIn" and "animationOut" properties allow us to specify the type of animation that the modal will appear and disappear upon interacted by the user.

5.2.5. Picker

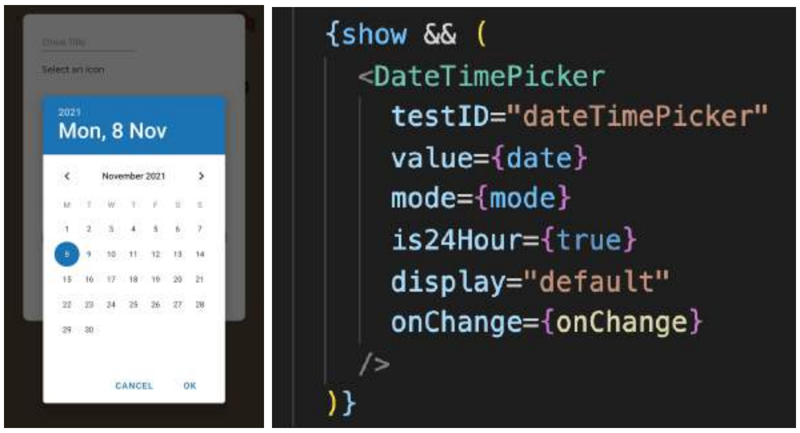


```
<Picker
  placeholder="MM"
  selectedValue={selectedMonth}
  style={{ height: 50, width: 100 }}
  onValueChange={(itemValue, itemIndex) =>
    setSelectedMonth(itemValue)
  }
  mode={"dropdown"}
>
  <Picker.Item label={"Jan"} value={1} />
  <Picker.Item label={"Feb"} value={2} />
  <Picker.Item label={"Mar"} value={3} />
```

Figure 5.6. Picker component

The <Picker> component from react-native-picker provided an easy way for us to implement drop down menus or an alternative interactive way for users to select date and time. In the figure above, we had to embed the picker's items inside the <Picker> component and this was denoted by the <Picker.Item> component. The <Picker.Item> component allowed us to input the values of the items and give each item a label to identify it. In the parent component, <Picker>, we essentially coded the necessary properties to implement the function. For example, the "onValueChange" property will execute a function and update the selected month whenever the user changes the selected value in the picker view.

5.2.6. Date Time Picker



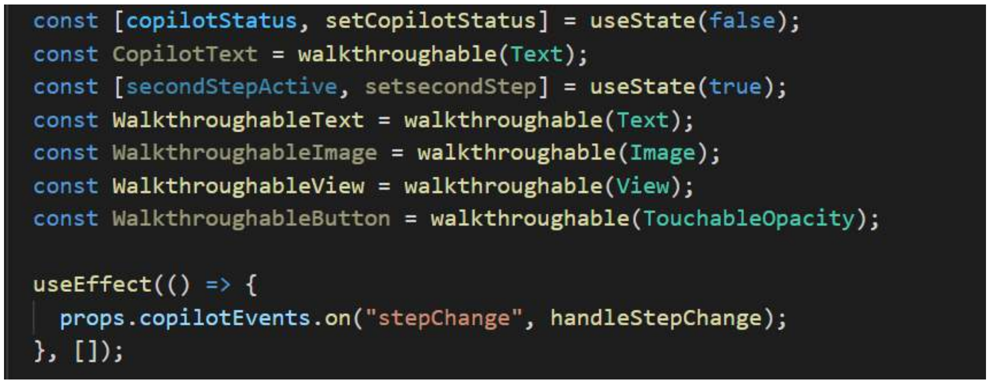
```
{show && (
  <DateTimePicker
    testID="dateTimePicker"
    value={date}
    mode={mode}
    is24Hour={true}
    display="default"
    onChange={onChange}
  />
)}
```

Figure 5.7. Datetimepicker component

<DateTimePicker> component was similar to the earlier <Picker> component. However, this particular library was more defined and suitable for implementation involving time. On top of being able to select month and date, the <DateTimePicker> component allows selection of time and we can specify the display format. It is a more graphical way of

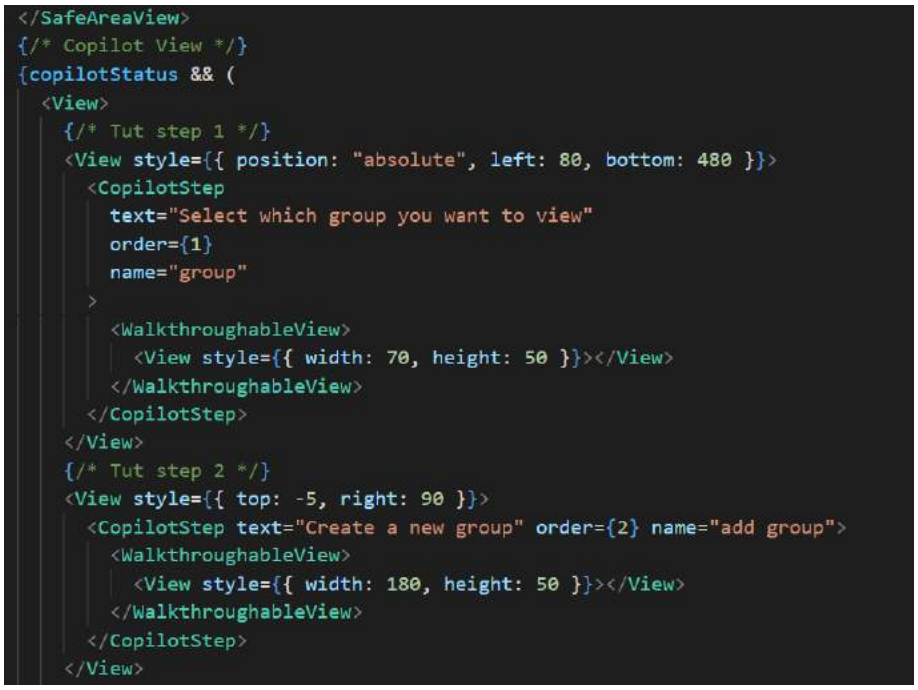
picking a date, as opposed to scrolling through a list of dates and months. From the figure above, the “show” was a state variable to control the visibility of the <DateTimePicker> component.

5.2.7. Copilot



```
const [copilotStatus, setCopilotStatus] = useState(false);
const CopilotText = walkthroughable(Text);
const [secondStepActive, setsecondStep] = useState(true);
const WalkthroughableText = walkthroughable(Text);
const WalkthroughableImage = walkthroughable(Image);
const WalkthroughableView = walkthroughable(View);
const WalkthroughableButton = walkthroughable(TouchableOpacity);

useEffect(() => {
  props.copilotEvents.on("stepChange", handleStepChange);
}, []);
```

Figure 5.8. Copilot variables

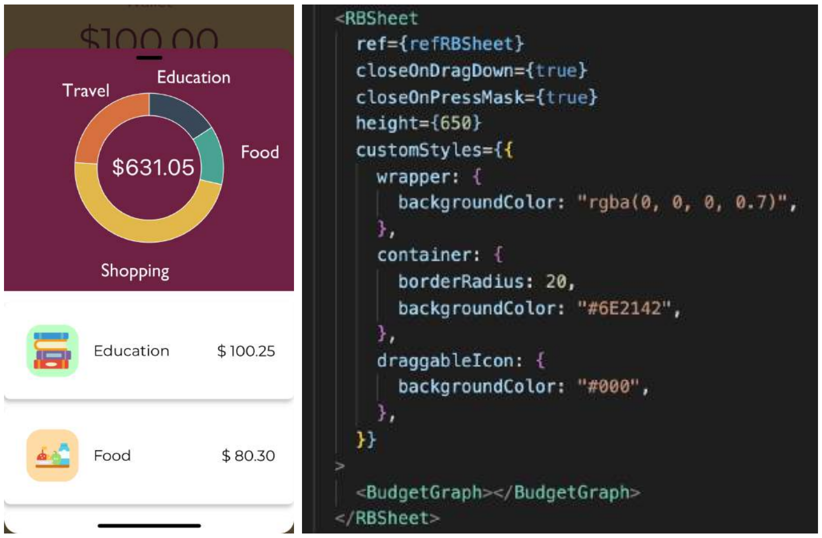


```
</SafeAreaView>
{/* Copilot View */}
{copilotStatus && (
  <View>
    {/* Tut step 1 */}
    <View style={{ position: "absolute", left: 80, bottom: 480 }}>
      <CopilotStep
        text="Select which group you want to view"
        order={1}
        name="group"
      >
        <WalkthroughableView>
          <View style={{ width: 70, height: 50 }}></View>
        </WalkthroughableView>
      </CopilotStep>
    </View>
    {/* Tut step 2 */}
    <View style={{ top: -5, right: 90 }}>
      <CopilotStep text="Create a new group" order={2} name="add group">
        <WalkthroughableView>
          <View style={{ width: 180, height: 50 }}></View>
        </WalkthroughableView>
      </CopilotStep>
    </View>
```

Figure 5.9. Copilot Component

For the group select and room screens, a step-by-step walkthrough for the tutorial using the Copilot library was achieved. To implement the tutorial, we need to use two important copilot components which are <CopilotStep> and <WalkthroughableView>. The component <CopilotStep> essentially allows us to specify the sequence of displaying each content. Inside the component, we input the message that we want to inform the user in the “text” property as seen in the above figure. Then for each sequence, we inserted the <WalkthroughableView> as a means to highlight a section on the screen that we want the user to see. As such, other areas on the screen except for this highlighted section will be dimmed. Hence, we are able to divert the user's attention to a particular function and guide them on the usage of it.

5.2.8. Bottom Sheet

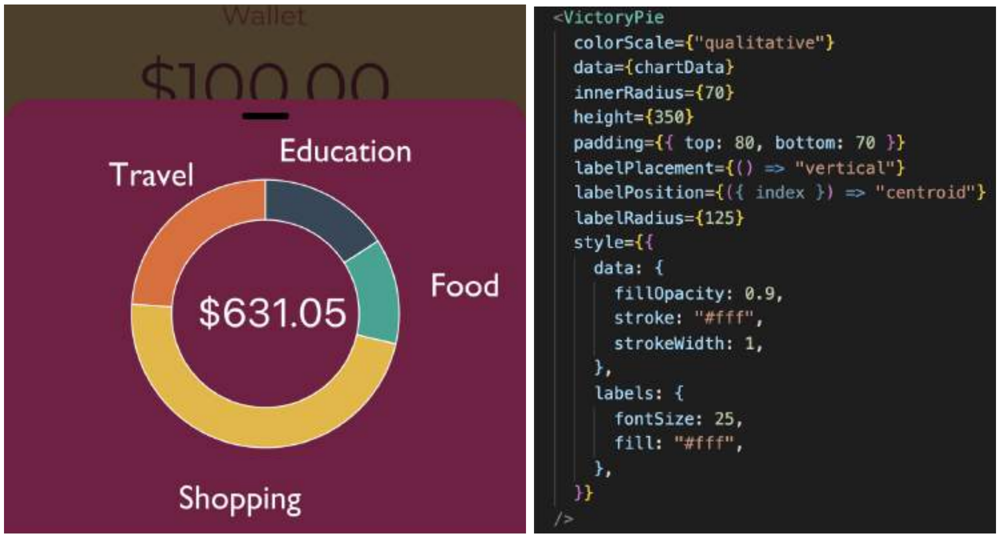


```
<RBSheet
  ref={refRBSheet}
  closeOnDragDown={true}
  closeOnPressMask={true}
  height={650}
  customStyles={{
    wrapper: {
      backgroundColor: "rgba(0, 0, 0, 0.7)",
    },
    container: {
      borderRadius: 20,
      backgroundColor: "#6E2142",
    },
    draggableIcon: {
      backgroundColor: "#000",
    },
  }}
>
  <BudgetGraph></BudgetGraph>
</RBSheet>
```

Figure 5.10. Bottom Sheet component

Bottom sheet is another way to display additional information when the user interacts with a button or slides up from the bottom of the phone. For our app, we used <RBSheet> from react-native-bottom-sheet library to implement this function. The components allow us to give different styles such as color, border radius and height of the sheet. In the above figure, a <BudgetGraph> component was rendered inside the <RBSheet>. This <BudgetGraph> component contains all the necessary information for users to view their budget details.

5.2.9. Victory Pie

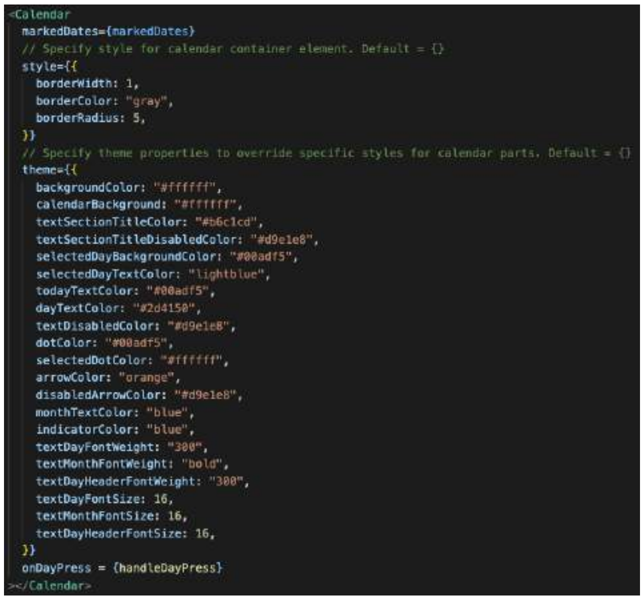


```
<VictoryPie
  colorScale={"qualitative"}
  data={chartData}
  innerRadius={70}
  height={350}
  padding={{ top: 80, bottom: 70 }}
  labelPlacement={() => "vertical"}
  labelPosition={({ index }) => "centroid"}
  labelRadius={125}
  style={{
    data: {
      fillOpacity: 0.9,
      stroke: "#fff",
      strokeWidth: 1,
    },
    labels: {
      fontSize: 25,
      fill: "#fff",
    },
  }}
/>
```

Figure 5.11. Victory Pie Component

In order to create a donut chart, <VictoryPie> component was brought in from the Victory library. Essentially, it looks like a normal donut chart where users can see how much they spent on each category. The component provides many properties for us to style according to the way we designed. In the above figure, we pass our budget data into the “data” property inside the component and it will be generated accordingly. Other properties are for styling and designing the donut chart.

5.2.10. Calendar

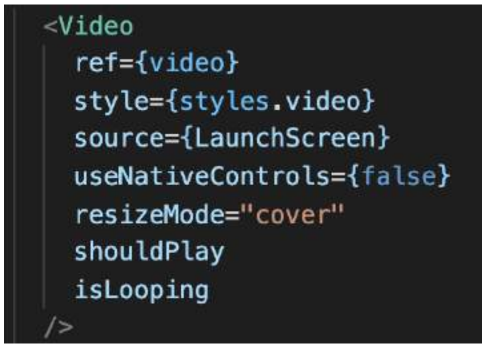


```
<Calendar
  markedDates={markedDates}
  // Specify style for calendar container element. Default = {}
  style={{
    borderWidth: 1,
    borderColor: "gray",
    borderRadius: 5,
  }}
  // Specify theme properties to override specific styles for calendar parts. Default = {}
  theme={{
    backgroundColor: "#ffffff",
    calendarBackground: "#ffffff",
    textSectionTitleColor: "#b6c1cd",
    textSectionTitleDisabledColor: "#d9e1e8",
    selectedDayBackgroundColor: "#00adf5",
    selectedDayTextColor: "lightblue",
    todayTextColor: "#00adf5",
    dayTextColor: "#2d4150",
    textDisabledColor: "#d9e1e8",
    dotColor: "#00adf5",
    selectedDotColor: "#ffffff",
    arrowColor: "orange",
    disabledArrowColor: "#d9e1e8",
    monthTextColor: "blue",
    indicatorColor: "blue",
    textDayFontWeight: "300",
    textMonthFontWeight: "bold",
    textDayHeaderFontWeight: "300",
    textDayFontSize: 16,
    textMonthFontSize: 16,
    textDayHeaderFontSize: 16,
  }}
  onDayPress = {handleDayPress}
></Calendar>
```

Figure 5.12. Calendar Component

The react-native-calendars library allows us to create a calendar by giving us a <Calendar> component that we can easily customize. We have to pass values like color, font size, and overall format, but it is significantly easier than trying to create our own calendar component from scratch.

5.2.11. Expo-AV

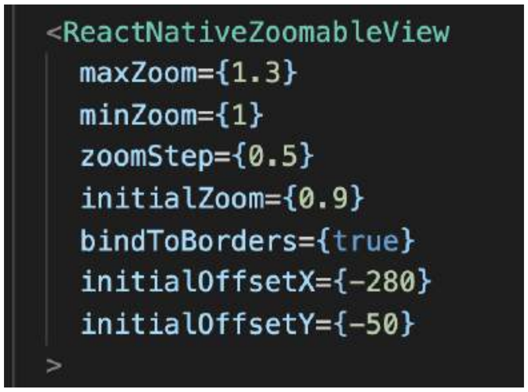


```
<Video
  ref={video}
  style={styles.video}
  source={LaunchScreen}
  useNativeControls={false}
  resizeMode="cover"
  shouldPlay
  isLooping
/>
```

Figure 5.13. Video Component

We developed our react native application by initialising an Expo project file. Hence, we were able to use all the libraries provided by Expo in our project. The Expo-AV library allowed us to implement and integrate audio and video into the app. In our project, <Video> component was used for displaying a short animation in our splash screen. We passed our video file into the “source” property so that Expo would be able to render accordingly. We also removed the media controllers as we do not want our user to be able to play and pause the animation in our splash screen. To ensure that the video was sized according to different display screens, we passed in “styles.video” into the style property so that Expo would render our video player according to the design specification written in the StyleSheet component.

5.2.12. Zoomable View



```
<ReactNativeZoomableView
  maxZoom={1.3}
  minZoom={1}
  zoomStep={0.5}
  initialZoom={0.9}
  bindToBorders={true}
  initialOffsetX={-280}
  initialOffsetY={-50}
>
```

Figure 5.14. Zoomable View Component

For our home screen, we provided a way for users to zoom in and out to see their furniture and also the surroundings better. In order to implement this, <ReactNativeZoomableView> component was imported into the project. This component allowed us to set the minimum and maximum zoom when the user interacts with the screen. We can also set the initial position and degree of zoom when the user first enters into the room from the group select screen. Other properties that the library provided such as zoom sensitively were not implemented in this case.

5.3. Back End Development

5.3.1. Database

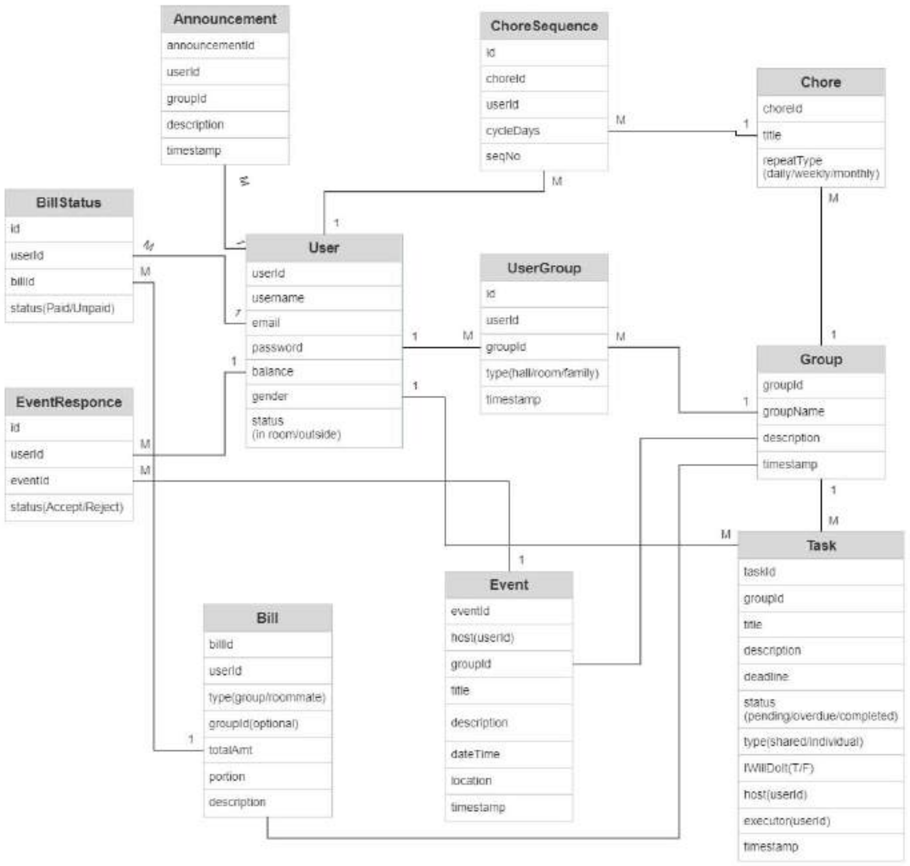
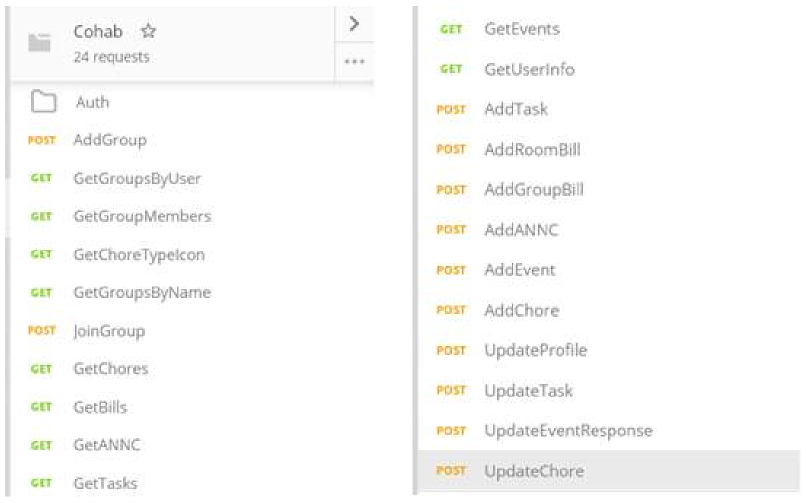


Figure 5.15. Entity Relationship Diagram for database description

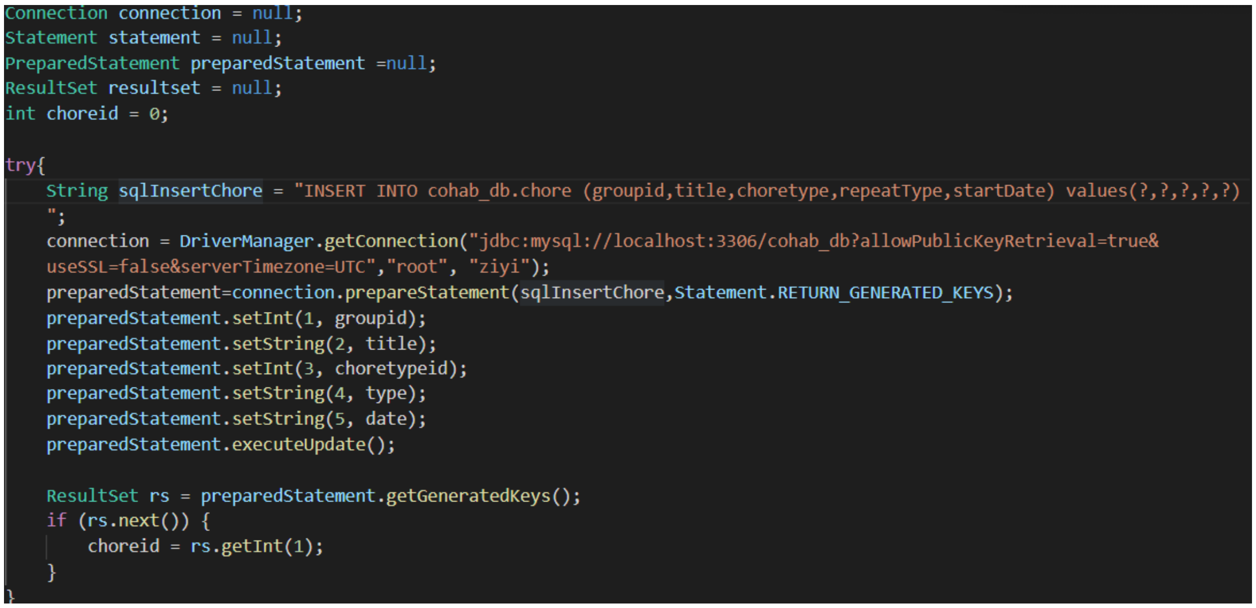
5.3.2. APIs

- Includes 24 requests to be linked with the screens
- Tested via Postman
- Code by using JavaServlet



```
Cohab ☆
24 requests
Auth
POST AddGroup
GET  GetGroupsByUser
GET  GetGroupMembers
GET  GetChoreTypeIcon
GET  GetGroupsByName
POST JoinGroup
GET  GetChores
GET  GetBills
GET  GetANNC
GET  GetTasks
GET  GetEvents
GET  GetUserInfo
POST AddTask
POST AddRoomBill
POST AddGroupBill
POST AddANNC
POST AddEvent
POST AddChore
POST UpdateProfile
POST UpdateTask
POST UpdateEventResponse
POST UpdateChore
```

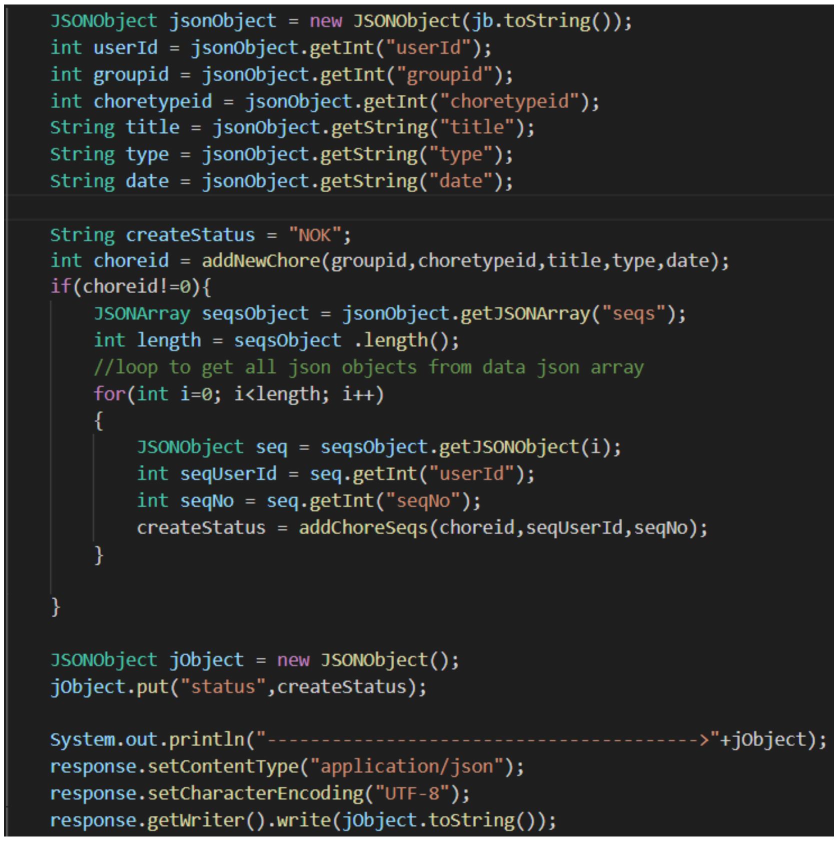
Figure 5.16. List of API requests



```java
Connection connection = null;
Statement statement = null;
PreparedStatement preparedStatement =null;
ResultSet resultset = null;
int choreid = 0;

try{
    String sqlInsertChore = "INSERT INTO cohab_db.chore (groupid,title,choretype,repeatType,startDate) values(?,?,?,?,?)
    ";
    connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/cohab_db?allowPublicKeyRetrieval=true&
    useSSL=false&serverTimezone=UTC","root", "ziyi");
    preparedStatement=connection.prepareStatement(sqlInsertChore,Statement.RETURN_GENERATED_KEYS);
    preparedStatement.setInt(1, groupid);
    preparedStatement.setString(2, title);
    preparedStatement.setInt(3, choretypeid);
    preparedStatement.setString(4, type);
    preparedStatement.setString(5, date);
    preparedStatement.executeUpdate();

    ResultSet rs = preparedStatement.getGeneratedKeys();
    if (rs.next()) {
        choreid = rs.getInt(1);
    }
}
```

Figure 5.17. Example of SQL connection and query to create a new chore for Chores Screen

- Use of PreparedStatement for preventing SQL Injection vulnerabilities



```
JSONObject jsonObject = new JSONObject(jb.toString());
int userId = jsonObject.getInt("userId");
int groupid = jsonObject.getInt("groupid");
int choretypeid = jsonObject.getInt("choretypeid");
String title = jsonObject.getString("title");
String type = jsonObject.getString("type");
String date = jsonObject.getString("date");

String createStatus = "NOK";
int choreid = addNewChore(groupid,choretypeid,title,type,date);
if(choreid!=0){
    JSONArray seqsObject = jsonObject.getJSONArray("seqs");
    int length = seqsObject .length();
    //loop to get all json objects from data json array
    for(int i=0; i<length; i++)
    {
        JSONObject seq = seqsObject.getJSONObject(i);
        int seqUserId = seq.getInt("userId");
        int seqNo = seq.getInt("seqNo");
        createStatus = addChoreSeqs(choreid,seqUserId,seqNo);
    }

}

JSONObject jObject = new JSONObject();
jObject.put("status",createStatus);

System.out.println("---------------------------------------->"+jObject);
response.setContentType("application/json");
response.setCharacterEncoding("UTF-8");
response.getWriter().write(jObject.toString());
```

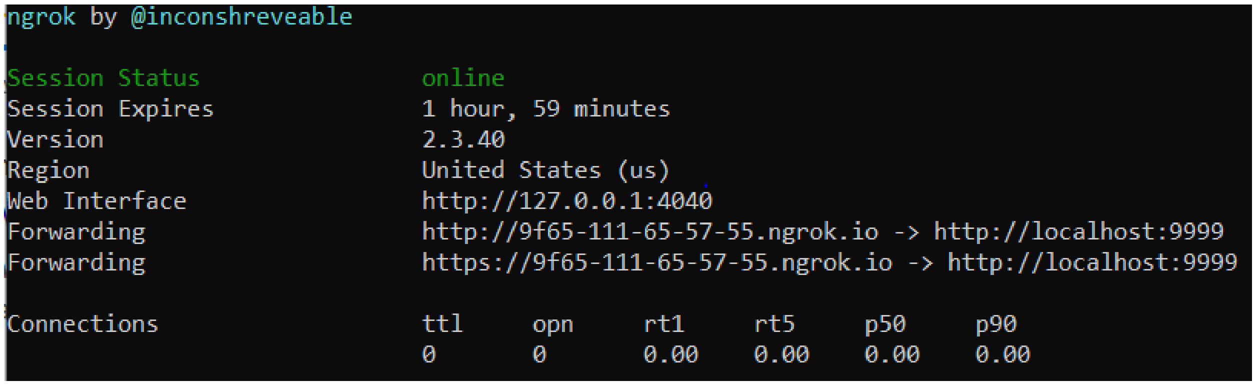
Figure 5.18. Retrieving posted JSON data and return result when adding a new chore for Chores Screen

- createStatus for responding insert result

5.3.3. Proxy

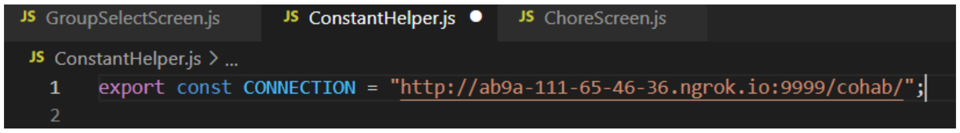
In computer networking, a proxy server is a server application that acts as an intermediary between a client requesting a resource and the server providing that resource.

That makes it easy to access the server remotely.



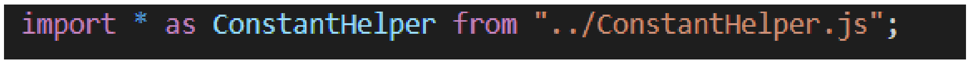
```
ngrok by @inconshreveable

Session Status                online
Session Expires               1 hour, 59 minutes
Version                       2.3.40
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://9f65-111-65-57-55.ngrok.io -> http://localhost:9999
Forwarding                    https://9f65-111-65-57-55.ngrok.io -> http://localhost:9999

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```

Figure 5.19. Ngrok Proxy



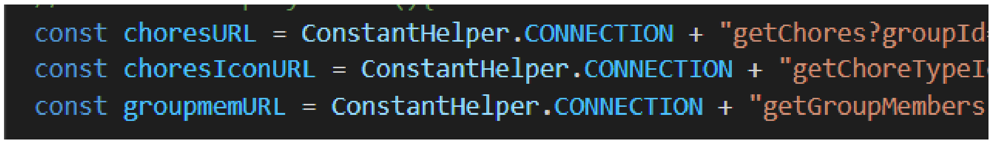
```
JS GroupSelectScreen.js    JS ConstantHelper.js ●    JS ChoreScreen.js
JS ConstantHelper.js > ...
export const CONNECTION = "http://ab9a-111-65-46-36.ngrok.io:9999/cohab/";
```

Figure 5.20. Current connection link



```
import * as ConstantHelper from "../ConstantHelper.js";
```

Figure 5.21. Use of global variable in a screen

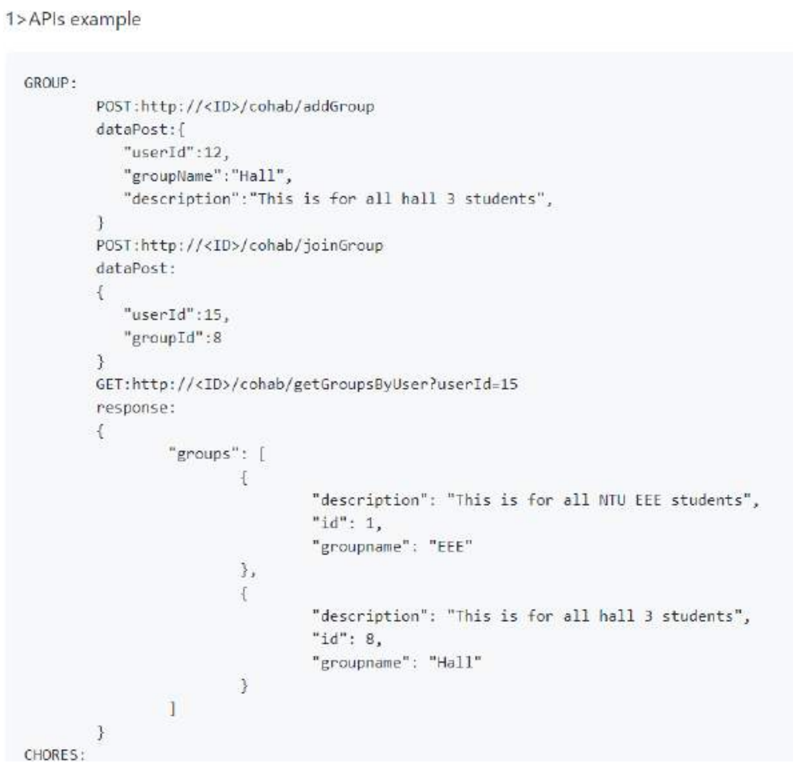


```
const choresURL = ConstantHelper.CONNECTION + "getChores?groupId
const choresIconURL = ConstantHelper.CONNECTION + "getChoreTypeI
const groupmemURL = ConstantHelper.CONNECTION + "getGroupMembers
```

Figure 5.22. Example of how the global variable simplifies link connection implementation

5.3.4. Integration with Front End

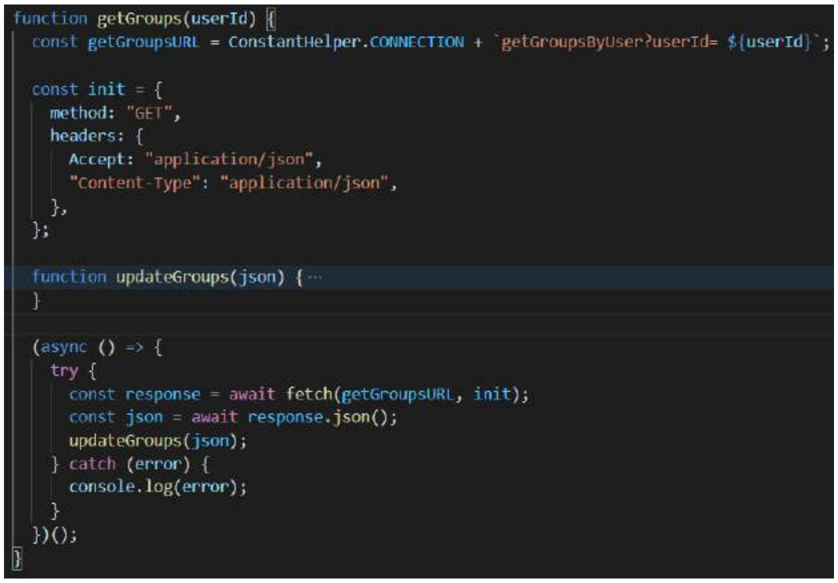
- RESTful APIs
- Document for APIs



1>APIs example

```
GROUP:
        POST:http://<ID>/cohab/addGroup
        dataPost:{
           "userId":12,
           "groupName":"Hall",
           "description":"This is for all hall 3 students",
        }
        POST:http://<ID>/cohab/joinGroup
        dataPost:
        {
           "userId":15,
           "groupId":8
        }
        GET:http://<ID>/cohab/getGroupsByUser?userId=15
        response:
        {
                "groups": [
                        {
                                "description": "This is for all NTU EEE students",
                                "id": 1,
                                "groupname": "EEE"
                        },
                        {
                                "description": "This is for all hall 3 students",
                                "id": 8,
                                "groupname": "Hall"
                        }
                ]
        }
CHORES:
```

Figure 5.23. An example of the “Group” API



```
function getGroups(userId) {
  const getGroupsURL = ConstantHelper.CONNECTION + `getGroupsByUser?userId= ${userId}`;

  const init = {
    method: "GET",
    headers: {
      Accept: "application/json",
      "Content-Type": "application/json",
    },
  };

  function updateGroups(json) {…
  }

  (async () => {
    try {
      const response = await fetch(getGroupsURL, init);
      const json = await response.json();
      updateGroups(json);
    } catch (error) {
      console.log(error);
    }
  })();
}
```

Figure 5.24. Calling API “getGroups” to fetch the JSON data

Section 6: Conclusion and Recommendation

6.1. Challenges

Organizing a group of 11

Having a large group is both a blessing and a curse. As a group of 11 members, we had many ideas about what our app should be like, but obviously could not implement every idea, no matter how brilliant. Even from the start, it took us a while to agree on what kind of app we were going to design. Everyone came up with their own ideas, and picking just one was almost impossible.

However, we tried our best to communicate and compromise, and eventually settled on voting for a single idea. Voting proved to be an effective strategy for us, even further down the road when working on the app itself. Although we would all be involved in discussions, when we eventually had to make a choice about something, simply putting it to a vote tended to work quickly and efficiently.

React Native

Another issue we faced early on was selecting a suitable framework to develop our app. We quickly settled on using React Native for the bulk of our app, but that posed its own problems. A few of us had experience with React or React Native, but the vast majority did not. This caused an issue early on, when everyone was still beginning to get comfortable with the framework. The problem was exacerbated by the short timeframe we had to develop working prototypes for our presentations.

Thankfully, those of us who had more experience programming were able to quickly teach the rest and help them get up to speed. The problem eventually resolved itself after a few weeks as we all became more familiar with React Native and its quirks.

Github

We faced a similar issue when learning how to use Git and Github. Again, some of us had experience with it, but to most people, it was extremely confusing.

The text-based interface of the Git GUI did not help either, producing enigmatic error messages that were difficult for the layperson to understand. Even those of us who had experience using Github were stumped at times, since we generally never had to manage a large project with many people who were still getting used to Github. A common issue was with merging commits, where multiple people would try to commit at the same time, resulting in an error for one of them.

Fortunately, we had known that we were supposed to use Github as part of this module, and started learning and teaching Github as early as our first meeting. It turned out to be vitally important that we started using Github early on, since we could help members that were confused, and made sure that nobody made any mistakes later on, when the Github repository got more complicated.

Blender

Blender was a challenge to learn, due to having to learn the functions of the workspaces, each with specific purposes and capabilities, and the properties listed in the 14 tabs and panels. Within the 3D viewport, we needed to familiarise ourselves with the object mode for placing objects within the scene, the edit mode for editing individual objects, and pose mode for adding animation keyframes. In order to complete the assets quickly, it was necessary to learn the keyboard shortcuts as well. In the beginning, learning how to edit the models using the vertice and face selection modes was challenging as in order to extrude, move, rotate, or scale the selection in the correct way, the origin, axes, and snap-to properties had to be properly defined, which we were unaware of. When the assets were completed, we then encountered difficulties obtaining the right images to implement in the app, due to the overlapping of certain objects. Hence some objects had to be rendered with the holdout property applied.

Implementing 3D

The biggest issue we faced was somewhere in the middle of our project, when we realized that we could not implement the 3D models that we had originally planned to. Our original plan was to design models in Blender and import them into our React Native app, ideally with a 3rd-party library that could do all the heavy lifting for us. Those plans fell through, however, when we realized that the 3rd-party library we had planned to use lacked certain key features we needed.

We considered a few different approaches, and settled on a workable 2.5D idea. We had already started designing in Blender, and did not want that work to go to waste, even if we could not execute our original idea perfectly. Our new idea involved taking screenshots of the 3D objects we designed and adding those screenshots to our app, using a bit of inventive coding to make them still seem as 3D as we could. Although we were disappointed that we could not execute our original idea, it was incredibly fulfilling to come up with a working solution on the fly.

6.2. Conclusion

In conclusion, CoHab serves as a one-stop platform for all things related to communal living. It provides convenience for cohabitants by allowing them to organise household and financial matters or life events in a centralised application through the various functions such as chores, task, finance and calendar.

6.3. Recommendation for Future Works

Non-player Character (NPC)

As each user has a unique username, a personalised NPC could be created for each user. Based on the GPS location of the user or whether they have checked into their house on CoHab, an NPC would appear or disappear on the home screen. So for example, if a user has checked into their home, their NPC character would appear on the home screen and could possibly be interacting with the furniture in the room, such as sitting on the sofa etc.

Greater customization features for the furniture

Currently, our customization involves allowing users to choose which furniture they want to add to their room and also assign a particular function to it. We have more than enough furniture for users to choose from to build their room since we only have 4 functions available at the moment. However, this will not pose a problem if we continue to introduce more functions that can help roommates live together more efficiently. In the future, we will like to allow users to be able to set different colors for their furniture and also be able to rearrange the furniture in the room, hence this will intensify their user experience on using CoHab.

Artificial Intelligence (AI) Avatar

At present, our avatar dialogue is based on hardcoded text. To make our avatar, Carl, more interactive, we would be interested in introducing AI to Carl such that each user can have a unique experience on CoHab. Each user would be able to talk with Carl and Carl would be able to understand and generate suitable responses for a two-way conversation.

Gamification

Gamification could be applied to further engage with users. We could use Unity 2D to create mini games for users to play within CoHab. Users may be able to customise their own avatar for a sense of personalisation. A reward system based on completed tasks or chores may be implemented within CoHab to encourage frequent usage and motivate each user to carry out their assigned chores or tasks. The awarded points from completing tasks or chores could then be converted to CoHab's game currency which allows users to "buy" in-game items or unlock more customisable outfits for their avatar.

Link finance features with PayLah!/PayNow and enable Scan-to-Pay

Currently, the application is able to allow users to update their wallet balance without deducting any real credits. In the future, we are still very much interested in bringing these features into the app so that users would not need to switch apps to pay for their expenses or food, especially for students living in the universities' halls.

Appendix

A. Design Diagrams

A.1. Use Case Diagram

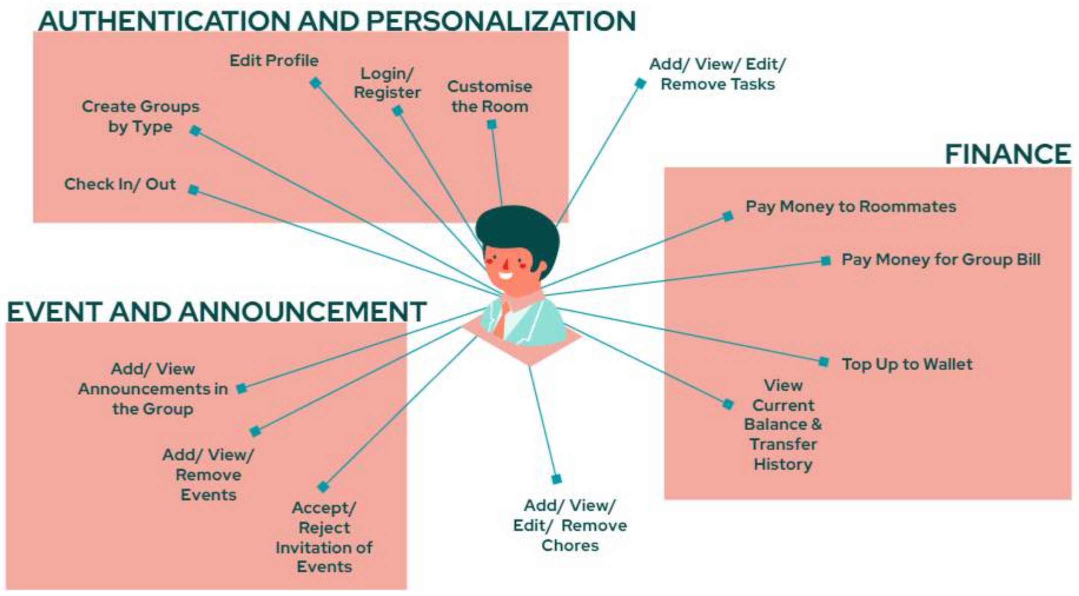

AUTHENTICATION AND PERSONALIZATION
Edit Profile
Login/ Register
Customise the Room
Create Groups by Type
Check In/ Out
Add/ View/ Edit/ Remove Tasks
FINANCE
Pay Money to Roommates
Pay Money for Group Bill
Top Up to Wallet
View Current Balance & Transfer History
EVENT AND ANNOUNCEMENT
Add/ View Announcements in the Group
Add/ View/ Remove Events
Accept/ Reject Invitation of Events
Add/ View/ Edit/ Remove Chores

Figure A.1. Use case diagram of CoHab

A.2. PAN Diagram

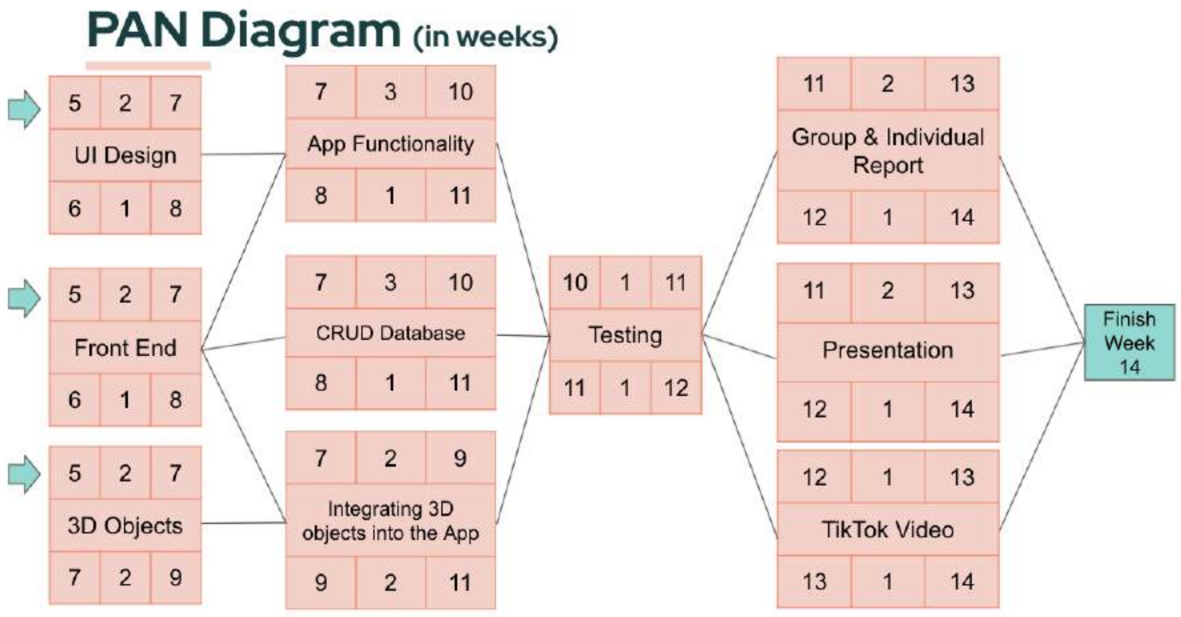
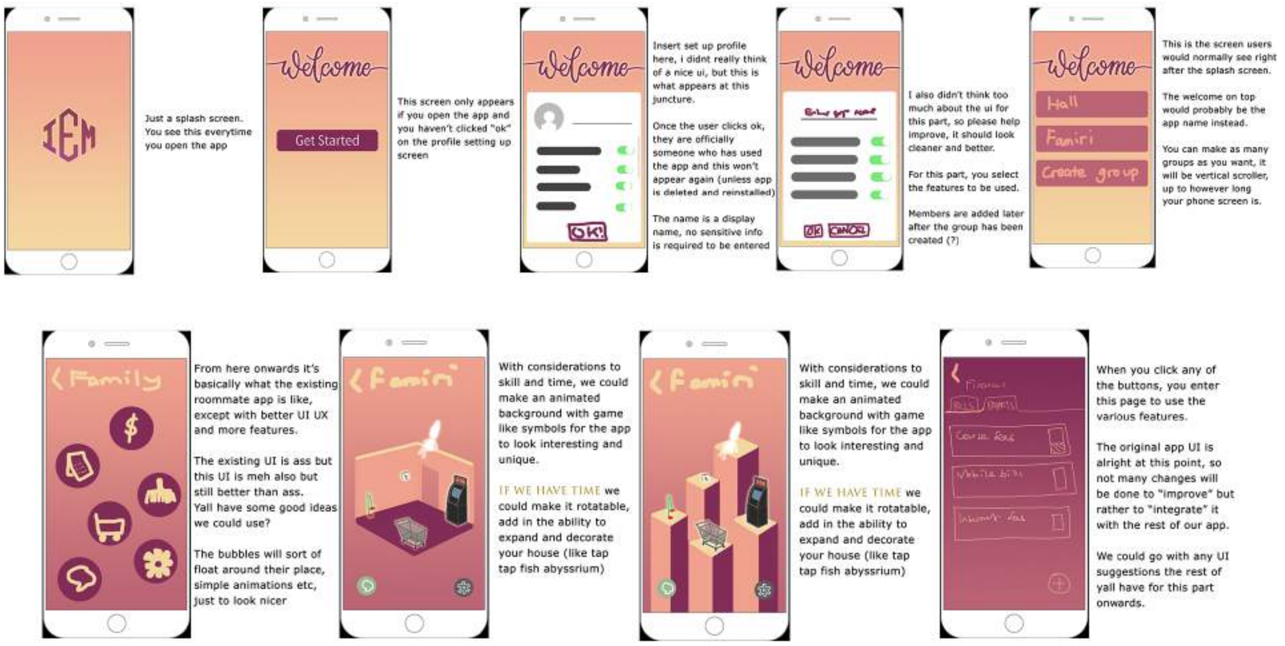

PAN Diagram (in weeks)
5 2 7 UI Design 6 1 8
5 2 7 Front End 6 1 8
5 2 7 3D Objects 7 2 9
7 3 10 App Functionality 8 1 11
7 3 10 CRUD Database 8 1 11
7 2 9 Integrating 3D objects into the App 9 2 11
10 1 11 Testing 11 1 12
11 2 13 Group & Individual Report 12 1 14
11 2 13 Presentation 12 1 14
12 1 13 TikTok Video 13 1 14
Finish Week 14

Figure A.2. PAN diagram shown during Week 6 presentation

A.3. Wireframing Sketches



Just a splash screen. You see this everytime you open the app

This screen only appears if you open the app and you haven't clicked "ok" on the profile setting up screen

Insert set up profile here, i didnt really think of a nice ui, but this is what appears at this juncture.

Once the user clicks ok, they are officially someone who has used the app and this won't appear again (unless app is deleted and reinstalled)

The name is a display name, no sensitive info is required to be entered

I also didn't think too much about the ui for this part, so please help improve, it should look cleaner and better.

For this part, you select the features to be used.

Members are added later after the group has been created (?)

This is the screen users would normally see right after the splash screen.

The welcome on top would probably be the app name instead.

You can make as many groups as you want, it will be vertical scroller, up to however long your phone screen is.

From here onwards it's basically what the existing roommate app is like, except with better UI UX and more features.

The existing UI is ass but this UI is meh also but still better than ass. Yall have some good ideas we could use?

The bubbles will sort of float around their place, simple animations etc, just to look nicer

With considerations to skill and time, we could make an animated background with game like symbols for the app to look interesting and unique.

IF WE HAVE TIME we could make it rotatable, add in the ability to expand and decorate your house (like tap tap fish abyssrium)

With considerations to skill and time, we could make an animated background with game like symbols for the app to look interesting and unique.

IF WE HAVE TIME we could make it rotatable, add in the ability to expand and decorate your house (like tap tap fish abyssrium)

When you click any of the buttons, you enter this page to use the various features.

The original app UI is alright at this point, so not many changes will be done to "improve" but rather to "integrate" it with the rest of our app.

We could go with any UI suggestions the rest of yall have for this part onwards.

Figure A.3. User interface sketches for group and home screen

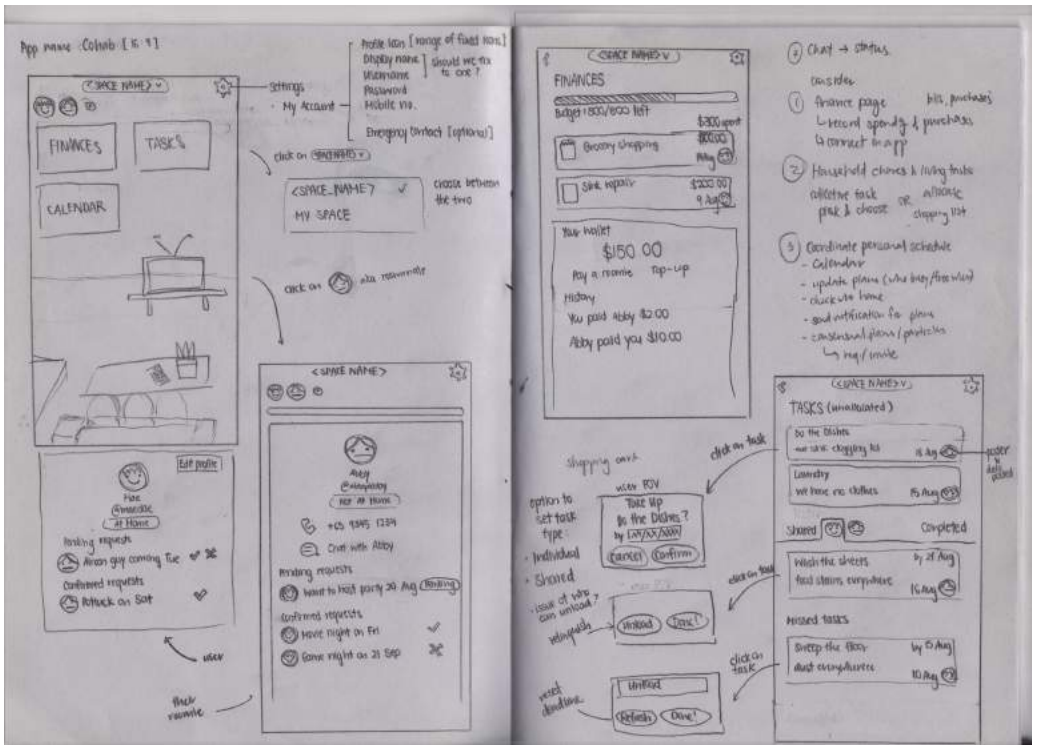


Figure A.4. User interface sketches for home, finance and task screen

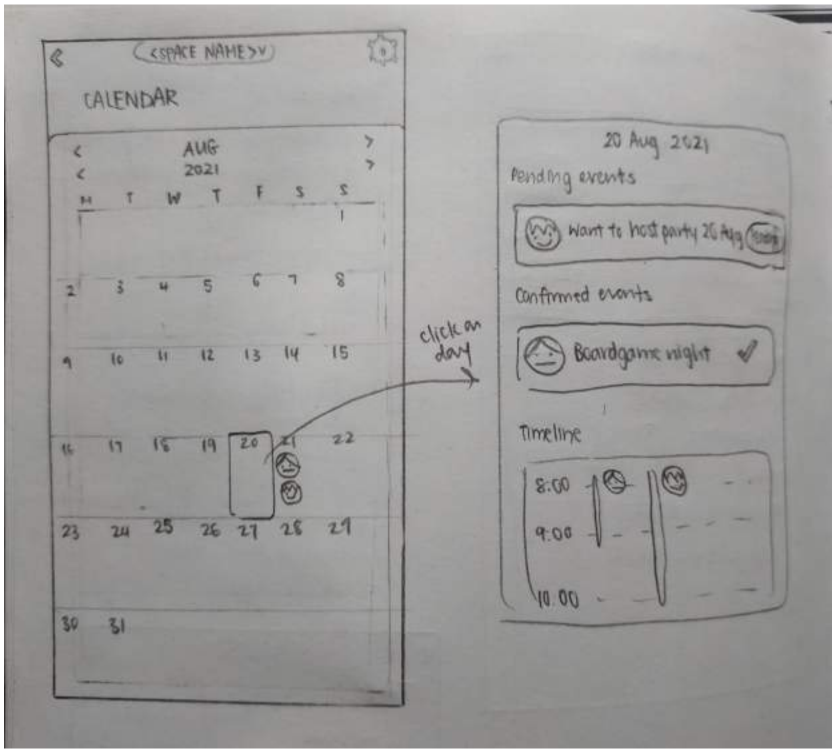


Figure A.5. User interface sketch for calendar screen

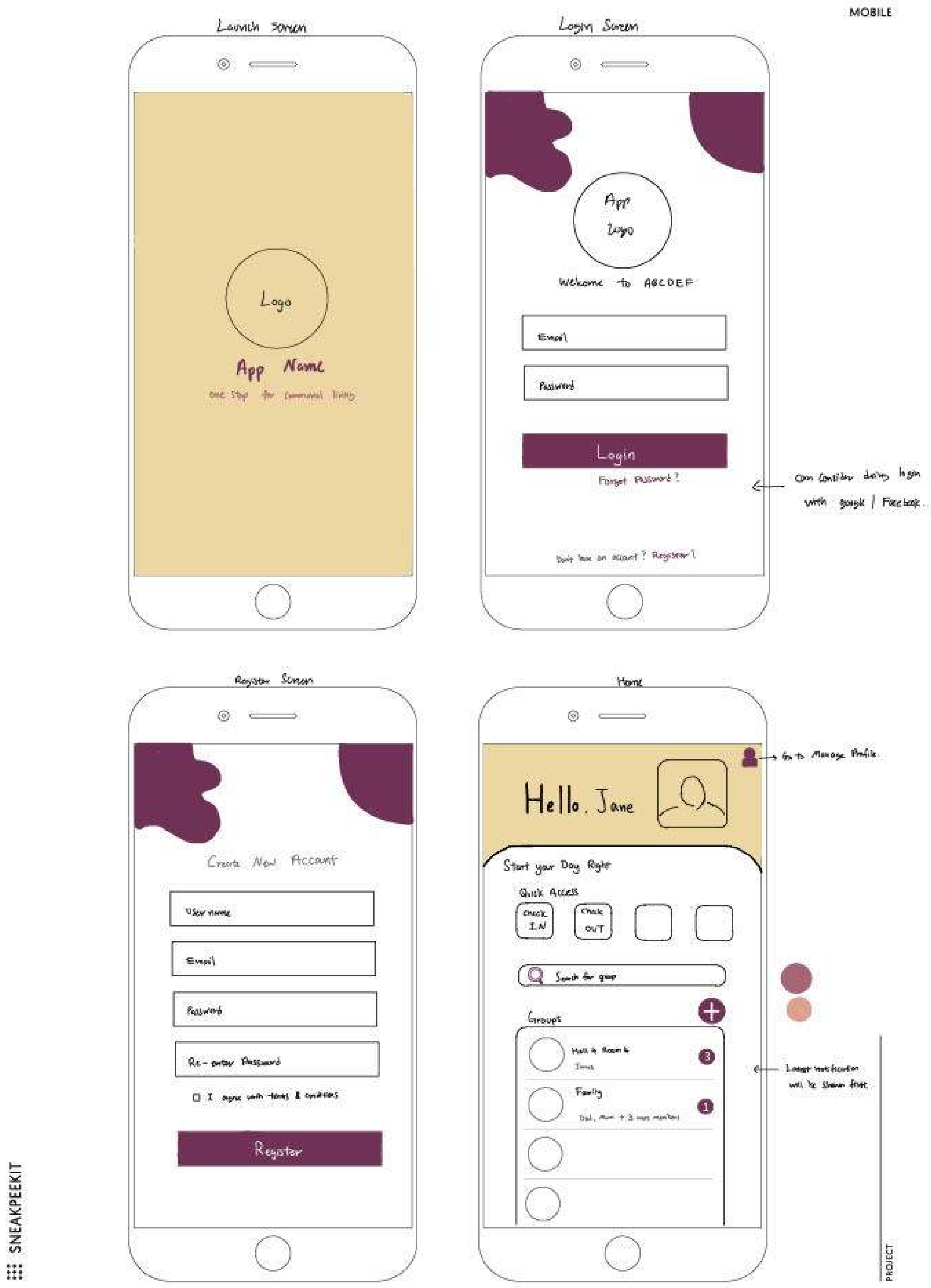


Figure A.6. User interface sketches for launch, login, register and home screen

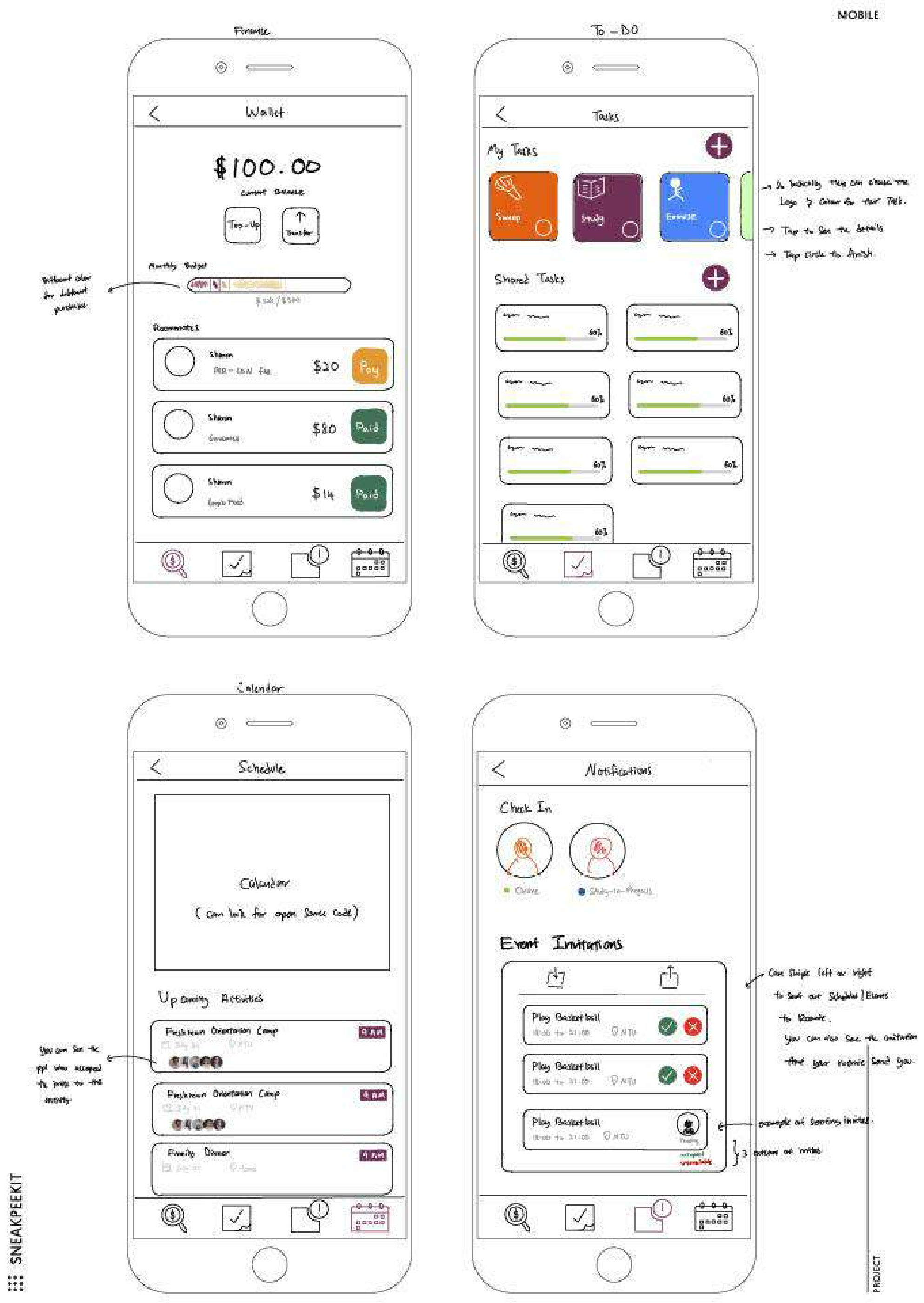


Figure A.7. User interface sketch for finance, tasks, calendar and notifications screen

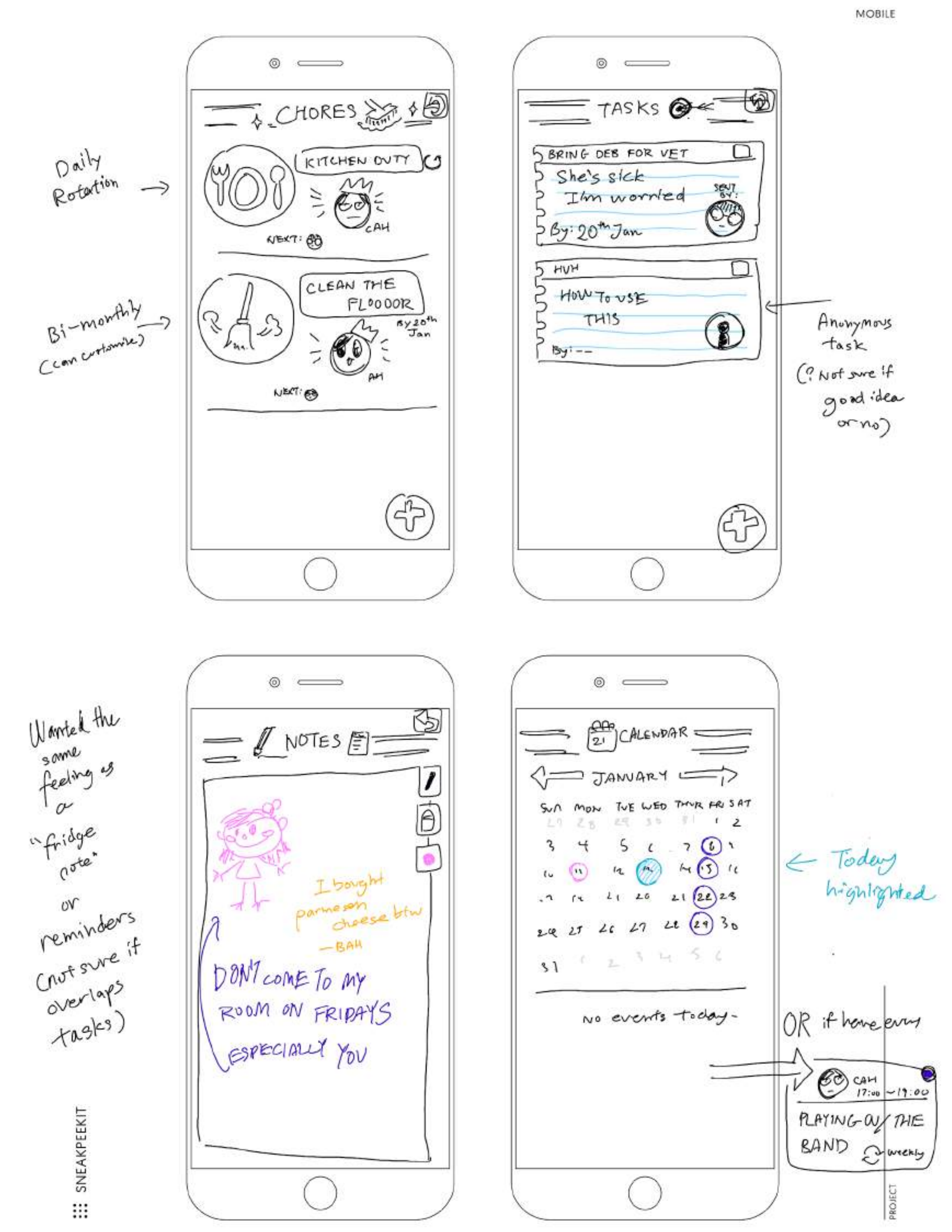


Figure A.8. User interface sketch for chores, tasks, calendar and an unimplemented “notes” screen

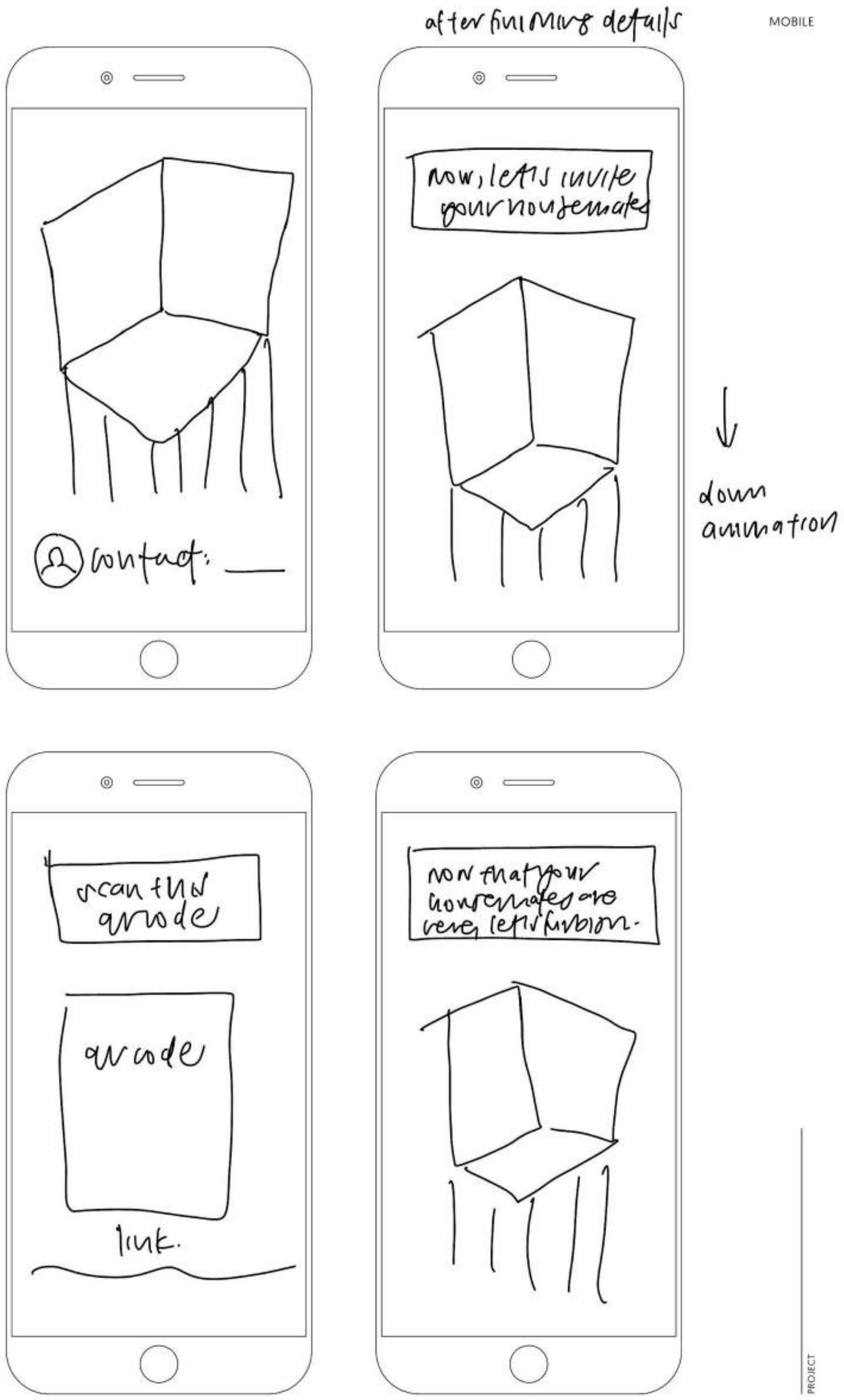

after finishing details
MOBILE
contact: ___
now, let's invite your housemates
down animation
scan the qrcode
qrcode
link.
now that your housemates are here, let's finish.

Figure A.9. User interface sketch for the flow of setting up the home page

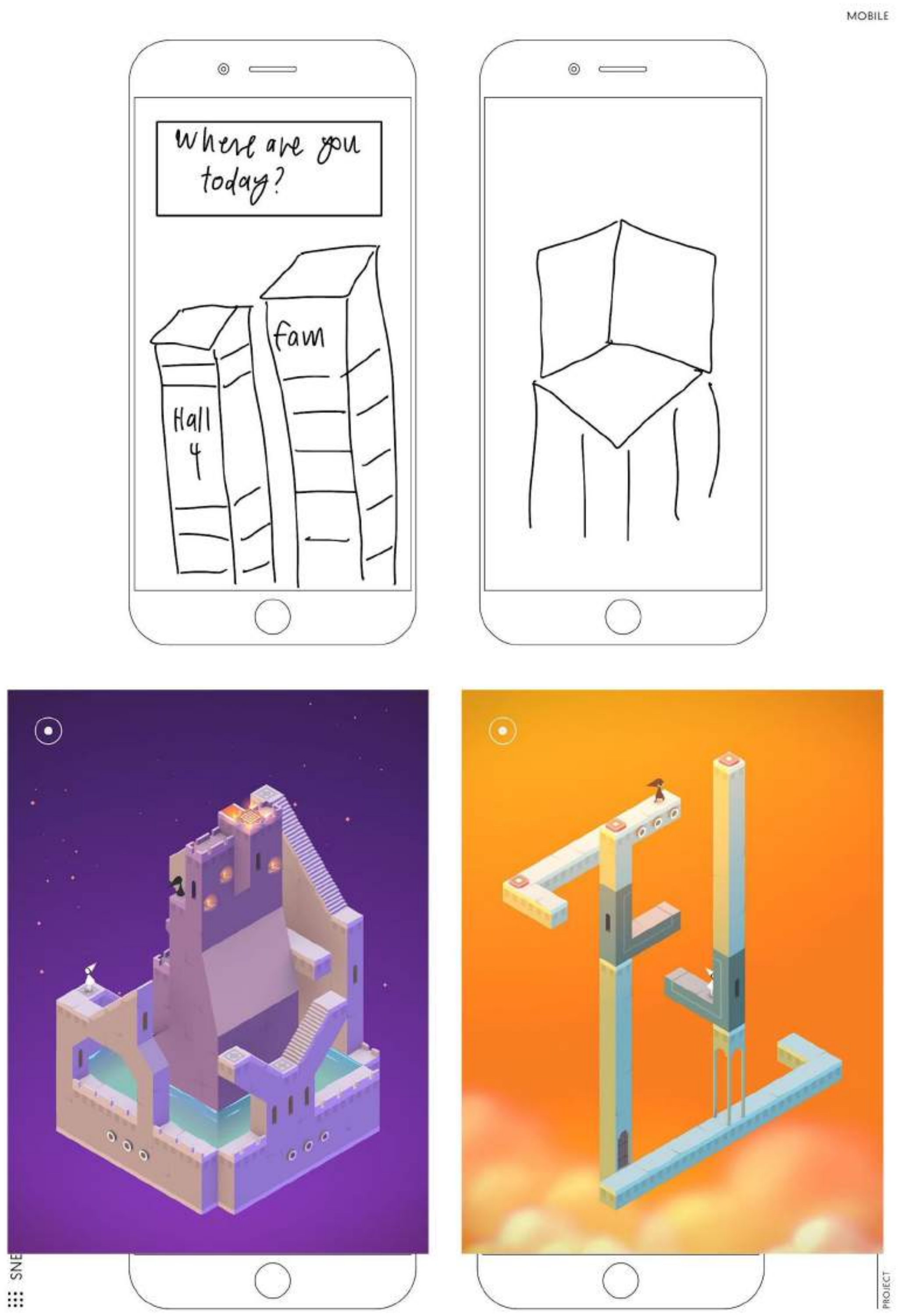


Figure A.10. User interface sketch for group and home screen, and a screenshot from a game

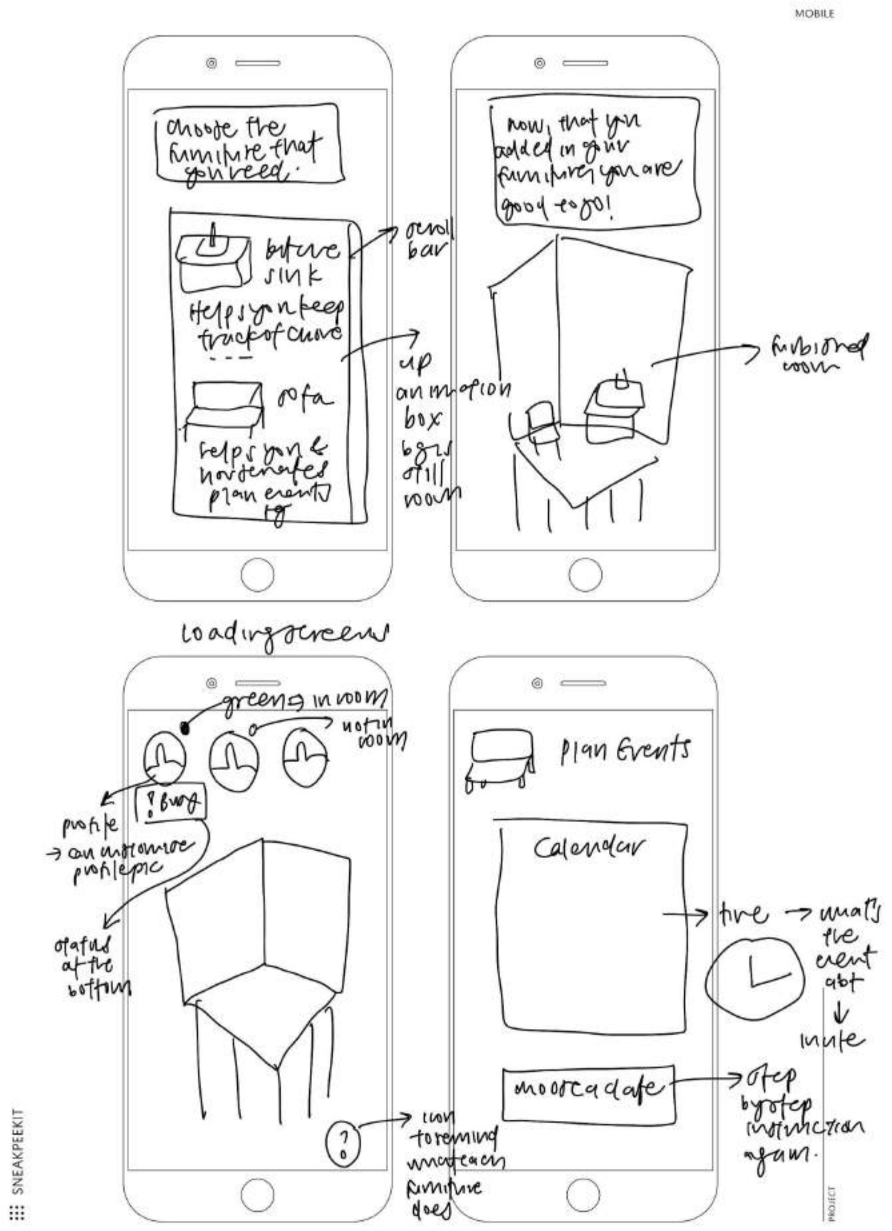
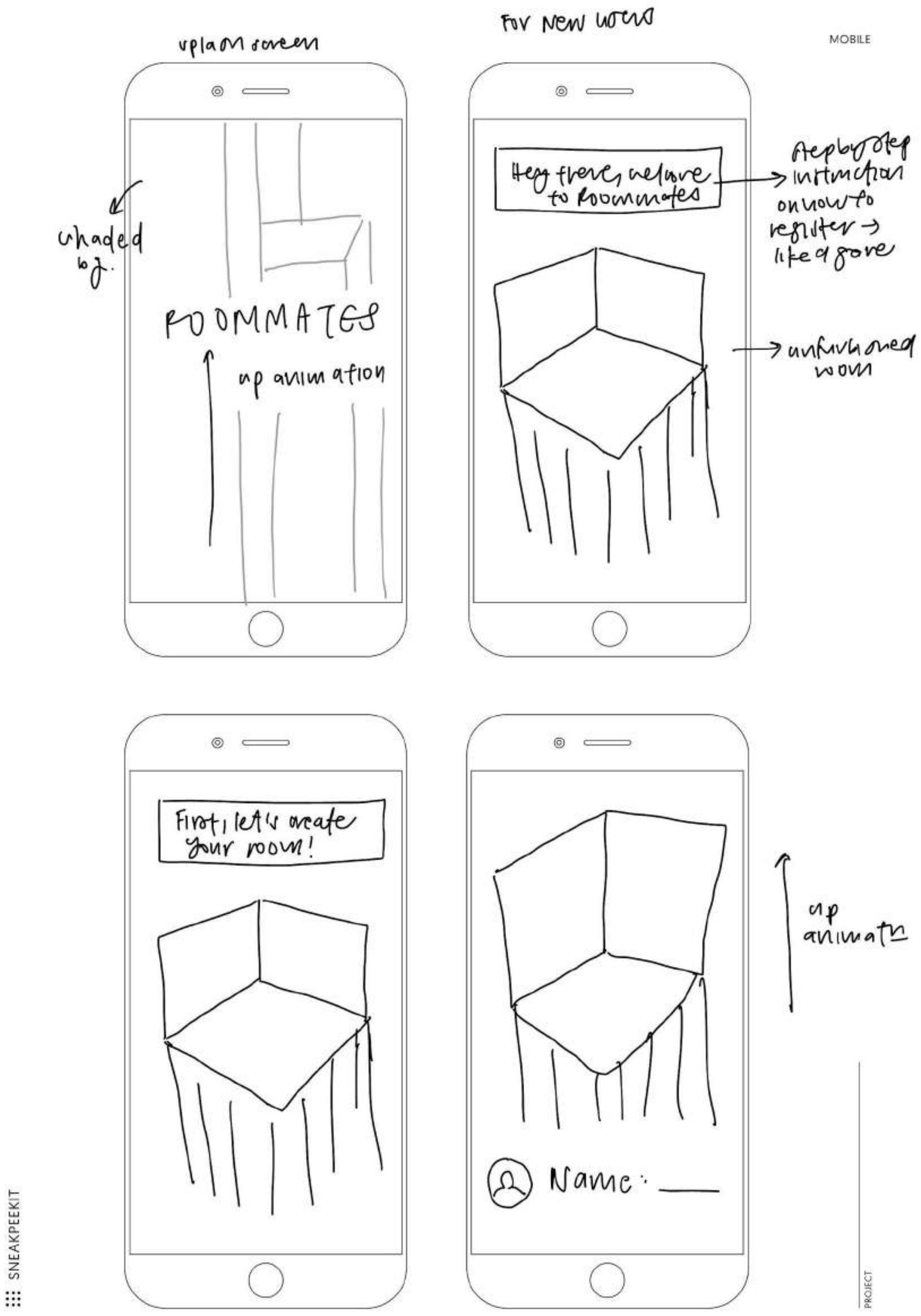

MOBILE
choose the furniture that you need.
bathe sink
Helps you keep track of chore
sofa
helps you & housemates plan events
scroll bar
up animation box
bgs still room
now, that you added in your furniture you are good to go!
furnished room
loading screen
green → in room
not in room
! busy
profile
→ can customise profile pic
status at the bottom
icon to remind what each furniture does
Plan Events
Calendar
time → what's the event abt
↓ invite
choose a date
→ step by step instruction again.
SNEAKPEEKIT
PROJECT

Figure A.11. User interface sketch for setting up the home screen



splash screen
For new users
MOBILE
shaded bg.
ROOMMATES
up animation
Hey there, welcome to Roommates
step by step instruction on how to register → like a game
unfurnished room
First, let's create your room!
up animatn
Name: ____
SNEAKPEEKIT
PROJECT

Figure A.12. User interface sketch for potential animations on home screen

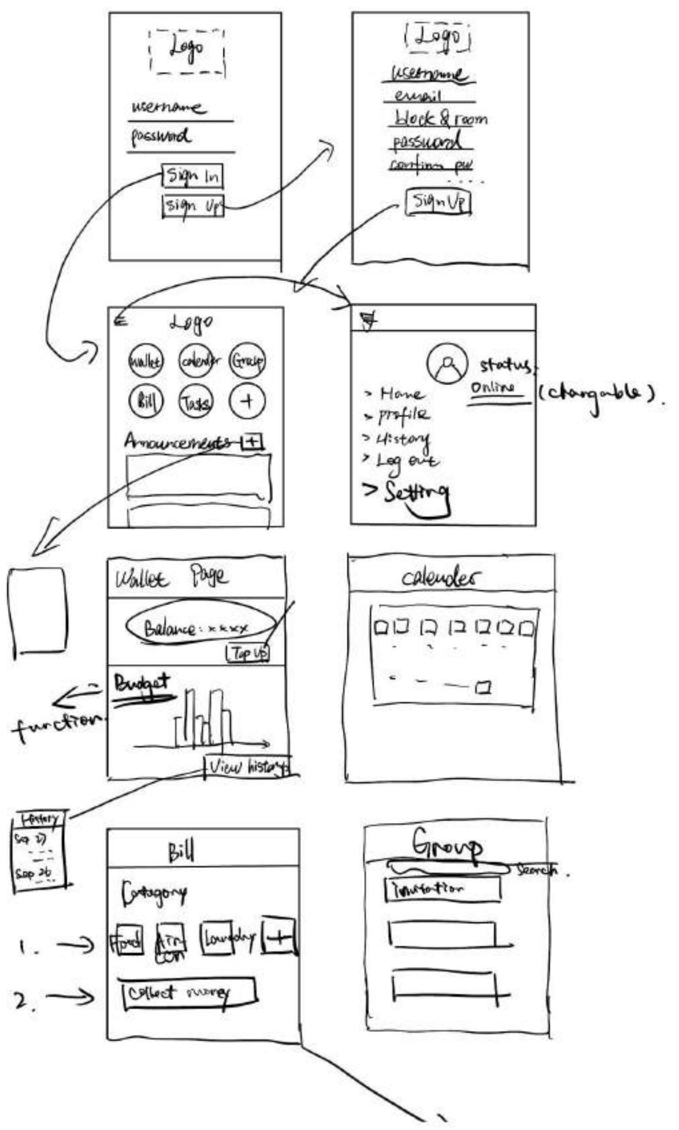


Figure A.13. User-interface sketch for navigation flow, login, signup, finance, calendar, group and home screens

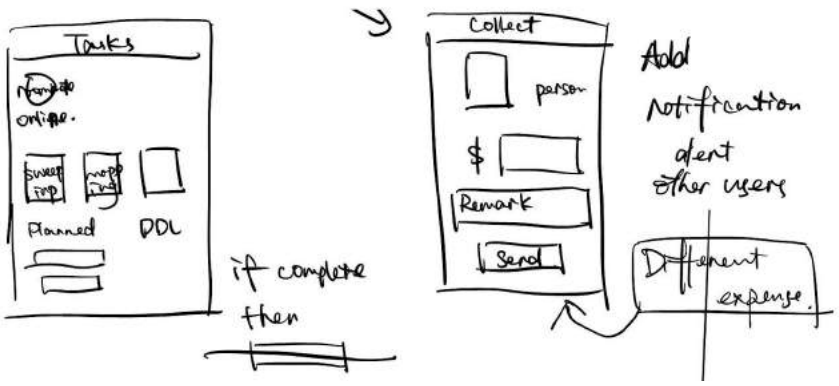
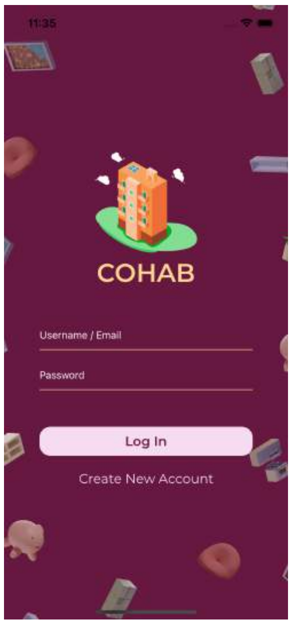
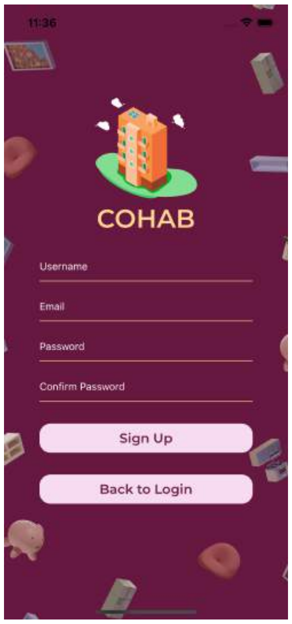
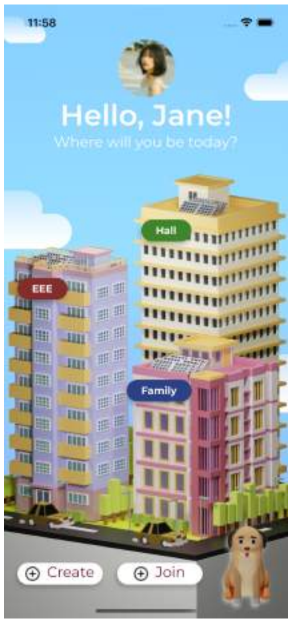
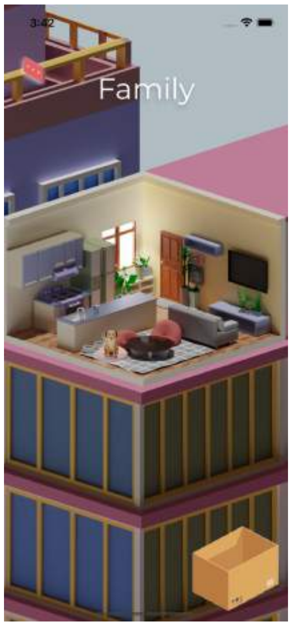
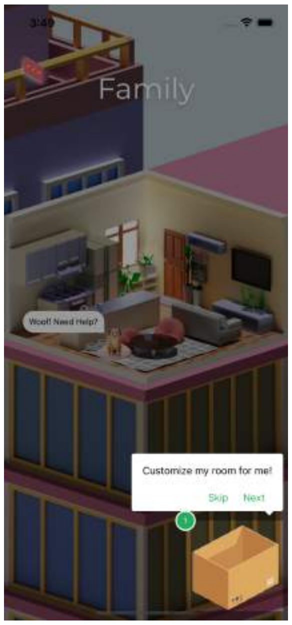
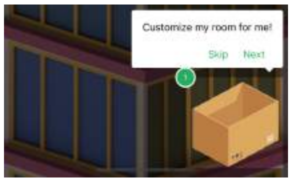
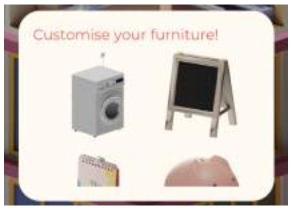
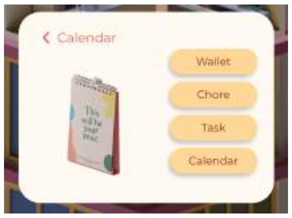
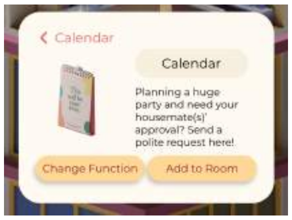


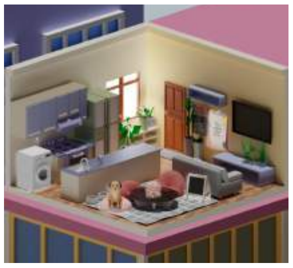
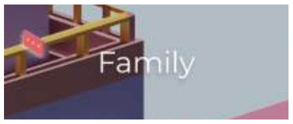
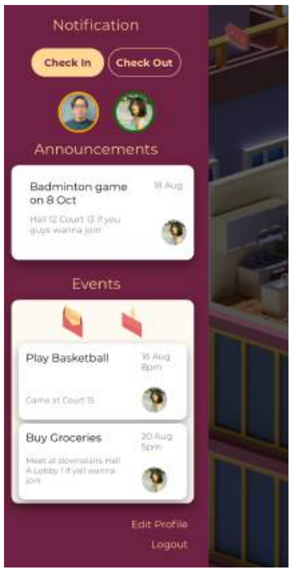
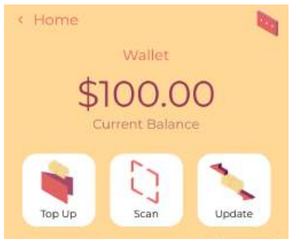
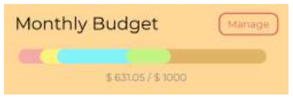
Figure A.14. User-interface sketch for task and function to collect payments from roommates

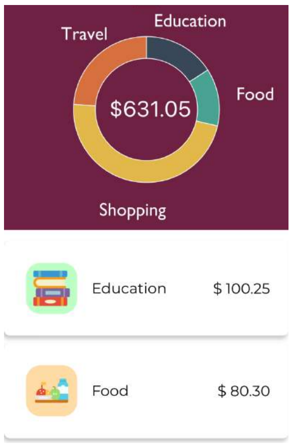
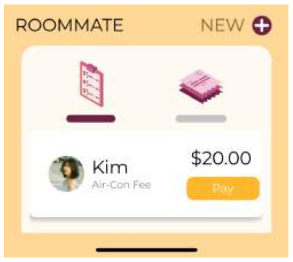
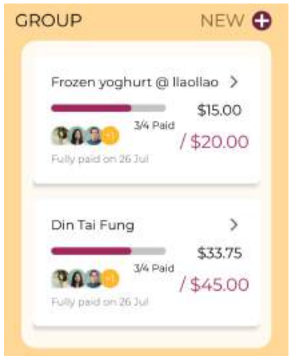
B. User Guide

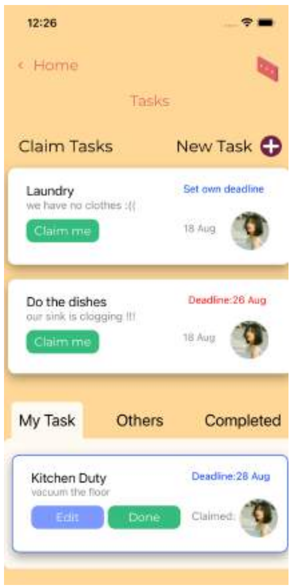
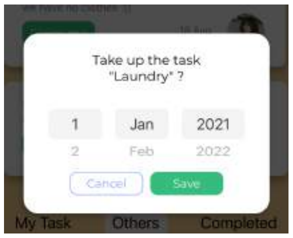
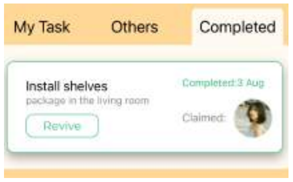
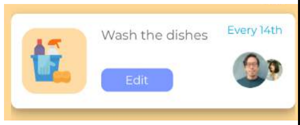
Screen	No.	Description	Image
Login	1.	If the user has an existing account, key in username/email and password.	 11:35 COHAB Username / Email Password Log In Create New Account
	2.	If the user does not have an existing account click on Create New Account	
Register	1.	Key in username, email, password and confirm password	 11:36 COHAB Username Email Password Confirm Password Sign Up Back to Login
	2.	Click on Sign Up to create a new account or Back to Login to return to login screen	

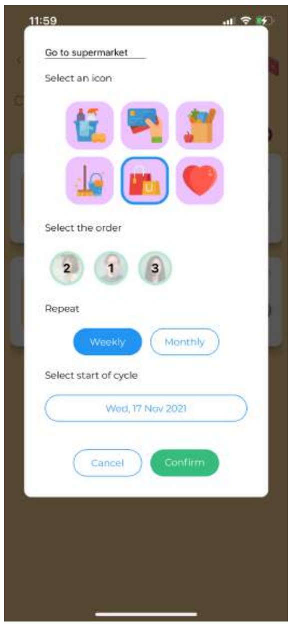
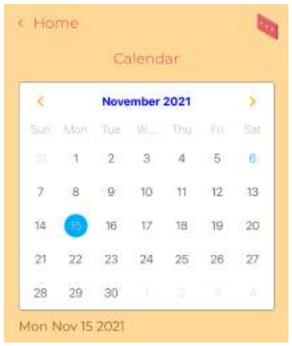
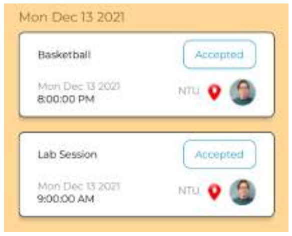
Group Select	1.	“Create” to create a new group or “Join” to join a pre-existing group. Once it is registered, it shows up on the screen. Click any group to manage. Click the dog Carl for guidance.	 11:58 Hello, Jane! Where will you be today? Hall EEE Family Create Join
Room (Main)	1.	Slide the screen to view the graphics.	

	2.	Click on the dog avatar to launch the user guide tutorial.	 Family Woof! Need Help? Customize my room for me! Skip Next
	3.	Click on the box on the bottom right of the screen to add furniture to the room.	
	4.	Select furniture.	
	5.	Select an app's function to associate with the furniture.	
	6.	Add to the room.	 Planning a huge party and need your housemate(s)' approval? Send a polite request here!

	7.	Click on the furniture to navigate to Finance, Task, Chore, Calendar functions.	
Notification	1.	Click on the notification button on the top left of the room screen.	
	2.	View your roommate's status, announcements and your events.	 Notification Check In Check Out Announcements Badminton game on 8 Oct 18 Aug Hall 12 Court 13 if you guys wanna join Events Play Basketball 18 Aug 6pm Game at Court 15 Buy Groceries 20 Aug 5pm Meet at downstairs Hall A Lobby ! if yall wanna join Edit Profile Logout
Finance	1.	Check your balance, top up, scan to pay and transfer balance.	 Home Wallet $100.00 Current Balance Top Up Scan Update
	2.	Check and manage your monthly budget by clicking on the manage button.	 Monthly Budget Manage $ 631.05 / $ 1000

	3.	Click on the progress bar to view your monthly expenses in a donut chart	 Education $ 100.25 Food $ 80.30
	4.	Pay your roommates. Click on the “NEW” to create an entry.	 Kim Air-Con Fee $20.00
	5.	Make group payments and share the expense among all your groupmates. Click on the “NEW” button to create an entry.	 Frozen yoghurt @ llaollao $15.00 3/4 Paid / $20.00 Fully paid on 26 Jul; Din Tai Fung $33.75 3/4 Paid / $45.00 Fully paid on 26 Jul

Task	1.	Check the list of current tasks. “Claim me” adds them into “My Task”, which means you are now responsible for it. Click “New Task” to add a new task up for claims.	 12:26 ‹ Home Tasks Claim Tasks New Task Laundry we have no clothes :((Set own deadline Claim me 18 Aug Do the dishes our sink is clogging !!! Deadline:26 Aug Claim me 18 Aug My Task Others Completed Kitchen Duty vacuum the floor Deadline:28 Aug Edit Done Claimed:
	2.	Set a deadline if the task you claimed has no deadline.	 Take up the task "Laundry" ? 1 Jan 2021 2 Feb 2022 Cancel Save My Task Others Completed
	3.	“Revive” a completed task if you wish to repeat them again without setting up.	 My Task Others Completed Install shelves package in the living room Completed:3 Aug Revive Claimed:
Chore	1.	Check the chores your group currently has, the type of chores, deadline and who’s currently responsible. Click “Edit” to edit the chore.	
	2.	Click “+” to add a new chore.	

	3.	Give a title to your chore and select an icon. Choose the order of responsibility by tapping on the profile icons. Select “Weekly” for weekly repeating chores (eg. Every Sunday) and “Monthly” for monthly repeating chores. Pick a day/date for the chore to start the cycle on.	 11:59 Go to supermarket Select an icon Select the order 2 1 3 Repeat Weekly Monthly Select start of cycle Wed, 17 Nov 2021 Cancel Confirm
Calendar	1.	Choose and click on the date to record an event.	 ‹ Home Calendar November 2021 Sun Mon Tue W... Thu Fri Sat Mon Nov 15 2021
	2.	View your upcoming events.	 Mon Dec 13 2021 Basketball Accepted Mon Dec 13 2021 8:00:00 PM NTU Lab Session Accepted Mon Dec 13 2021 9:00:00 AM NTU

C. How-To Set Up Guide

To run CoHab, first we have to install Node.js (https://nodejs.org/en/download/). Node.js is not used in our application, but it comes bundled with a package manager, npm, that we use to run our application.

Once npm is installed, we can install expo, which is a framework that helps run React Native, by running the following in our terminal:

```
npm install -g expo-cli
```

Next we need to obtain the source code for CoHab, either by cloning the Github repository or by downloading it as a ZIP folder and unzipping it.

After navigating into the unzipped folder, we run:

```
npm install
```

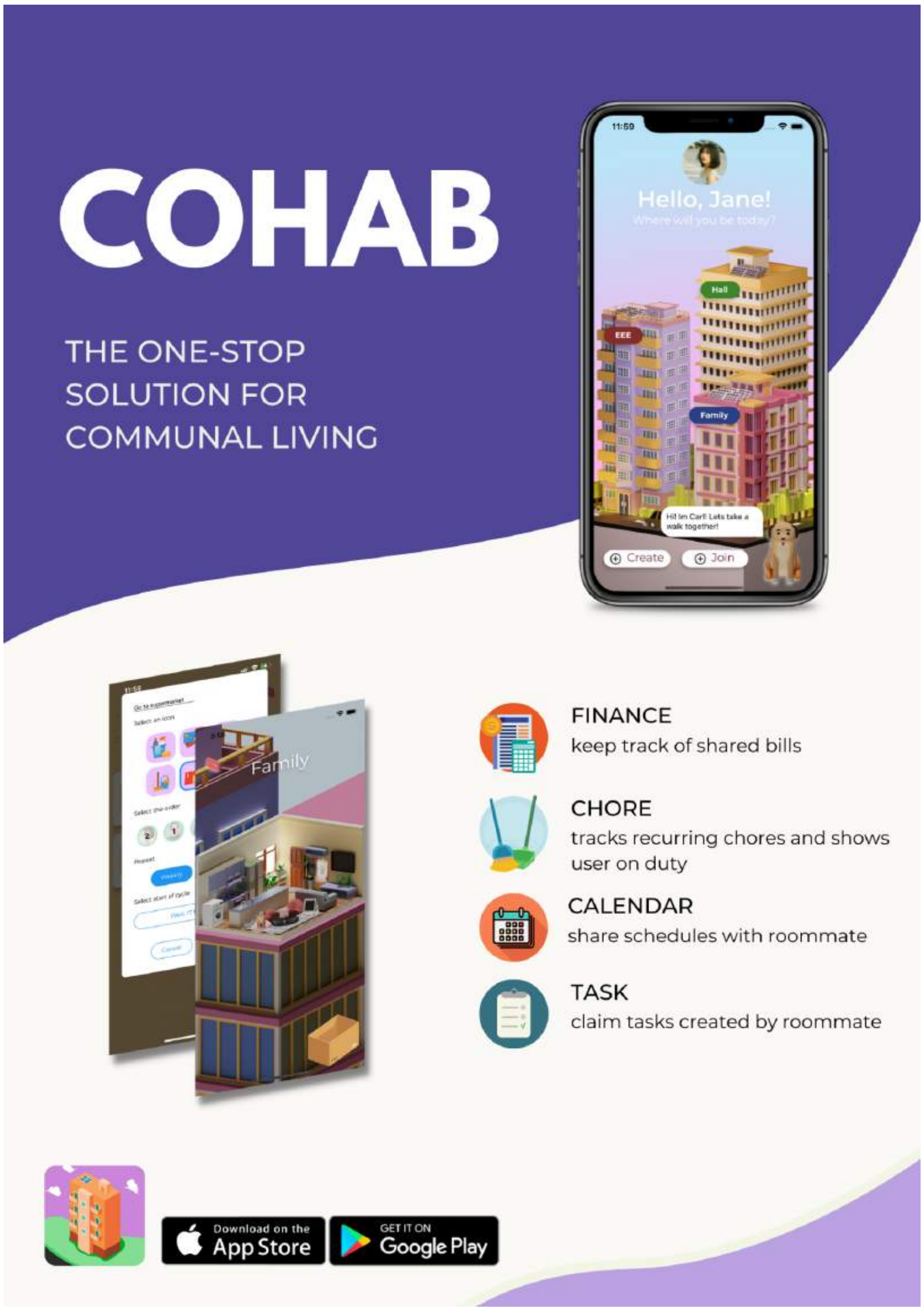
After that, run:

```
npm run start
```

A new page should open in your browser. Remember to keep your terminal open.

After that, download the Expo app on your phone from Google Play or App Store. Make sure that your phone is on the same wifi network as your computer. Scan the QR code given in the web page, and the app should be running on your phone.

D. CoHab Poster


COHAB
THE ONE-STOP SOLUTION FOR COMMUNAL LIVING
11:59
Hello, Jane!
Where will you be today?
Hall
EEE
Family
Hi! Im Carl! Lets take a walk together!
Create
Join
Family
FINANCE
keep track of shared bills
CHORE
tracks recurring chores and shows user on duty
CALENDAR
share schedules with roommate
TASK
claim tasks created by roommate
Download on the App Store
GET IT ON Google Play

E. Source Code

Source code can be located at GitHub URL: https://github.com/JiXiangChua/Cohab