Project Proposal

CM2020 – Agile Software Projects Final Proposal

Date of Submission: 20 September 2021

Created By:	**Student Number**	**Student Email**
WENG ZHU EN NATHANAEL	200261126	zenweng001@mymail.sim.edu.sg
POOJA SURESH	200261115	suresh008@mymail.sim.edu.sg
LIM XUAN TING, JACE	200629393	xtjlim001@mymail.sim.edu.sg
FOO JO MUN	200629430	jmfoo003@mymail.sim.edu.sg
LIM WEE KIAT	200629197	wklim012@mymail.sim.edu.sg

Table of Contents

Background

Planning

The initial plan for this project was to create a webpage that allows users to create their own e-books or e-writings, however, after much consideration and discussion, we decided to scrap that plan and to go with another idea which is to create an application that helps users find study spaces. We concluded that technically, a webpage did not suffice as the requirements we were given were for an application, therefore, Spaceus was born.

Logo

What inspired us to have the idea for this application was our experience of being students for over 12 years. On many occasions we find ourselves going out, seeking places to study alone or in a group with friends. However, taking into account how many unforeseen factors had negatively affected our study productivity, we trusted that creating this application would help students all around. The logo shown above that our designer had produced represents a door from an empty space to a beautiful and colourful world, assuring users a whole new world of productivity when using this application.

Goals

Spaceus: Our goal is to create an application that will help people find good study spaces. We want to be able to identify places that provide the optimal environment for a person to focus on their objective at hand, be it studying or doing work in general. Our application strives to provide an easy and excellent user experience and to allow our users to experience the major difference between good and bad studying environments. We also aim to integrate social functions for example linking with other people who study the same or similar modules to study together and help each other progress.

1) Application: Spaceus

2) Study space locator with built-in social features

3) Simple user experience

4) Appropriate functions

5) Simple yet elegant design.

Objectives

Primary objectives

1) Planning
2) Research
3) Creating a survey
4) Compiling survey data
5) Designing
6) Prototyping
7) Testing

Secondary objectives

1) Gathering additional survey data
2) Possibly finding new functions to add
3) Exploring the social function
4) Collaborating with owners of the study spaces

For the first 13 weeks, we focused more on planning, prototyping and our proposal, meeting once a week to figure out our approach to this project. After which we created a plan, a proposal and a prototype on Figma which we submitted for our midterm. It was only after the midterm assignment that we started on creating an actual application with the use of Android Studio.

We have met all our primary objectives without much hassle and have already started implementing some secondary objectives. We have already successfully created an application that works according to our specifications, we have a Log-in/Create Account page, a Home page, a Search page, pages for the specific locations, Profile page, etc. The application, although still has much room for improvement, works.

The features that are yet to be implemented are the social features, adding location features, real-time trackers and map features, however, we believe this will not be an issue in the near future.

Project Scope

Project description:

This application (Spaceus) will be used by students to improve their productivity during study sessions.

Project objective:

To create an application that helps students find good study spaces, share their favourite study spaces, and find people with the same or similar subjects/fields to study with.

Project timeline:

This project will start with research and gathering data, then proceed onto designing and prototyping. After we have come to a consensus, we will begin the development of the application. After the development is completed, we will test it within our private circle before releasing it to be used by the public.

Deliverables:

The Spaceus application will feature functions like:

1) Users will be able to add locations to the database

2) Users can search for optimal study environments to their liking with a filter function

3) Real-time crowd checker

4) Linking up with people who are studying similar modules

5) Distance indicators

Research

We researched 2 applications as inspiration for our application; the first being Waze and the other being Fern. Waze is a map and directions application which features functions that we strive to implement into our own application. The features being real-time trackers, user input and distance indicators. Our application strives to allow user input for locations that we had not yet included, distance indicators for user-friendliness and real-time trackers for crowd scanning.

Fern on the other hand is an application largely similar to ours. It is a study space locator native to Texas, United States of America. We thoroughly researched this application in order to find its flaws and strengths so as to not repeat their mistakes and to take note of what they were doing right, allowing us to strive for an application superior to Fern.

We also conducted a survey to gather information and feedback on how we could improve our user experience and what kind of functions users would enjoy on this application. The survey that was completed by over 100 participants served as a good indicator as to which direction our application needed to head towards. User and usability testings were also organised with the same intentions.

Group

Our group was divided voluntarily to choose what roles we would partake in. We unanimously voted in our leader and proceeded with a very democratic approach. Our workload was split evenly by the leader into 5 different tasks per week and everyone then chooses a task they wish to complete. We hold a meeting weekly for progress updates, task delegations and overall discussions.

We utilised tools like Kanban boards, google sheets, google forms, google docs and Gantt charts to help us achieve our goals. A Kanban board was used to delegate tasks and to track our progress, we used google forms to create a couple of surveys to gather information and used google sheets to track their results. Google Docs was used as the main body of the proposal due to its extremely convenient nature. Gantt charts were created to plan out our activities for the entirety of the project.

Upholding the democratic nature of our group, we believe that although some of us hold better skills at coding, designing or writing, we managed to somewhat equally divide the workload. All 5 of us contributed to this project and managed to bring this to what we believe is a huge success and a big accomplishment.

Planning and Research

First half: Project Start to Midterms Deadline

Over the course of these 22 weeks, we were required to produce two sets of proposals, the first for our midterms and the second for our finals. This means that we have the first 13 weeks to complete components for the midterm, and the remaining 9 weeks to focus on our final submission. There were a total of three deliverables we needed to submit: A prototype, a report and version control logs. In order for us to successfully complete all the work required, we had to weigh our options based on our resources. The varying types of work meant that while some components could be shared between members, others were better off being done by someone with the skills to do it. Hence, we decided to split the work among ourselves equally, based on each member's capabilities.

For the midterms, our prototype would only consist of a Figma drawn prototype, while our report covers details on the process of creating the prototype and the versions we went through to come up with the current design. For the finals, our prototype will be coded into software and the report will be edited to include further details. In order to better manage our time, our group decided to create a Gantt chart with our activities planned out. The Gantt chart of the originally planned schedule is shown below (Fig 1), as well as a brief explanation for each activity. Our activities have been generalised into these 11 items and the time given for each activity is inclusive of buffer time. The chart starts from Week 6, which was the week we had our first meeting.

Initial Schedule

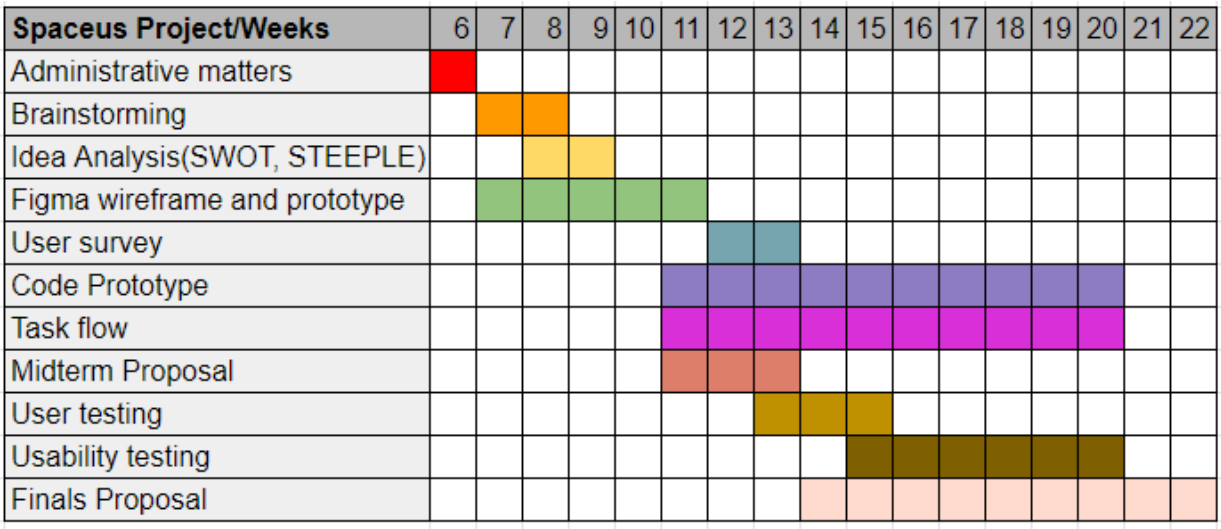


Fig 1. Gantt Chart of originally planned activities

Details of Activities

Administrative matters: *Miscellaneous activities that are necessary but are not directly related to the project itself. For example, icebreaker activity.*

Brainstorming: *The process of coming up with ideas and voting for the group's preferred ones, as well as discussing the pros and cons of the ideas.*

Idea Analysis (SWOT, STEEPLE): *Evaluating the idea with proper research into the subject matter and with methods such as SWOT analysis.*

Figma wireframe and prototype: *Drawing a wireframe on Figma and proceeding to produce a prototype using Figma's features, creating an interactive app through functional buttons that lead to each slide.*

User Survey: *A survey intended to understand users and their study-related needs.*

Task flow: *Charting out the individual functions of the project in terms of a step by step process of how a user would accomplish a specific action, for example, Logging into the app.*

Midterm Proposal: *Typing out the proposal, checking for errors and deviations.*

Code prototype: *Activities necessary for creating the prototype, like learning the programming language, IDE, and coding the actual product on the IDE.*

User testing: *A survey intended to get opinions from users on how a study place locator should work, what it should look like, and get users to evaluate our current design.*

Usability testing: *A series of tests intended to get users to trial our product and thus evaluate the product's usability.*

Finals Proposal: *Typing out the proposal, checking for errors and deviations.*

Actual Progress of Activities

Initially, our group faced trouble finding our members due to the Phase 2 Heightened Alert (P2HA) measures in Singapore leading to an all-online format for our lessons. Eventually, we managed to hold our first meeting in Week 6. It was mostly administrative work as we needed to settle our roles and self-introductions before we start work, so as to better understand the abilities of each member of the group. After that, we organised a rough plan and completed the group activities on Coursera.

The goal for the next meeting was to quickly brainstorm for an idea and settle on it, in order to maximise time for prototyping, testing and coding. Team members were requested to come up with a project idea to present to the group and do a SWOT analysis on it. Members were also required to conduct research on their project in order to substantiate their SWOT analysis. During the meeting, we would then assess each other's ideas, and discuss what improvements could be made as well as the feasibility of the idea. After the initial brainstorming process, we first came up with the idea of an eBook website, similar to that of Wattpad, where users are able to publish their own stories on the website.

Unfortunately, due to the definition of software not clearly including a website, as well as the market being filled with competitors, the idea was later set aside. This meant that we had to take an additional meeting to discuss a new idea and this landed us on our current idea of a study place locator app. Appendix 1 contains a simplified diagram of our initial workflow from brainstorming to writing the midterm proposal. A more detailed breakdown of the tasks completed by each team member throughout the initial phase of the project is presented in the table below.

Table 1: the contribution of each team member for midterms

TASK	Name	Start Date	End Date	Duration
Brain storming of ideas and SWOT ANALYSIS	ALL	05/22	05/25	3
Preparing of Research questions (Ebook - Readers) :				
what can we learn from our competitors	Jo Mun	05/25	06/04	10
What design decisions to make?	Wee Kiat	05/25	06/04	10
What are the advantages and disadvantages	Jace Lim	05/25	06/04	10
What are the characteristics which can affect user satisfaction	Suresh Pooja	05/25	06/04	10
Why would our users prefer our product?	Zhu Weng	05/25	06/04	10
Research Questions was reviewed & Prototype was selected	ALL	06/04	06/04	0
Presenting Taskflows				
Registration -> Home page(after login)	Jace Lim	06/11	06/18	7
Home Page(after login) -> Reading a book	Jo Mun	06/11	06/18	7
Home Page(after login) -> Writing a book	Wee Kiat	06/11	06/18	7
Home Page(after login) -> Report an issue	Suresh Pooja	06/11	06/18	7
Home Page(admin) -> Review a book	Zhu Weng	06/11	06/18	7
Selected a new idea for the project : Study spot locator (SPACEUS)				
SWOT analysis	Wee Kiat	06/18	06/23	5
Research about any existing study spot locator	Suresh Pooja	06/18	06/23	5
Creating branding ,logo and UI design	Jace Lim	06/18	06/23	5
Stakeholders and general info	Zhu Weng	06/18	06/23	5
Survey for data collection	Jo Mun	06/18	06/23	5
A new prototype was selected				
Completion of survey questions	Wee Kiat &Jo Mun	06/23	06/26	3
Research on more study spaces	Zhu Weng	06/23	06/26	3
Research about Maps Javascript API	Suresh Pooja	06/23	06/26	3
SWOT analysis (adding extra points)	Zhu Weng	06/23	06/26	3
Task Flow	Jace Lim	06/23	06/26	3
Software Project Proposal				
Aims and Objectives	Jo Mun	06/26	07/01	6
Planning and Requirements	Suresh Pooja	06/26	07/01	6
Formative Testing and Evaluation	Zhu Weng	06/26	07/01	6
Prototping Techniques	Jace Lim	06/26	07/01	6
Evaluation Techniques	Wee kiat	26/21	07/01	6

Each assignment listed in the table was meant to be finished before the following meeting. Future tasks were all done in a similar fashion, with a deadline for completion given, which was either the next meeting or another specified meeting. The division of each activity into subtasks allowed us to concentrate on one thing at a time, resulting in a more productive process. For example, our project proposal was split into different sections, and every section was delegated to a member of the group.

Having deadlines kept us on track, and the weekly meetings helped us examine our shortcomings while also motivating us to track our progress. We were able to focus on quickly finishing each piece of work by breaking down a single task into many sub-tasks. Finding each team member's strengths and weaknesses during the ice breaker activity helped us to assign work based on their strengths, allowing them to complete the tasks more quickly and effectively before the deadline. This led to us finishing a large majority of the goals we wanted to achieve in the midterms.

However, due to the sudden change in ideas, our schedule had to change as well. By taking an additional week to brainstorm, we had one less week to start work on our code prototype. We also had to put some time into learning how to code Kotlin and how to use Android Studio. In response to these changes, we decided to spend our first few weeks learning more about Android Studio and mobile development.

Second half: Midterms Deadline to Finals Deadline

Unfortunately, coding a mobile app turned out to be tougher than expected and our buffer time was unable to cover everything we wanted to include in our project. Our goals had to be reviewed and our end product condensed to only include the core features that represent what our project is about. As a study space locator, we wanted to give our users a holistic experience when it comes to finding a good place to study. This meant that only our most critical features would be included in the project: a function to list study spaces, a search function, a login, and a create account function.

This also meant that our plans had to change accordingly. Below is the updated Gantt Chart that takes account of the new circumstances.

Reviewed Schedule

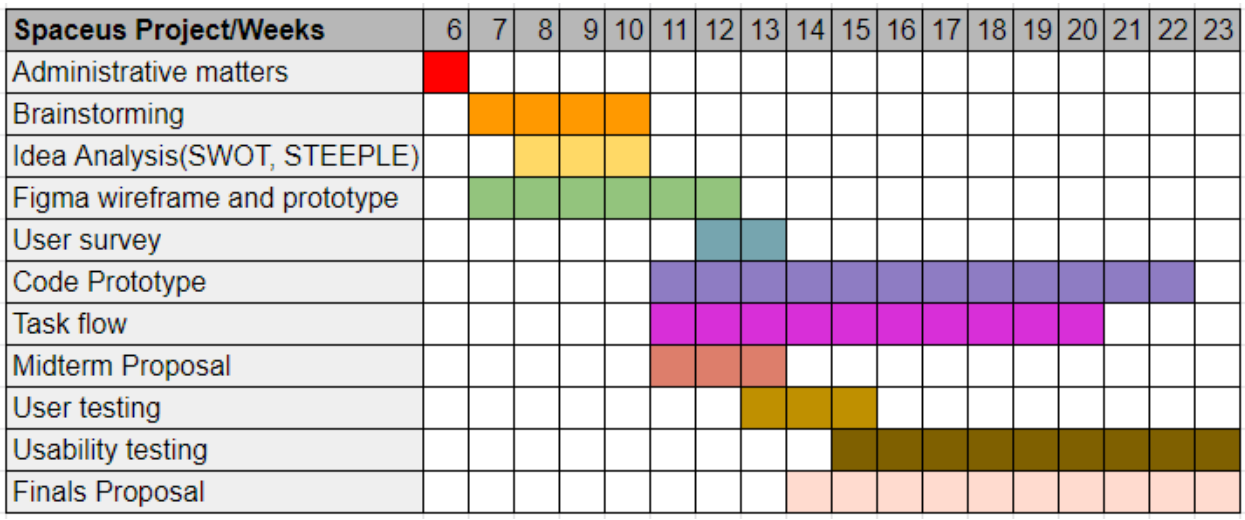


Fig 2. Updated Gantt Chart

With more time spent on brainstorming and a new idea that ventures into unknown territory, the time we spend on coding and our proposal must also increase to adjust to the higher level of technical difficulty. This also means that more time is needed to work on adjusting task flows and conducting usability testing. Furthermore, the deadline for submission was extended by an additional week. Thus, our plan has changed to allocate more time for coding the prototype, testing, and writing the proposal. This additional time will help us to improve our product and refine our proposal for submissions, since the one resource we lack is time.

Learning from survey results

As shown in the schedule, we performed a survey to better understand the user requirements. We had a total of 111 respondents and from them, we received a mixed set of data. When we evaluate our prototype, we need to find out if our product is effective, efficient, and satisfies users. To better understand the criteria for usability, we have to understand what users need and want out of an app like this.

Hence, we set out to ask them the following questions during the survey:

1) Where do you normally study?
2) What would you consider a conducive study environment?
3) What are some essential features a study space needs?
4) How far in terms of travel time via public transport would you travel for a conducive study location? (e.g Less than 20 minutes of travel, more, etc.)
5) How often do you experience a situation where you are planning to study at a certain place but could not find a seat?
6) What affects your productivity when studying?
7) What can be done to improve the productivity of your study sessions?
8) When do you feel most productive?
9) Would you be open to studying with strangers?
10) Would studying the same topic with strangers improve your productivity and why? Please elaborate.
11) What time do you prefer to study?
12) How much time in a week do you spend studying outside home?

Results from the survey

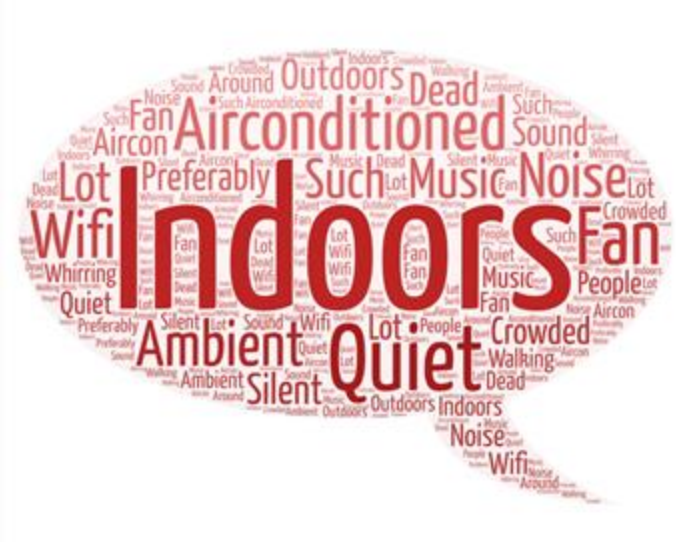
Questions 1 to 4 deal with the study environment the respondents would prefer and to know how far each user will be to travel since some users don't mind travelling long distances.

Here is what we gathered :

1) When asked about the places users commonly preferred to study

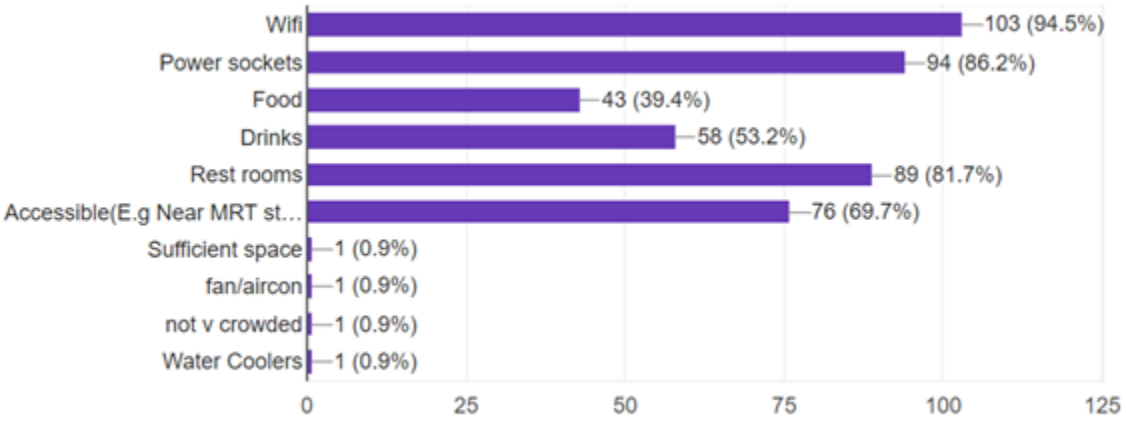
More than 3/4th of the respondents answered they would prefer to study at home followed by library and café having equal votes.

2) When asked about the characteristics of what a user considers a conducive environment



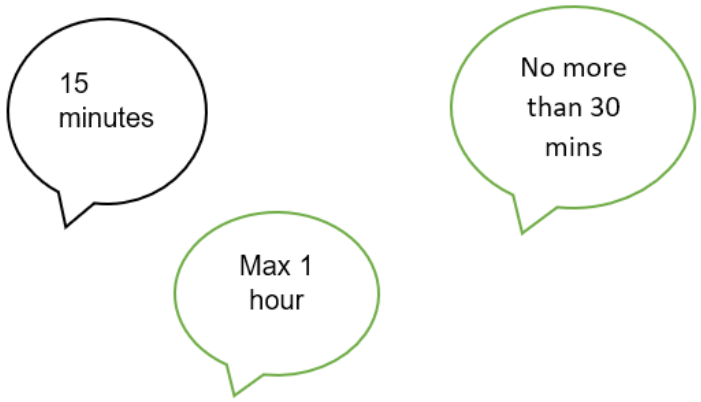
This question was drafted to know what factors can affect one's productivity sessions to improve features in the app to possibly have filters such as noise levels and wifi availability to cater to user's needs. The majority of the participants preferred to have study spaces that are indoors, air-conditioned, with Wi-Fi availability and ambient and quiet settings.

3) When asked about what are some essential features a study space needs?



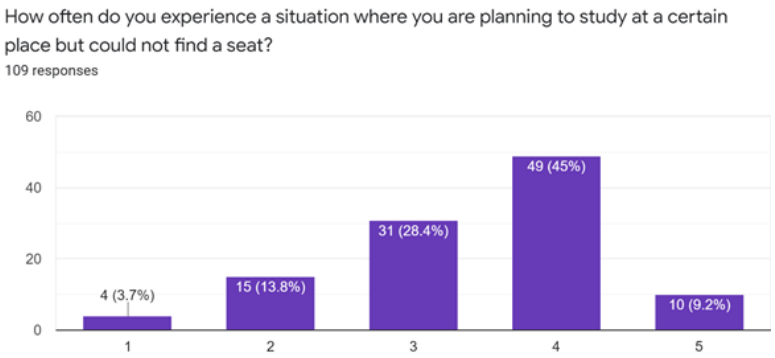
The majority of the survey respondents chose Wi-Fi followed by power sockets and restrooms. This was to figure out what were considered to be important factors for determining a study space.

4) When asked about how far survey respondents were willing to travel



The majority of the respondents were willing to travel anywhere between 15 to 30 mins and a few also mentioned they can travel up to 1 hr to find a study location. This question helped us to analyze the importance of finding good study environments around the user's current location.

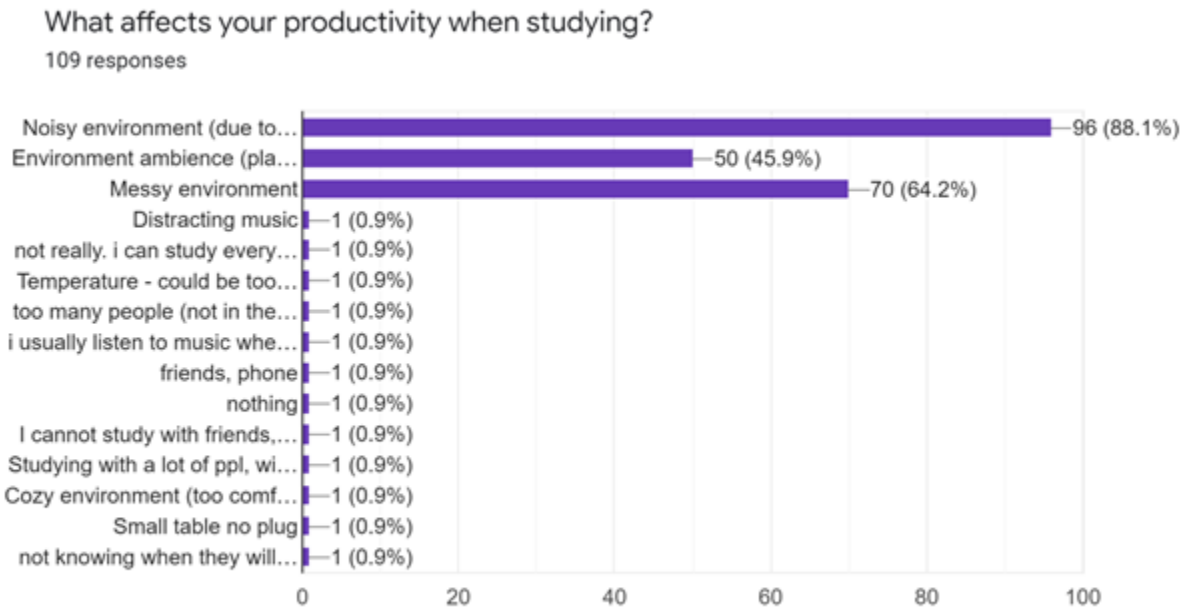
5) When asked about how often the participants of the survey were unable to find seats due to crowd levels.



The majority of the participants selected 4 from a range of 1 to 5 given “1“ refers to not being able to find seats and “5” refers to being able to find seats. From the above question we were able to infer that the respondents were able to find seats only on some occasions.

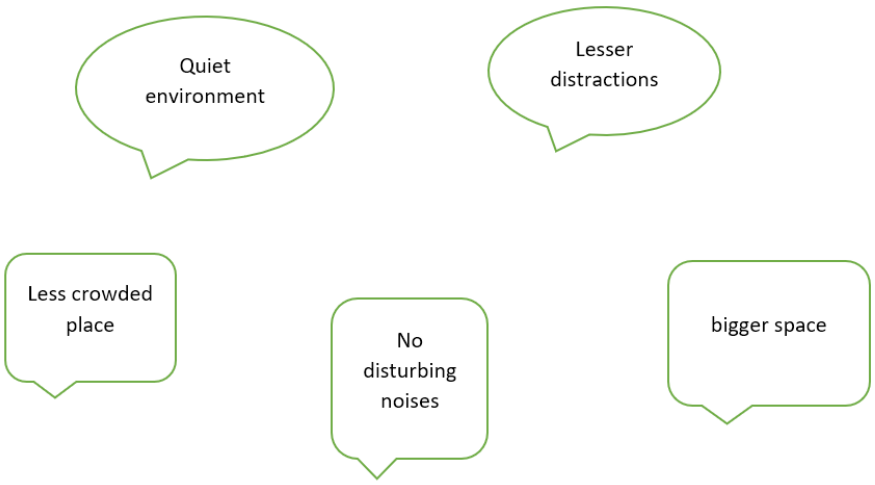
Questions 6 to 10 dealt with productivity. The questions involved in asking users what factors affect one's productivity and what can be done to have good study sessions

6) When asked about what affects user's productivity while studying



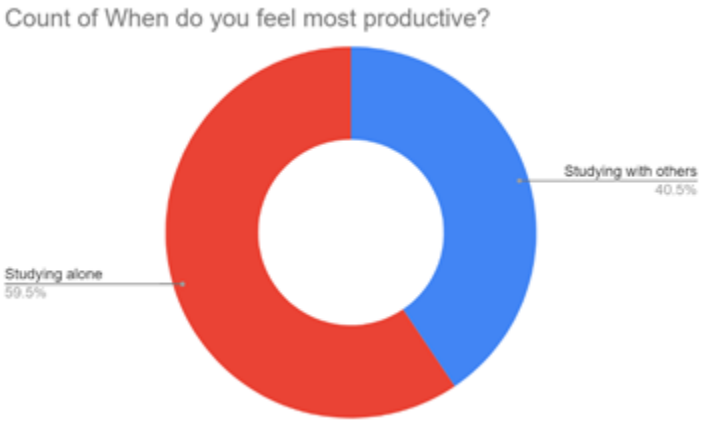
There were varying responses but noisy environment is in lead by 96 votes followed by a messy environment and environment ambience. This question helped us to analyze the user's distractions which can help in designing the filters more effectively.

7) When asked about what improvements can be made to increase productivity



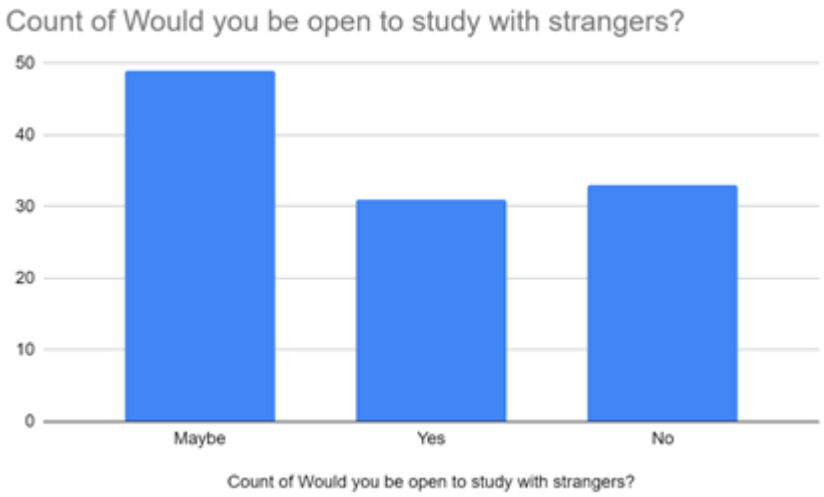
The speech bubble highlights the top five responses. The participants would prefer a quiet environment with fewer distractions. This gives us insight into what may be the distractions of the questionnaire respondents

8) When asked about whether users are most productive while studying with strangers or alone



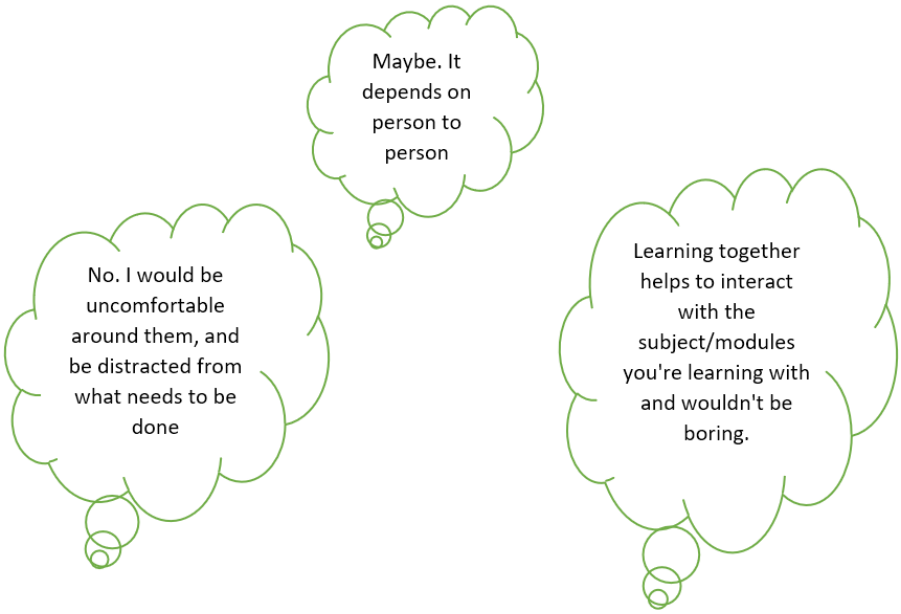
A huge number of the respondents answered that studying alone increases productivity rather than studying with others. This question allows us to decide regarding the features for the app

9) When asked whether those who responded to the survey would be open to studying with strangers



Those who submitted their responses to the survey responded with a “maybe” and almost equal levels of yes and no answers. One of the objectives of the app is to allow users the option to study with strangers, hence the question allowed us to have a rough idea about creating that type of feature.

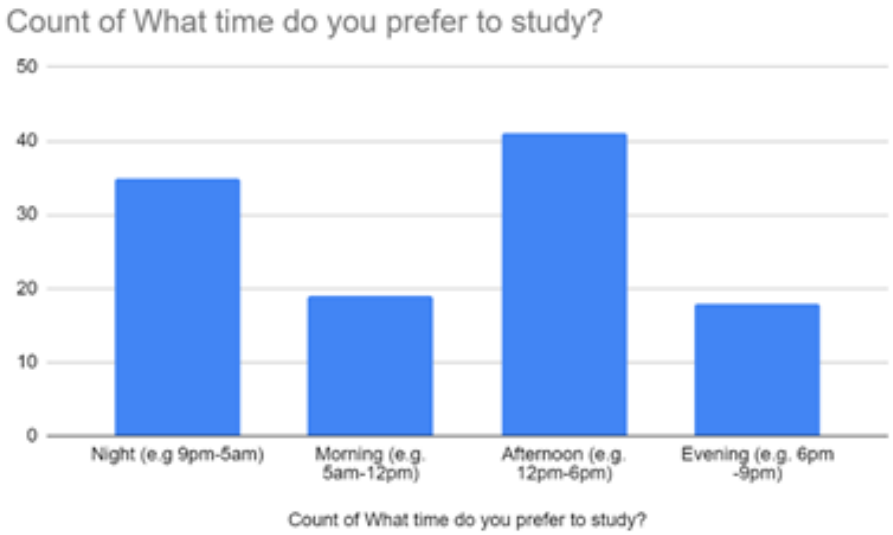
10) When asked about whether they would be willing to study similar subjects with strangers



These are a few of the responses we received for the above question. The responses were mixed. The question was a follow-up question to the previous question. We wanted the users to elaborate on why they chose to answer a particular way to the previous question.

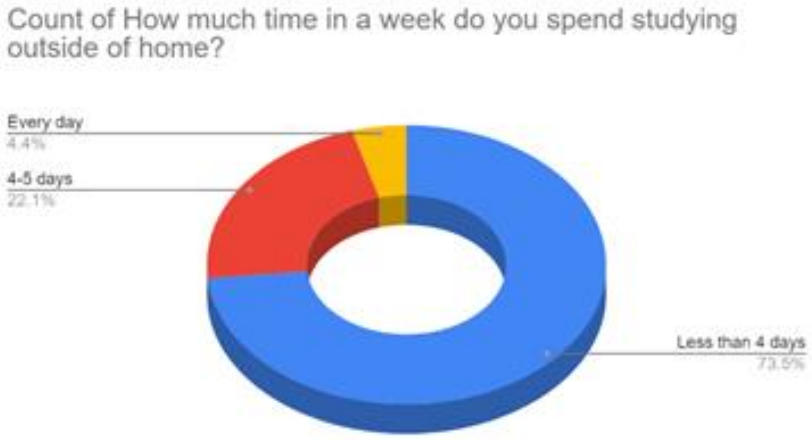
Questions 11 and 12 dealt with timings the users preferred to study and how many days they study per week.

11) When asked about the timings survey respondents preferred to study



Those who responded to the survey responded that they prefer studying in the afternoon followed by night. This can also be helpful to provide study spaces for users to study anytime during the day.

12) When asked about how many days per week users preferred to study



From the above chart, we were able to infer that users prefer to study less than four days and the number of users who study every day is significantly lesser. This can help us to analyze how many users may prefer using study spaces each week.

Our team has analysed the survey results and processed a mixture of quantitative and qualitative data into diagrams that will be relevant to setting standards for our prototype testing.

From the data in Appendix 4 Fig 1, we can see that a large majority of users study at home. When they study outside, the top three options are as follows: Cafe, Library, School. This suggests that across the target audience, they are likely to frequent these locations and identify them to be conducive for work.

Appendix 4 Fig 2 also tells us that users need a quiet, air-conditioned environment for them to be productive. This overlaps with the features of cafes, libraries, and schools, making these likely be the reasons why users study at these locations.

Appendix 4 Fig 3 shows what essential features users think a study location should have. Wifi, Power Sockets, and Restrooms are listed among the top three features. Users are likely to download our app in hopes of finding locations that match these criteria. Thus, we can conclude that if we include functions that are relevant to the app, it can make the app enjoyable to use.

User Persona

Based on the survey data, we have generated a user persona. While our target audience is students, other potential users include work from home staff and people just looking for a quiet place to sit. Hence, we made a profile for our users, Cafe Cammy. Cafe Cammy likes to study alone at quiet locations that are air-conditioned, with Wifi and power sockets readily available. She dislikes noisy places with bad lighting that take more than 30 minutes to get to. While Cafe Cammy stays at home to study most of the time, she occasionally studies at cafes and libraries, usually in the afternoon or nighttime. Cafe Cammy is a general image of what our users should be like. Appendix 5 contains a diagram summarising information on Cafe Cammy’s like factors they consider to be a conducive study environment and their habits.

Prototyping and Iteration

Our initial idea for the project was actually to create an eBook reader. However, after careful consideration, we decided to shift our focus towards a more creative aspect. This change in approach led us to instead set up our very own study space locator, Spaceus. We started by laying down the fundamental aspects of the app. This included the aims and objectives of building the app, references from existing projects which were related to the theme we were heading for. It was an important step to help us develop more design ideas and the features we would potentially incorporate into our app. As a team, we went through a variety of colour palettes. We managed to narrow down our options to either a blue or neutral theme, which we finalised on the latter for our application. We personally designed the basic prototype features: create account, home, location, favourites, search, filter and settings page.

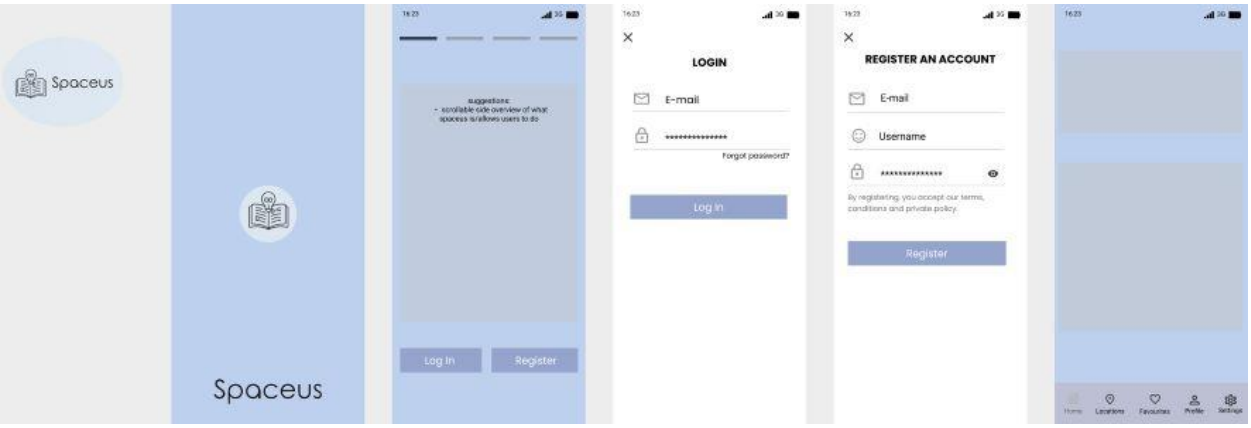


Fig 3. Blue colour palette

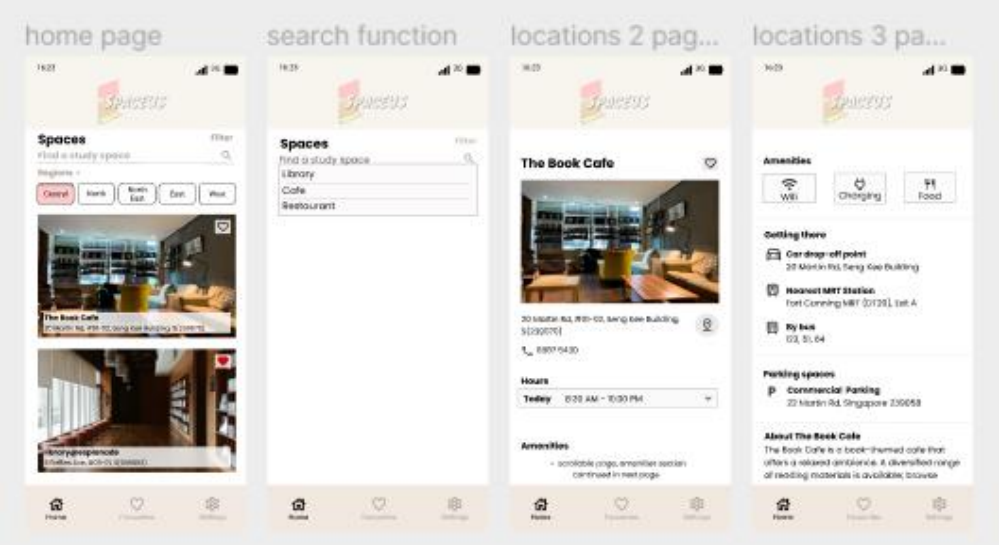


Fig 4. Neutral colour Figma palette

We collectively decided to create a survey to help in the process of deciding specific features of the app. The survey consisted of 14 questions, in which we collected over 70 responses and consolidated them to build the features on Figma. Figma was chosen as we felt like being able to jointly provide our inputs and visualising how the prototype would manifest while working on it was of importance to us.

After the submission of our midterm project, we started to brainstorm ideas on how to bring the design into a working prototype. We individually assumed responsibility for learning the Kotlin language to the best of our abilities by referring to videos and tutorials on the internet. We chose Kotlin as our programming language for our prototype due to its simplified syntax and capacity to have less code than Java. Initially, the prototype we had on Figma revolved around ideas taken from the survey done before our midterm project.

We opted to do our first round of unit testing before designing our actual prototype on the android software. Thus, we asked age groups ranging from secondary school to university students to navigate through the non-working prototypes. We successfully gathered valuable feedback on the effectiveness, efficiency, user needs and satisfaction for each app feature.

The Summary of Round 1 of Unit testing is given below:

Our initial prototype was a Figma prototype with no actual code. The prototype merely switches between fixed images of what we envisioned the app to be like, with each button swapping between the relevant images. Since we have limited functionalities, we have not done any user testing yet.

Our group asked users from different student groups to view the interface and try using the prototype. Success will be measured for each function in terms of effectiveness, efficiency, and user satisfaction. The evaluation criteria for effectiveness is whether the user can achieve an outcome or not. For efficiency, it is how quickly a user can achieve the outcome without extra steps. For user satisfaction, it is how pleasant the process of achieving the outcome is. An example of a table used for filling in testing data and an example set of questions has been included below.

Example Evaluation table:

Function	Question	Effectiveness	Efficiency	Satisfaction	Comments
Sign up		Yes	No	Yes	Too tedious to sign up
Sign up					
Login					
Search					

Example questions:

1) Can you create an account on the app? Yes/No, comments
2) Is it easy to do so? Yes/No, comments
3) Do you like the interface? Yes/No, comments

Throughout development, more testing will be needed. Users will know what they need and want. By continuously testing each version of the prototype, users will be able to point out flaws in the prototype. This is called iterative user testing, and this process helps us to build on the current prototype and improve its functionality over the software development life cycle. We need to be prepared to trial our prototype on a wide variety of users that match our user persona, with different conditions and scenarios, so as to thoroughly ready the application for usage in real-world conditions.

The user testing results provided a lot of guidance while working with the design layout. The majority of user feedback included a preference towards having the settings page combined with the profile for easier view. Other suggestions included removing unnecessary details such as the “about us” page due to its potential to deviate attention, an option for mail logins for convenience, and an eye symbol to help maintain password privacy. We followed up by analyzing the data and discussed the possibility of implementing the suggested features and modified the low-fidelity prototype on Figma accordingly.

We implemented a create button to allow a user to create their account while being able to navigate to the login page from the create account page and vice-versa.

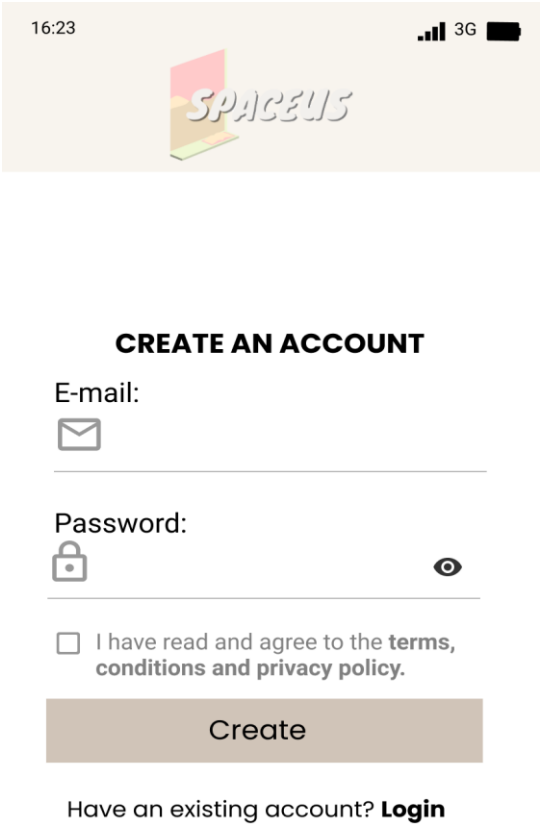


Fig 5. Create an Account Page with the Create Button

A simplified three-item navigation bar that included Home, Favorites, Settings as its components with reference to user feedback. “Home” allows the user to search for the location and includes a filter button. “Favorites” allows the user to view the bookmarked locations. Finally, “Settings” includes the profile page and general settings for the app, with the terms and conditions & privacy policy page.

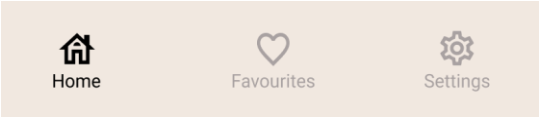


Fig 6. Three item navigation bar as seen on Figma

The Home page was combined with the location page, inclusive of a search bar along with a horizontal navigation bar to categorize locations by regions and a list of locations, giving the user opportunities to bookmark the location.

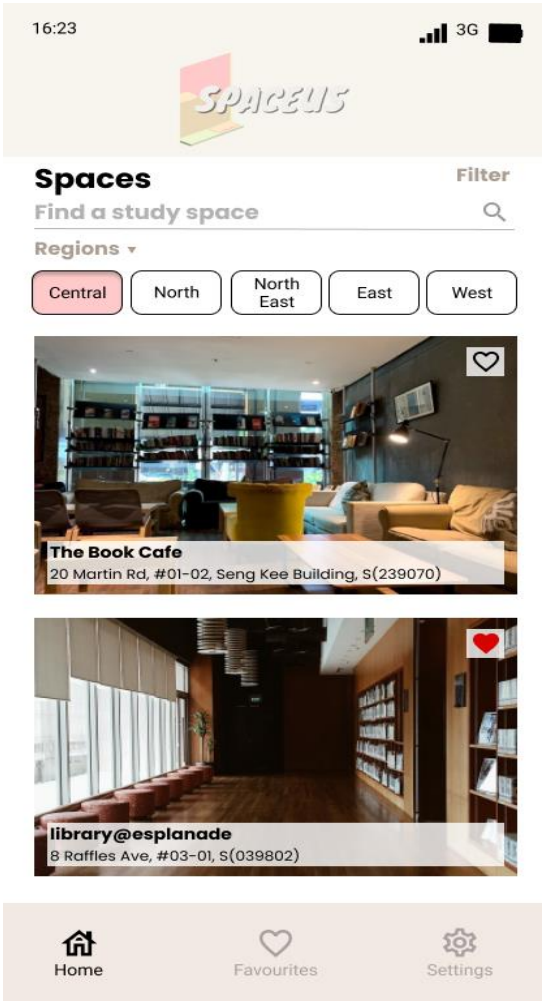


Fig 7. The Home Page on Figma

As the user feedback showed a preference towards a minimalistic design, the filter page was changed to a more visually simplified interface that has a plain background, unlike the one made on the previous prototype. The filter we created after user testing had five buttons. They classify the places by Wifi, Restrooms, Plug Points and 2 separate modes of transport; travel via bus or Mass Rapid Transit (MRT) individually. Two buttons, namely the apply and the clear all button, were included for user convenience

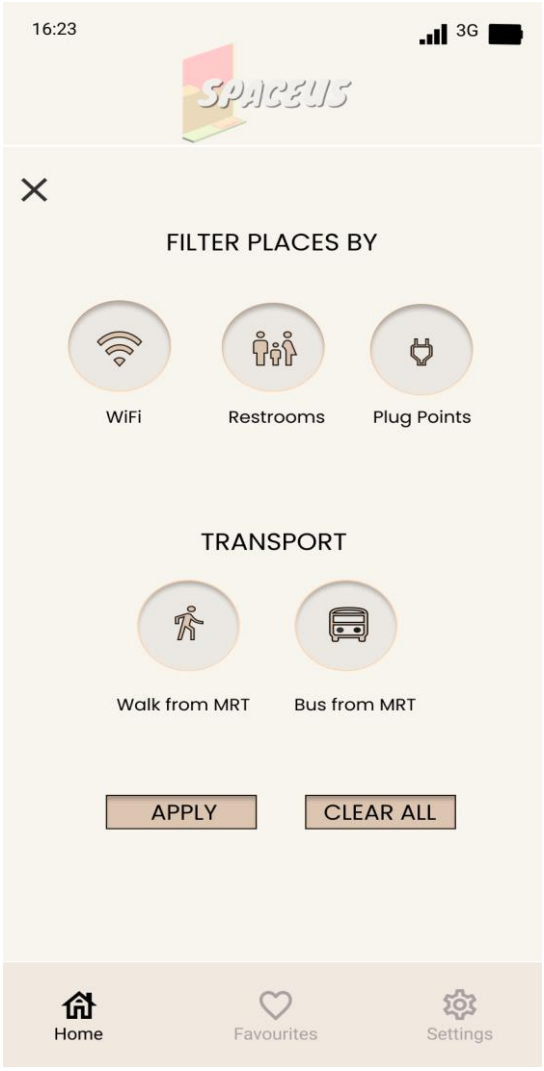


Fig 8. Filter Page on Figma

The Profile page was then merged under the Settings page. The updated Settings page allows the user to view and edit their profile while having a feature which enables the user to add a location to their personal list.

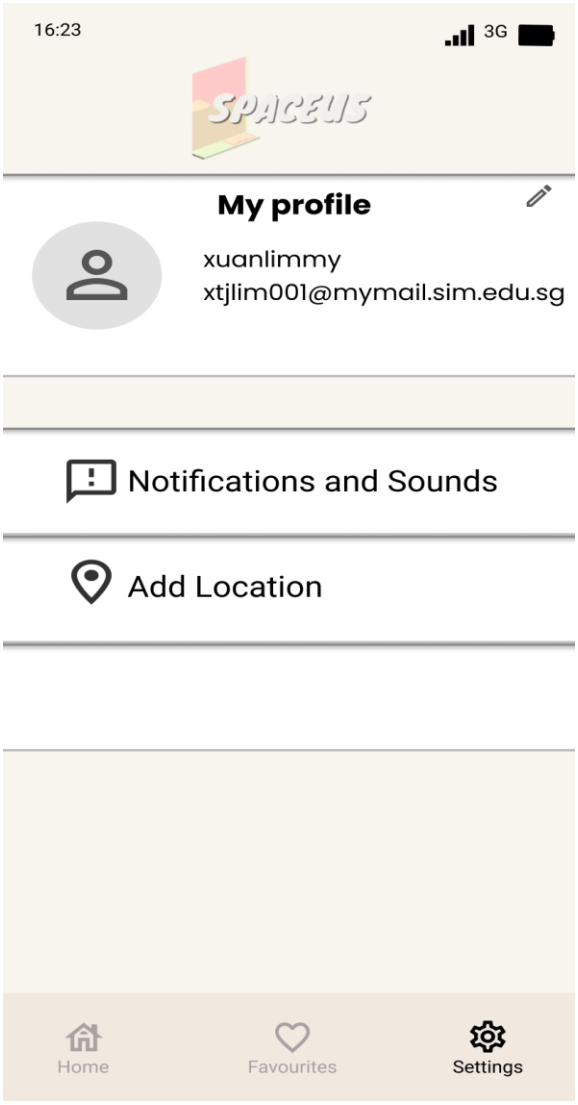


Fig 9. The Settings Page on Figma

For a variety of reasons, the shift from a low fidelity prototype to a high fidelity prototype proved difficult. Our team was amateur in coding and Kotlin was a new language which made the coding process very tedious. This led us to make changes in our initial elaborate low fidelity prototype which consisted of a lot of functions that we had to remove in order to work with the programming language efficiently.

The coding process involved having the team members split the app into its components and working on each component separately, followed by sending in their codes for merging and editing for the next meeting. We used Android Studio, which is an application which consists of activities and additional components such as the fragments, values, layouts, manifest file, as our main work platform. The bottom navigation bar was made of fragments, which essentially represents a portion of the user interface in an activity (an activity is a single-screen user interface which performs an action on the screen). The navigation bar helped us to navigate through different components in our app.

We decided to work with a constraint layout while adapting relative layouts for certain pages of the app to maintain uniformity. Constraint Layout simplifies the creation of large and complex layouts. The main advantage of Constraint Layout is that it helps to create a flat view hierarchy which is very beneficial in the case of performance analysis.

When we first started coding, everyone quickly discovered that it was difficult to replicate the exact features designed on the template (in Figma) onto the actual Android application. We had to make some design changes several times during that process. Each of the teammates brought up the issues they encountered while coding for a specific feature, and we collectively discussed the possible solutions.

The final version of the prototype was executed using an android emulator that is present in android studio. We all agreed to test the application on google pixel 3a with android 11. The android emulator simulates Android devices on our computers, allowing us to test how it would run on different devices.

We had to include a username and a contact number field in the Create Account page to make the Create Account and Profile page on settings work in unison for the android application.

The Forgot Password page, which involves sending an email to the user for them to create a new password if they had forgotten it, was not designed on Figma because it was not the primary function. As a result, we decided to stick with the initial design we came up with after deciding to add the function to our application.

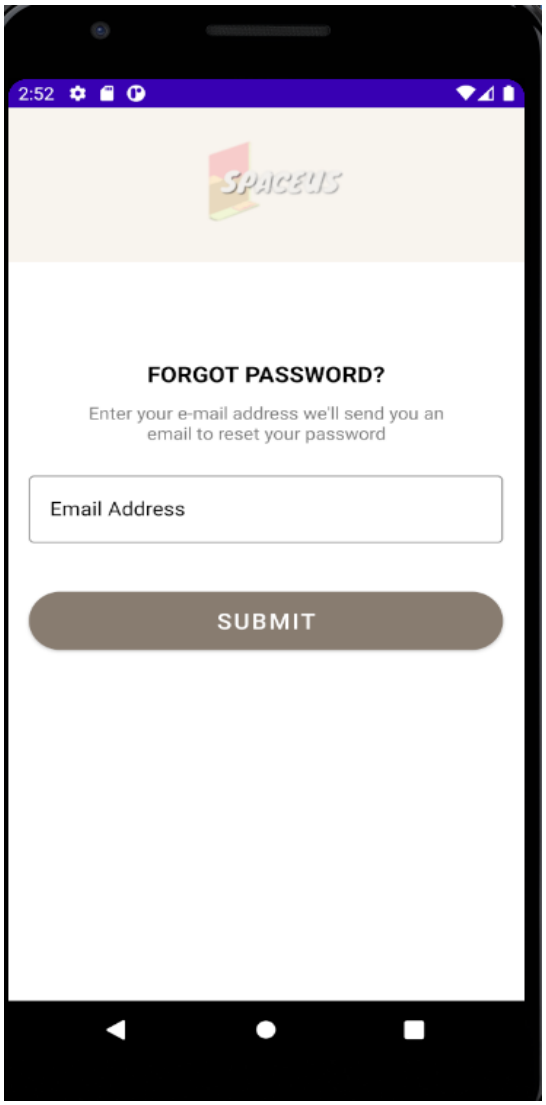


Fig 10. The Forgot Password Page on Android Studio

The Home page had undergone several changes due to the lack of time and the complexity of coding. We had tried to implement buttons which can filter the places based on regions. However, we came to a consensus to remove it and keep the interface minimal to avoid cluttering too many functions into a single page.

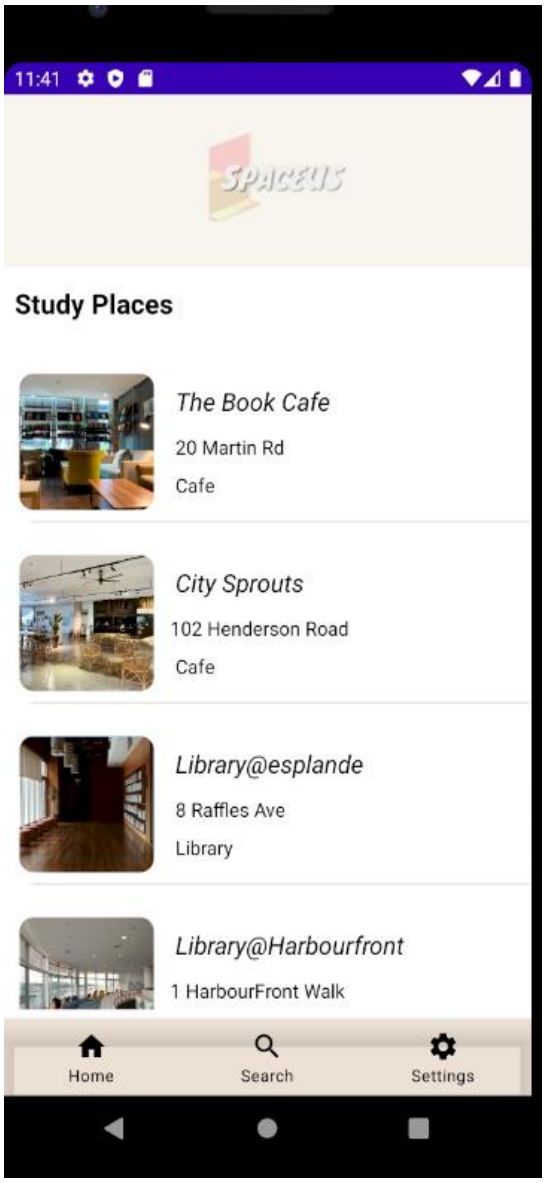


Fig 11. Final Version of the Home Page on Android Studio

We agreed to remove the favourites page given the difficulty and time constraints for the actual prototype, hence it was replaced by the search page. The goal of the search page was to narrow down a list of locations based on the type of study space. For example, if a user types "café" into the search bar, a filtered list containing cafes would be displayed. This feature allows the user to easily navigate to the preferred study space. We also decided to move the filter button from the home page to the search page, but due to time constraints, we were unable to make the features of the filters work.

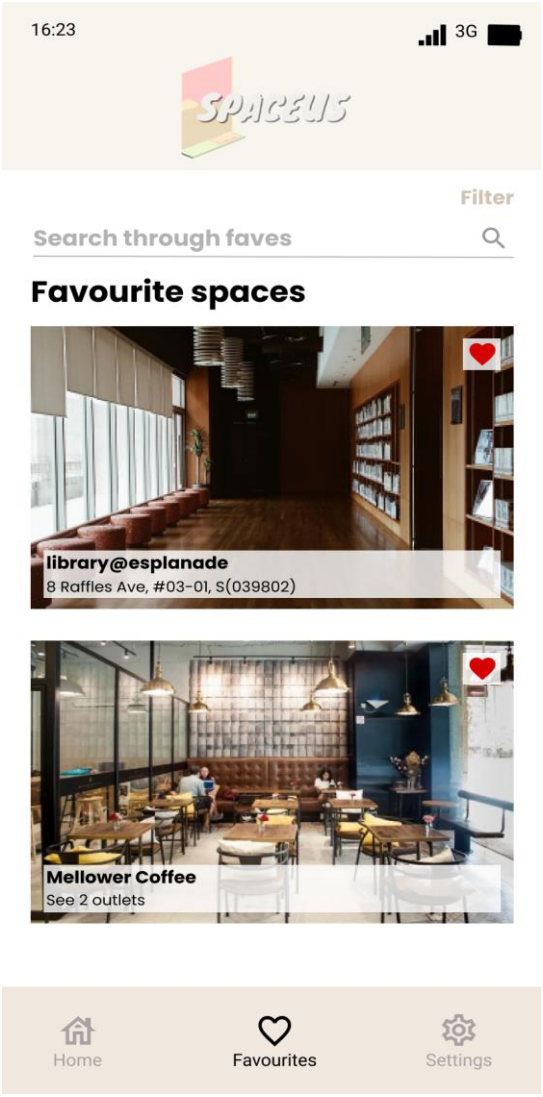


Fig 12. Favourites Page on Figma

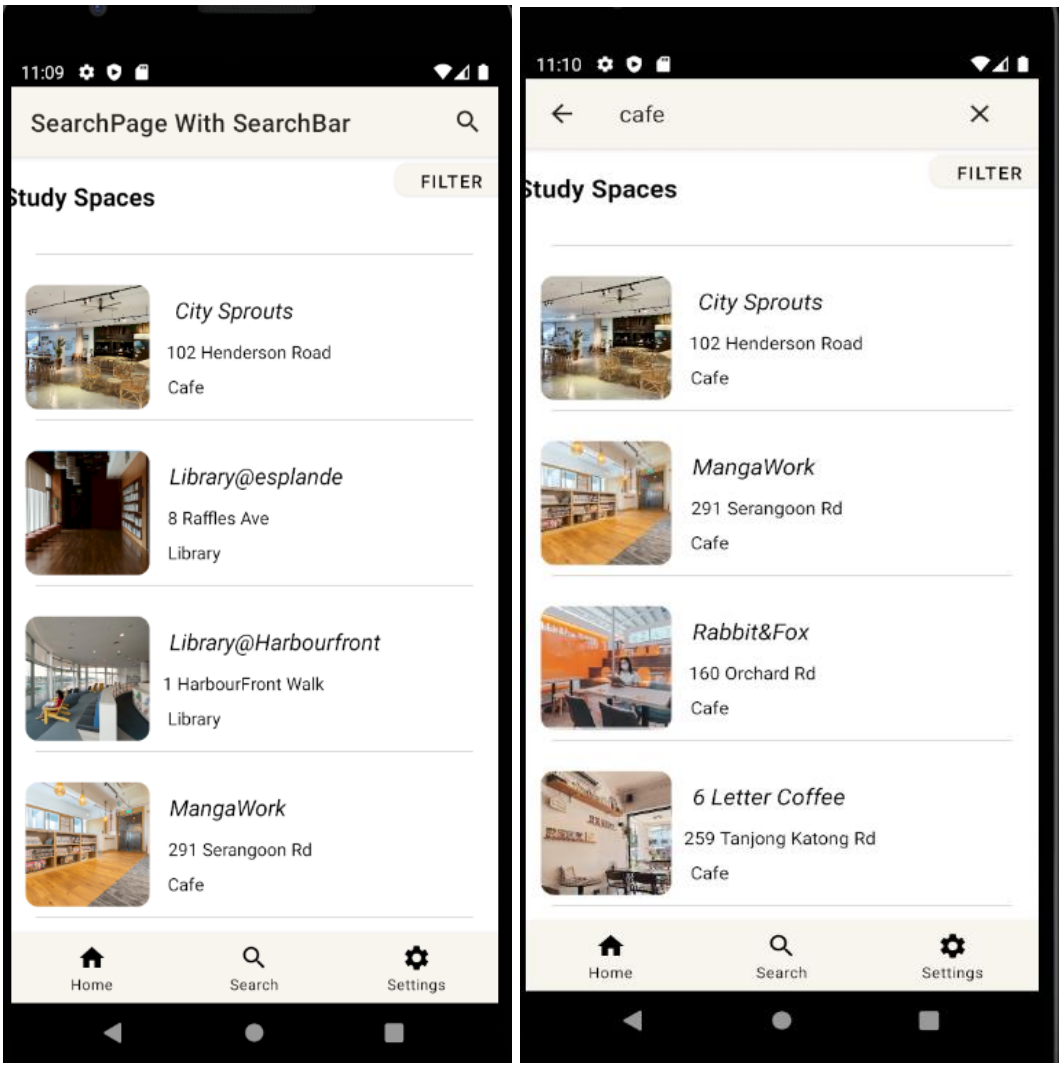


Fig 13 & 14. One of the versions of the Search Page which was unable to be implemented on Android Studio

Likewise, for the Settings page, it proved to be difficult to exactly replicate the Figma prototype on Android Studio. Adding to that, we realized that some parts of the Settings page were confusing. Thus, we decided not to include complicated details on the My Profile page in the updated prototype. As a result, we limited ourselves to the essentials. Thus, in the final version of the prototype, a user would be able to easily input and edit their profile information, as well as undo and save their changes.

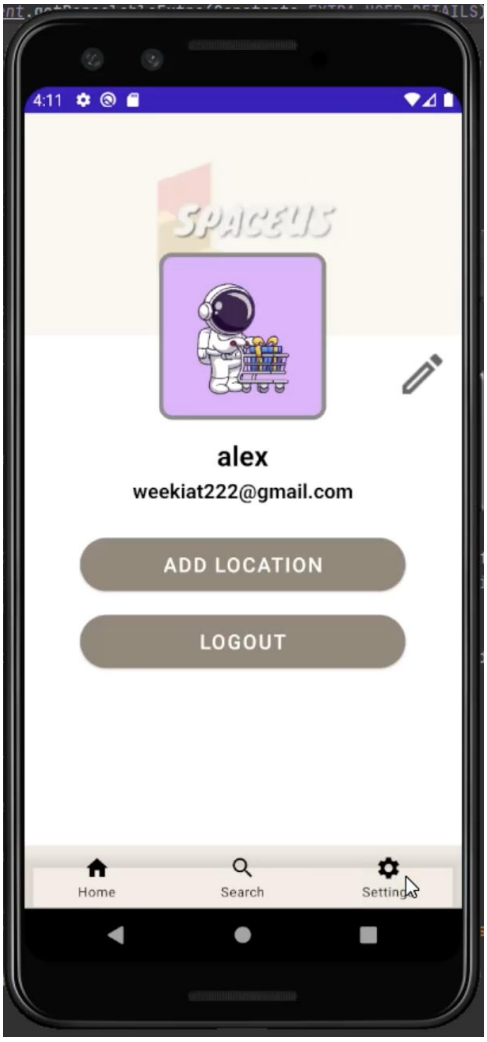


Fig 15. The Settings Page on Android Studio

The Create New Location page is another page that we discussed during the process of designing the application. This function is similar to the one found on Google Maps. It allows the user to recommend places that are not on the app, and once the legibility of a place is verified, it will be added to the application's list of places.

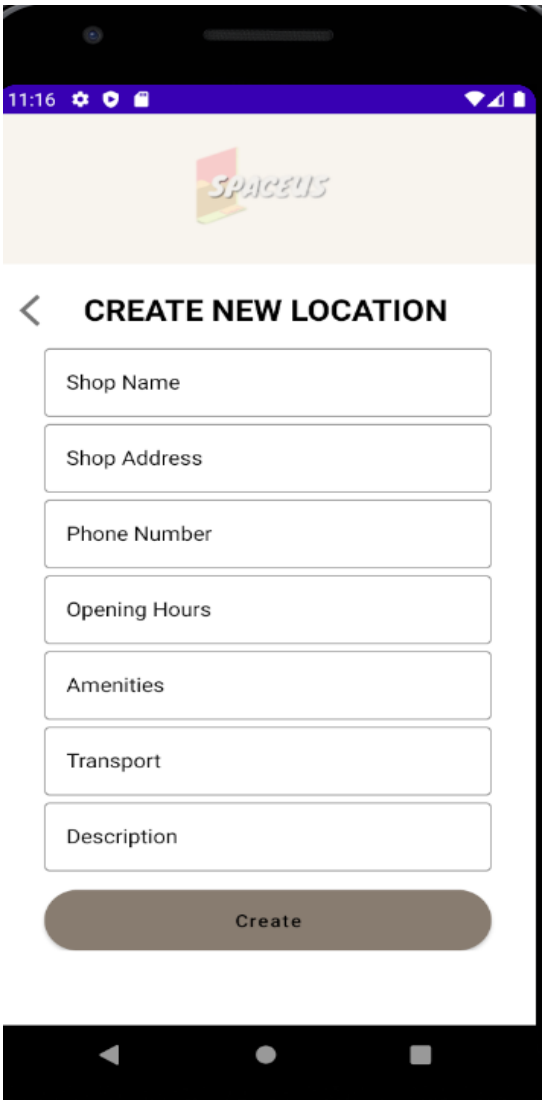


Fig 16. Create New Locations Page on Android Studio

The Specific Location page was a rather difficult one to combine. We had wanted to create a template for all the locations, but we were unable to do so due to the complexity of coding. The final design of the application was similar to the final Figma prototype design. Regrettably, we were not able to properly implement the function since it resulted in other functions crashing during the process of coding for the prototype. Since this XML file was hard coded in Android Studio, converting the hard coded file into a template proved to be tedious.

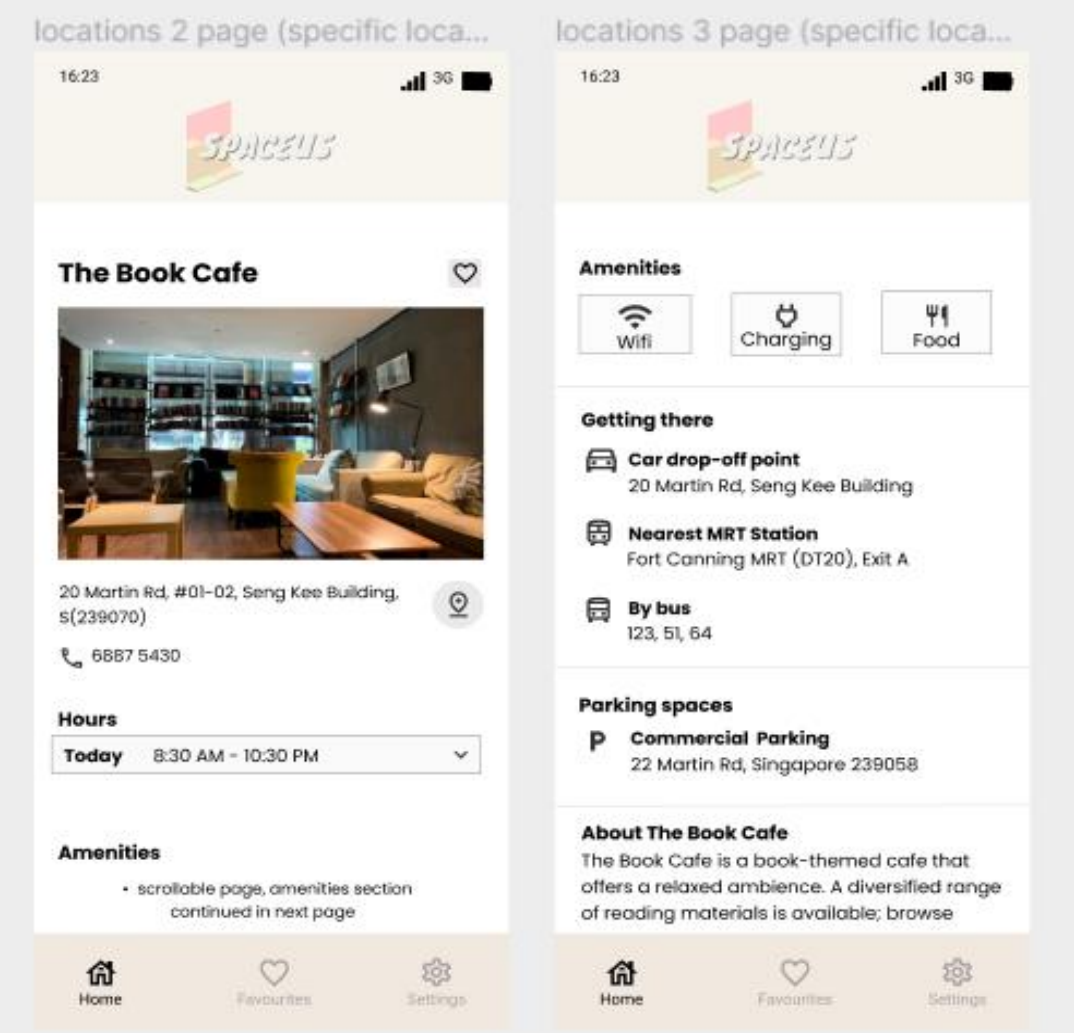


Fig 17 & 18. Specific Locations Page on Figma

After building our semi-working prototype on Android Studio, we decided to perform a second round of unit testing to evaluate the product properly by using the System Usability Scale (SUS) and tested a sample of 5 users. The list of the template questions are included below:

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

SUS Results:

Name	Odd Questions Score	Even Questions Score	Score
Zhang Jia Qi Jacinta	18	20	95
Chew En Qi	20	18	95
John Tang	17	20	92.5
Devansh Gupta	16	18	85
Joshua Lim	16	18	85

Users from a range of ages, including both students and working adults, were satisfied with the product and found it intuitive to use. However, the missing functions were concerning to them, which lowered the score significantly for some users. Overall, users found that the product was something useful to them and a few mentioned during the testing that they would be looking forward to it being an actual software available for use in Singapore.

Our takeaways from creating our project were that low fidelity prototypes allowed us to be more exploratory and opened the door to various innovative ideas. Contrarily, high fidelity prototypes enabled us to exclude unnecessary features while focusing on the crucial details.

Design

The project we are building is Spaceus (pronounced *spay-c-us*). Spaceus is a study spot locator Android application based in Singapore, aiming to help locals find places to study/work at. The designs of Spaceus have gone through multiple changes.

Task Flow Diagrams

Improvements have been made to our task flow diagrams after our Midterm Proposal; some functions were added or removed and some processes were slightly changed. As such, our workflow diagrams had to be edited to better reflect the process of our application.

In the Create Account process, a slight change of the input fields required to be filled by the user was made and is reflected in the task flow. In this task flow diagram, blue rectangles are used for events and purple diamonds are used to present a decision point.

The process begins with the user entering the application. Upon entering the app, the Login page will be displayed by default; if the user does not have an account, they will click “Create now” to navigate to the Create an Account page where they will fill up the form with their details. All users creating an account must select the terms, conditions and privacy policy checkbox before they are allowed to submit their form. If both of these conditions, valid email and checked checkbox, are met, then the account will be created. If either one of these conditions is not met, the user will be required to complete the form properly before proceeding.

If the user already has an account, then they will enter their details and the details will be validated to check if the account exists or if the password entered is valid. If either condition is not met, the user will have to enter their details again. Upon entering the correct email and password, the user will then be logged into their application.

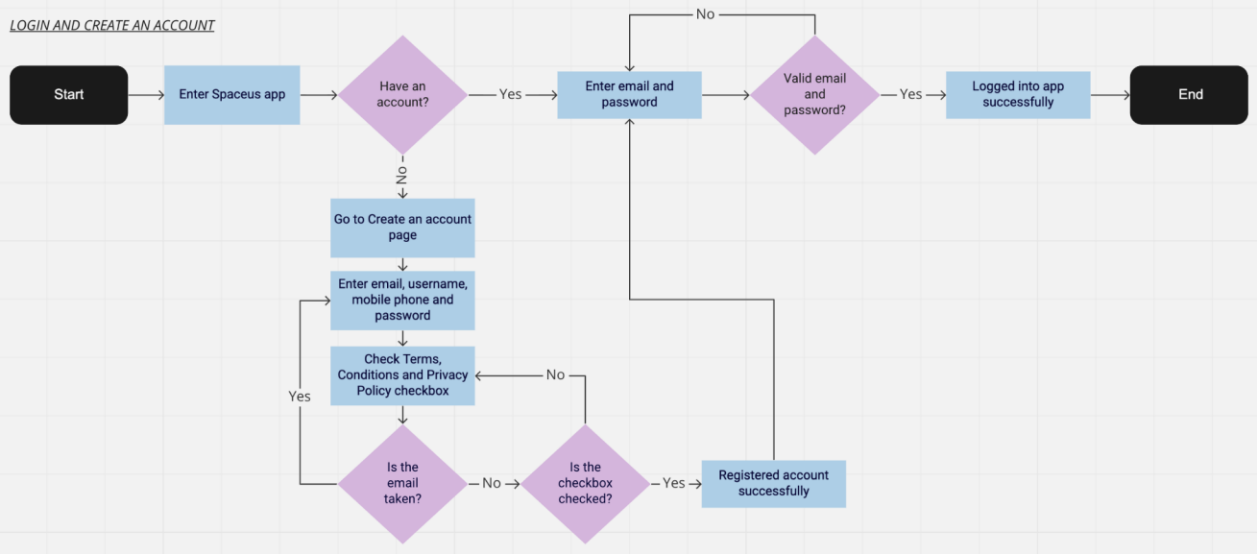


Fig 19. Updated task flow for Login and Create an account process

For the forgot password process, yellow rectangles are used to represent an event and green diamonds are used to present a decision point and the process begins with the user entering the application and being at the login page. If the user has not forgotten their password, they will input their details and validation will be done. If all conditions are met, the user will be successfully logged in to the application. If not, they will have to re-enter their credentials.

If the user has forgotten their password, they will click on the “Forgot password?” prompt and it will take them to the password reset page where they will submit their registered email. After submission, an email with a password reset link will be sent to the email address they inputted. The user will click on the link and be redirected to a browser where they will input a new password.

The new password will be saved and the user has to return to the login page to login using their new password. If the email address and password are not valid, then the user will be required to enter them again. If the email and password are valid, then the user will be successfully logged into the system.

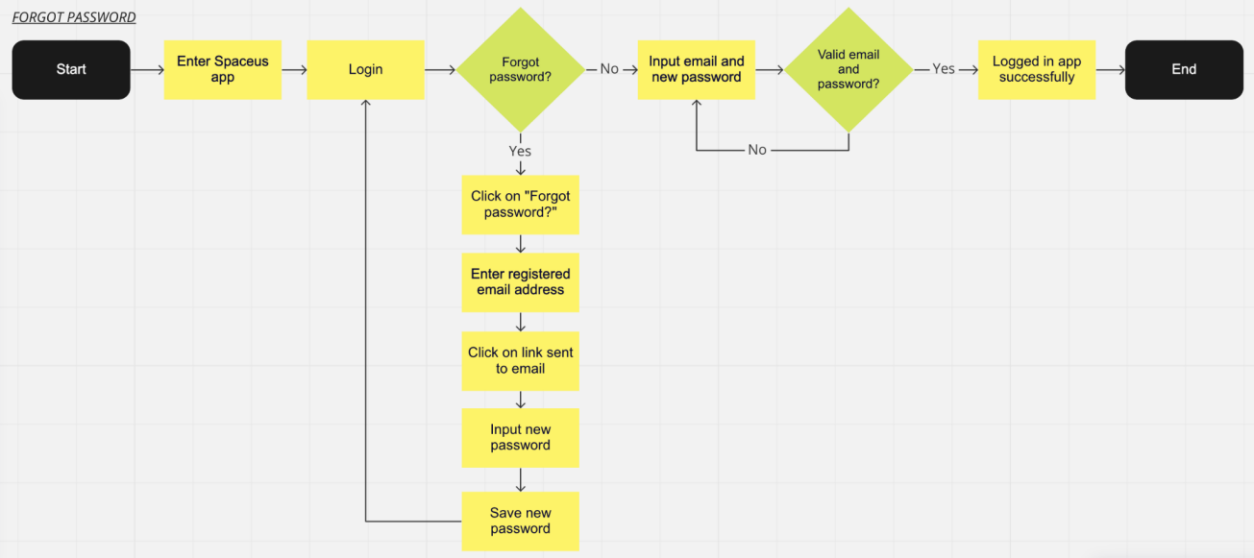


Fig 20. Task flow for Forgot Password process

Under the Create New Location process, the green rectangles represent the events in the process and the blue diamonds represent the decision points in the process. This process begins with the user entering the application (assuming they successfully login as well) and going to the Settings page. The user then clicks on the “Add location” button where they will be directed to the Create New Location page.

Here, they will fill up the form with the details of the location they wish to add. If all required fields are not filled in, the user will have to enter the information. If all required fields are filled, the form will be submitted and checked by the admin before the location is approved or rejected. If the location submitted is rejected, the process will end. If the location submitted is approved, the location will be added to the Home page as a new location and the process ends here.

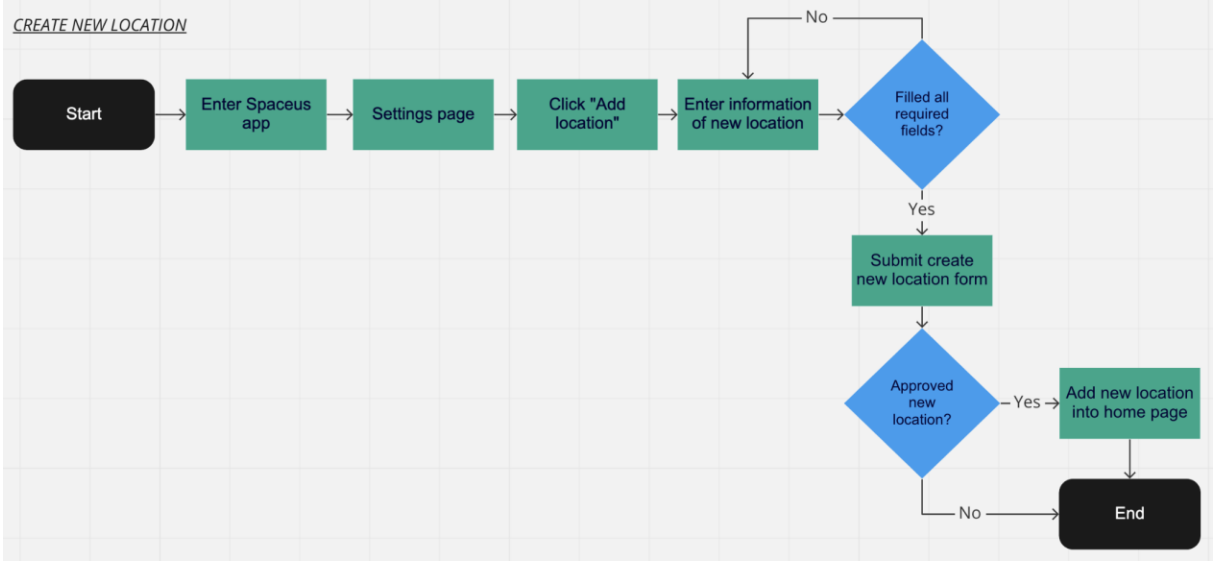


Fig 21. Task flow for Create New Location process

Update Profile process begins with the user entering the application (assuming they successfully login as well) and going to the Settings page. The user will click on the edit button and be brought to their profile page. There, the user will make changes to fields that allow change/editing i.e. contact number. The user will not be able to make any changes whatsoever to the username and email fields.

If any compulsory fields are left empty, the user will not be able to update their profile and will have to fill them. If all compulsory fields are filled, then and only then will the user's profile be updated and the process ends.

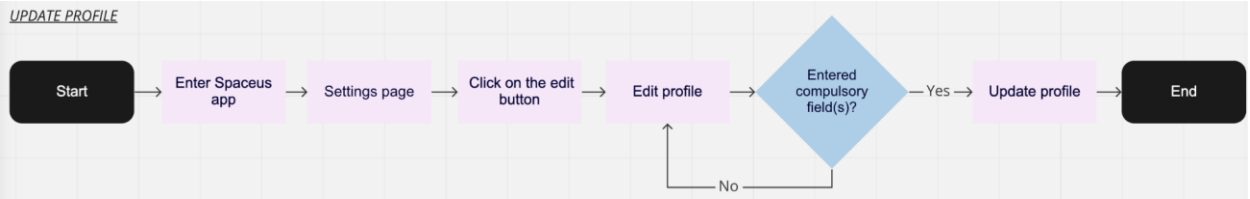


Fig 22. Task flow for Update Profile process

Improvements to design in Figma

After we carried out the survey, we improved our application's design based on potential users' feedback and suggestions. For the design of our application, we used Kotlin to design them in Android Studio, creating the design/layout in an XML file.

For the Create Account page of our application, we initially removed the username field and the back button of the page. As our application is mainly focusing on allowing users to find study spaces, requiring the user to input a compulsory username when they are creating an account would be an extra step for them and seemed redundant as well. The back button was removed and instead, a Login page prompt below the 'Create' button, that would redirect the user should they want to return to the Login page, was created. This change made the 2 pages appear more seamless and cohesive.

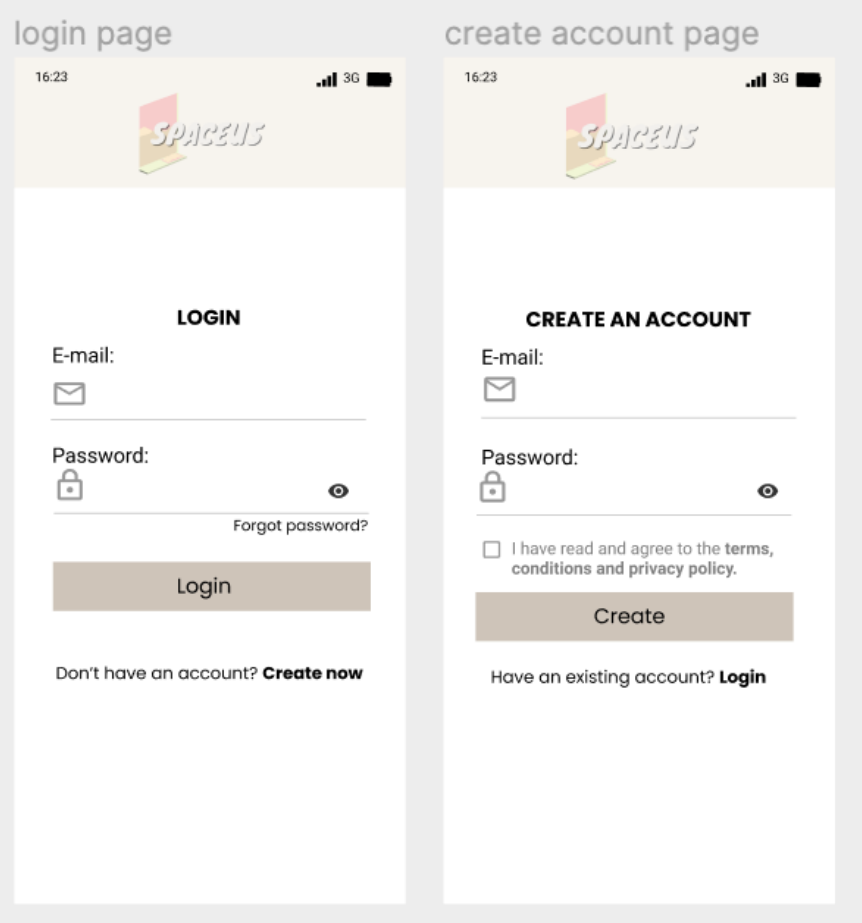


Fig 23. Updated design of Login and Create Account pages

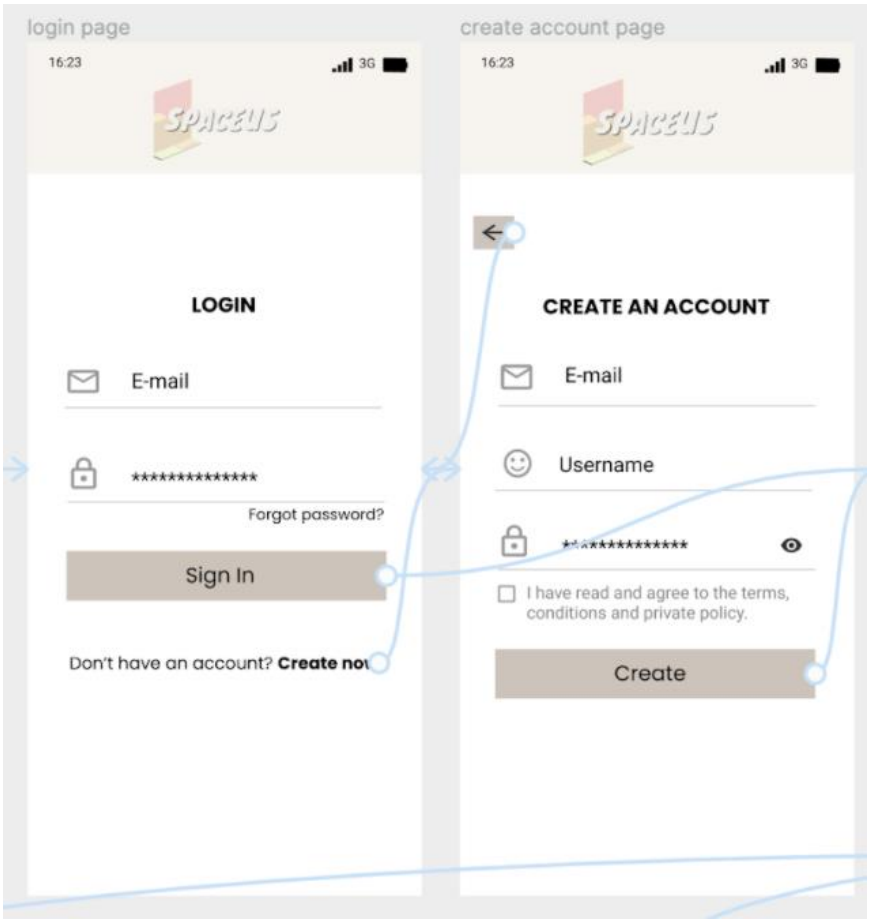


Fig 24. Previous design

Some users gave feedback that the navigation bar at the bottom of the pages was too complicated, thus we simplified it from 5 items to 3 items. We combined pages that were similar like the Home and Favourites page as well as the Profile and Settings page; users mentioned the redundancy of these pages when trying to navigate our prototype as 4 separate pages as these pages have features and functions that are similar which would be better combined.

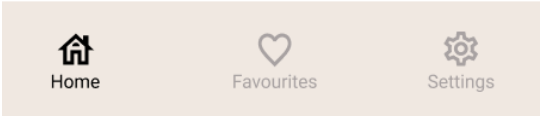


Fig 25. Updated design of navigation bar (Version 2)

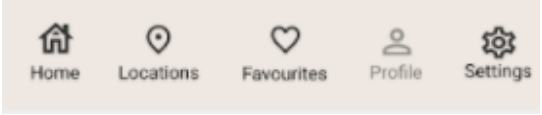


Fig 26. Old navigation bar in Figma (Version 1)

With the Home page and Locations page combined, the updated design of the Home page now displays all locations and will be a scrollable page. We also added an expandable Regions section at the top of the page to allow users to find study spaces based on the 8 regions in Singapore; when the users select the ‘Central’ option, all locations under the Central region will be filtered and displayed to the user. Another form of filtering allows the user to select from a list of filters; these filters consist of amenities the study spaces provide i.e. WiFi and ways the user can access the location i.e study space is located within walking distance of MRT (Singapore’s railway network) or user can take a bus from the MRT.

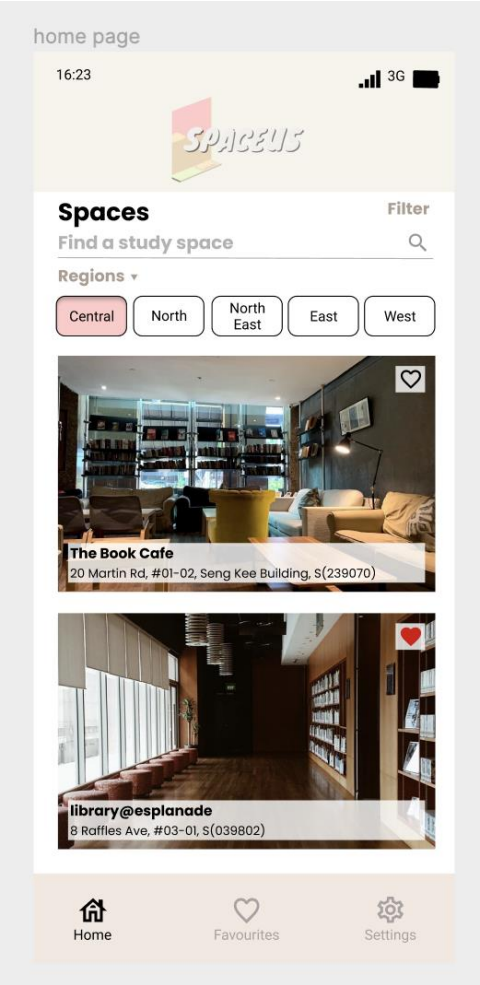


Fig 27. Updated Home page (combined with Locations page from previous design)

Changes to the filter choices and design of the filter page were made as there were users who suggested the blurred background of the original filter page to be changed to a solid colour instead as it would provide ease of viewing for them. Filter options will be simplified to allow users an easier time when using our app; fewer filters will require users to spend less time reading what type of filters they can choose from and they will be less likely to be confused or intimidated by the number of filters to choose from.

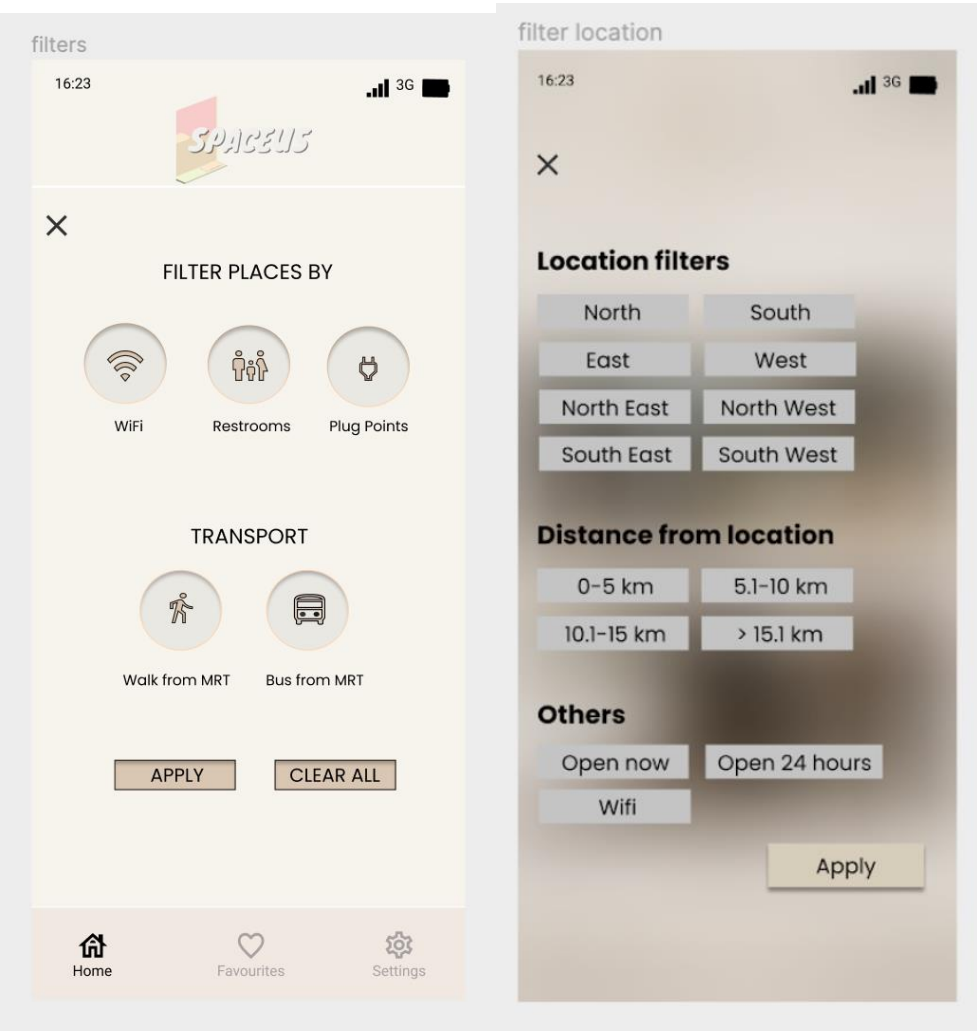


Fig 28 & 29. Updated filters and design of filter page (left) VS Previous design of filter page (right)

Profile and Settings page were also merged to become one page under the Settings page. The updated Settings page allows users to view and edit their profile details; users will also be able to change their settings options from the menu shown below. The design of the new Settings page will also allow the user to have a better experience navigating and using our application.

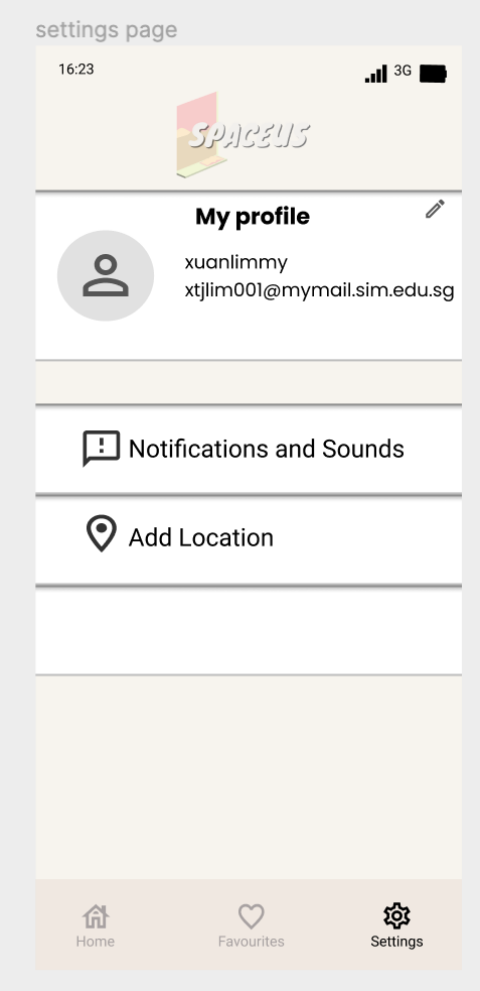


Fig 30. Updated Settings page

Among the changes to our existing design in Figma, we had also created the design and layout for how the specific location listing will look; this page will be displayed to the user upon selecting a study space in the Home page and will be a scrollable page for the user to navigate through. We felt the need for a specific location page because we considered it to be a basic necessity in most applications; since there is a home page displaying all locations, it is only right to have a specific location page to show the location's details.

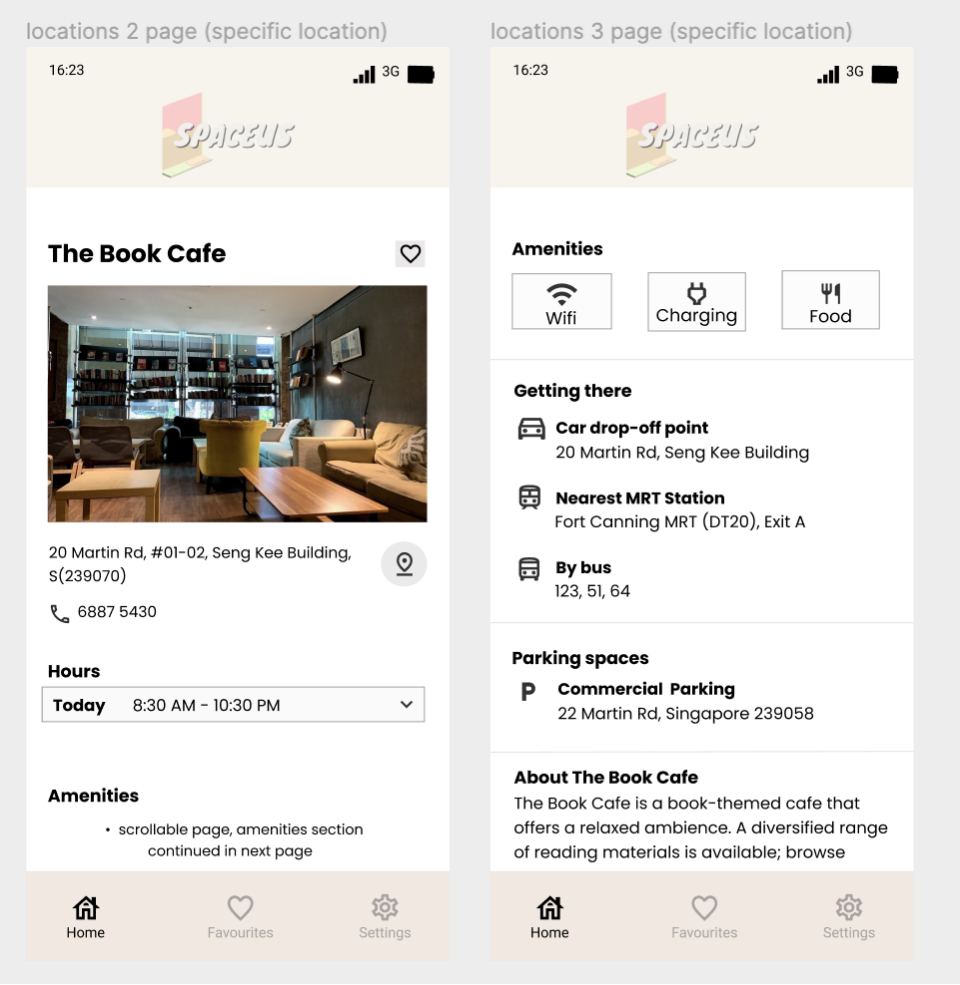


Fig 31. Specific location listing

Differences between final application and Figma design

After revamping the design and adding additional pages into the design prototype in Figma, we started creating the application in Android Studio, using Kotlin. Some components of the app ended up being different from the updated prototype design in Figma as there were some constraints throughout the development of our application; some components were added back into our application when we were developing our application and some design elements of the application could not be replicated exactly to the design prototype in Figma.

We have had to change the navigation bar of our application twice due to different circumstances; the 1st change occurred after our user survey was done when some users mentioned that the original navigation bar in our Figma prototype was too complicated, and the 2nd change occurred during the development of our application when we had difficulty integrating some functions of the application.

As we were not able to integrate some of the functions we intended to include in the application, we had to replace the Favourite tab/functionality with the Search tab/functionality instead.

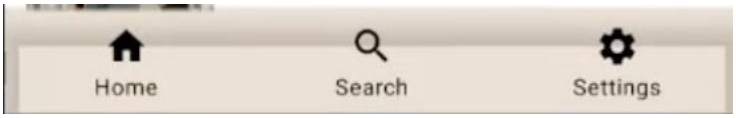


Fig 32. Final navigation bar of application in Android Studio Emulator (Version 3)

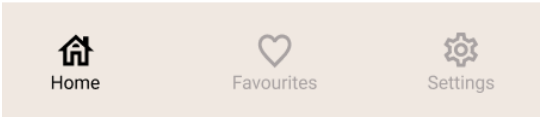


Fig 33. Updated design of navigation bar (Version 2)

Initially, our final design prototype excluded the username field and did not have a contact number field, but we decided to include those 2 fields into our app because we were not familiar and have no experience with Firebase and coding for web development. This resulted in the addition of the 2 fields (username and contact number) to the Create Account page in order to make our Profile page viable.

Besides the addition of the username and contact number fields, the design of the Create Account page has also been changed slightly, mainly the input fields and button shape. We decided to use boxes instead of simply a line (from our previous prototype) for user input as it was neater; with the addition of the 2 fields, the Create An Account form would look more cohesive and in order.

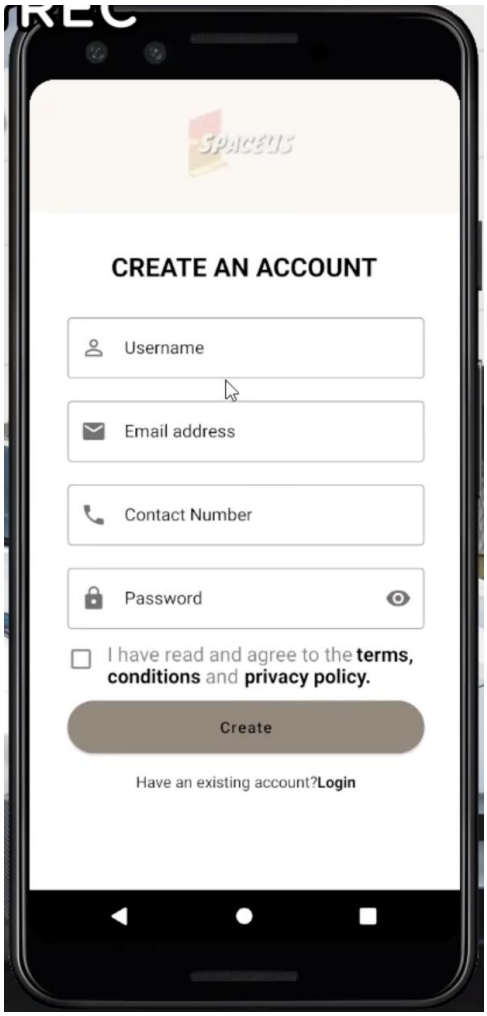


Fig 34. Final look of Create Account page of Spaceus application in Android Studio emulator

Initially, we had intended to have our Terms & Conditions and Privacy Policy to be on a single scrollable page but we had decided to separate both sections as it looked

cluttered when both sections were combined into a single page, considering how much text/information was in each section, and also allowed for a better reading experience for the users.

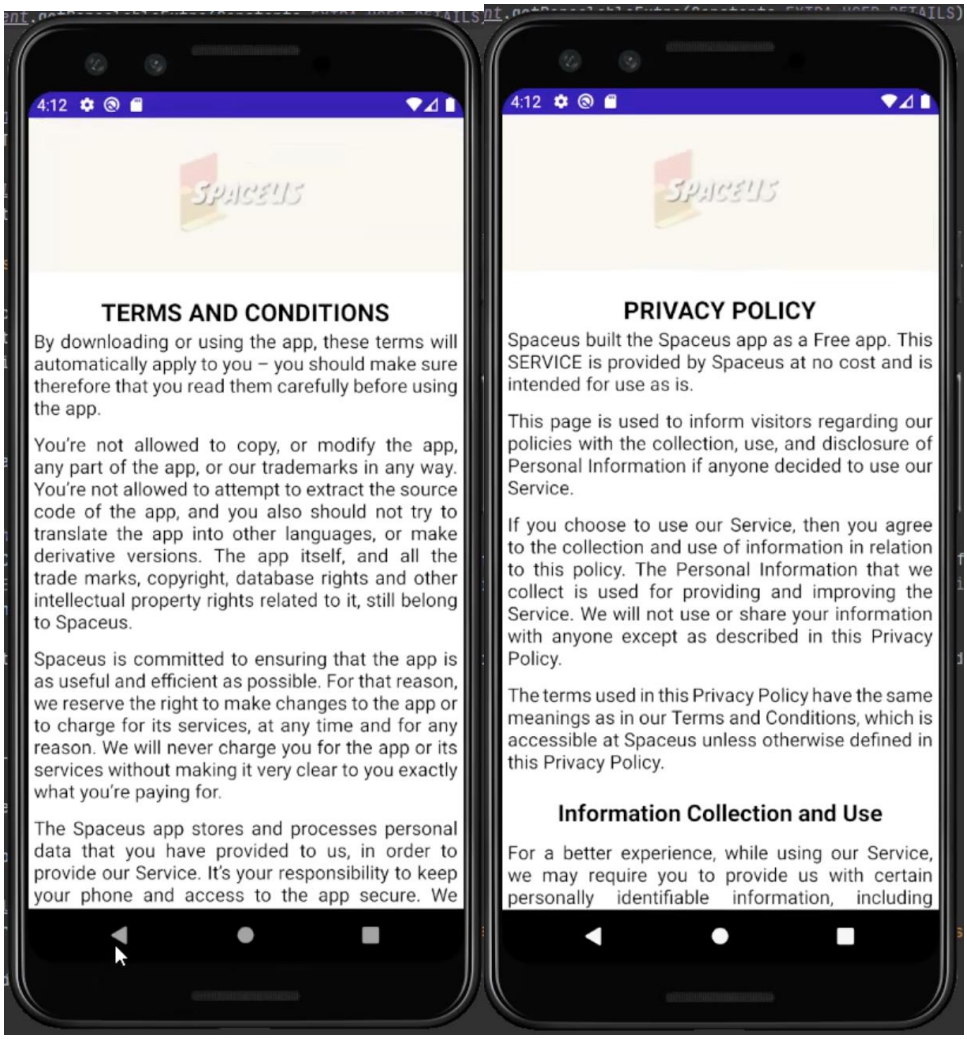


Fig 35. Final look of Terms and Conditions page (left) and Privacy Policy page (right)

Besides the design change for the Create Account page, we also made slight changes to the Login page; we used boxes instead of lines for input fields (as shown in the

prototype design in Figma). The box input field provided a better distinction between both fields on the page as compared to our design prototype in Figma.

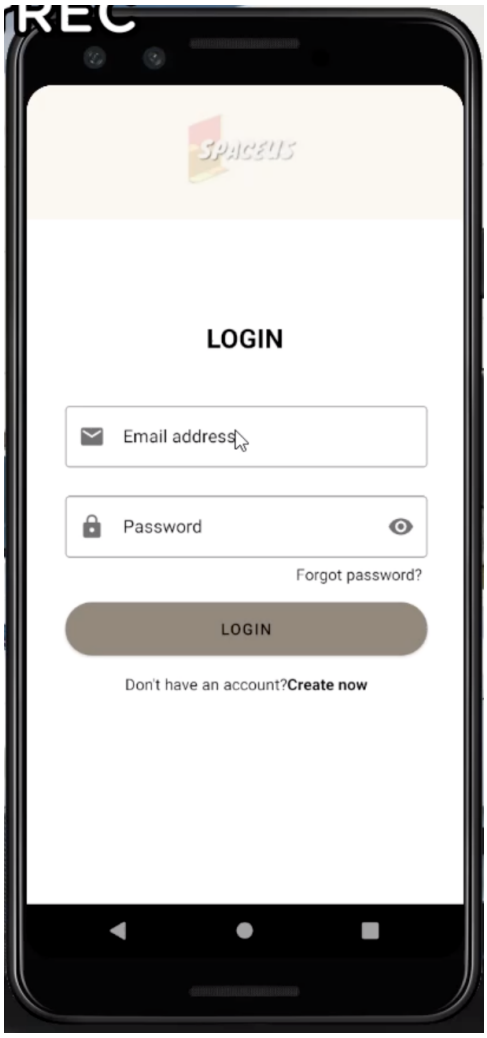


Fig 36. Final look of Login page of Spaceus application in Android Studio emulator

The Forgot Password page is a page we did not design in Figma as it was not the main function like Login or Create Account. We wanted to focus on designing and prototyping

pages of the application’s main functions as they were a higher priority as compared to the extra functions like Forgot Password.

From the beginning (in our midterm proposal), as seen in both Forgot Password task flow (midterm and final) we planned to have an email sent to the user if they forgot their password and then click on a link in the sent email to change their password. In this final application and proposal, we managed to accomplish this and stick to the initial design we came up with after we decided to add the Forgot Password functionality into the application. As seen in Fig 20, we have also managed to stick to the workflow of this process.

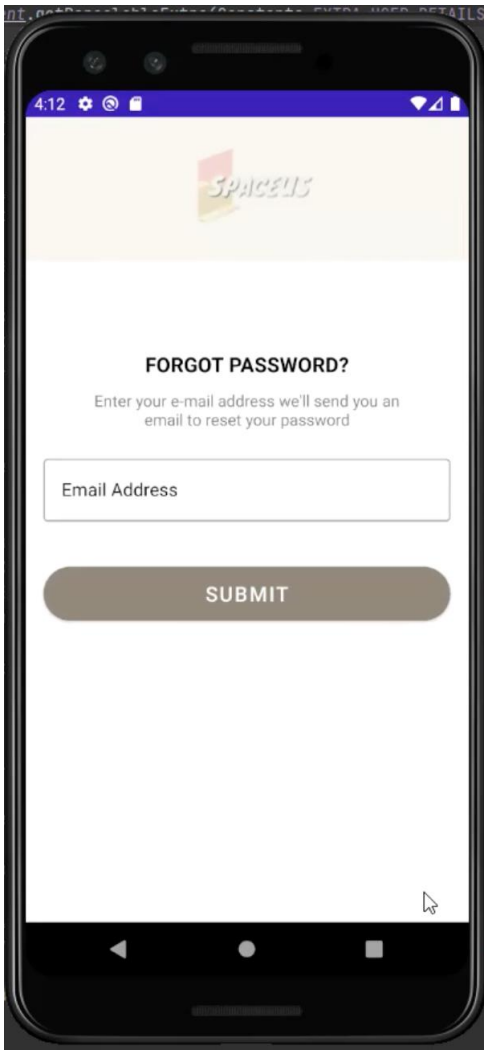


Fig 37. Final look of Forgot Password page in Android Studio Emulator

Our Home page has some minor differences to the Figma design; the picture of each location only takes up the left side of the listing as compared to taking up the entire

listing in Figma. The Regions portion was removed as we were not able to integrate that portion due to our unfamiliarity with Kotlin. The other difference to the Home page is that there is no favourite button for users to add a particular location to their favourites. We also did not have enough advanced knowledge in Kotlin to be able to integrate these functions into our application. Because of this, we had to change our direction and make some changes to the Home page like removing the search bar from the Home page and making a separate page for it.

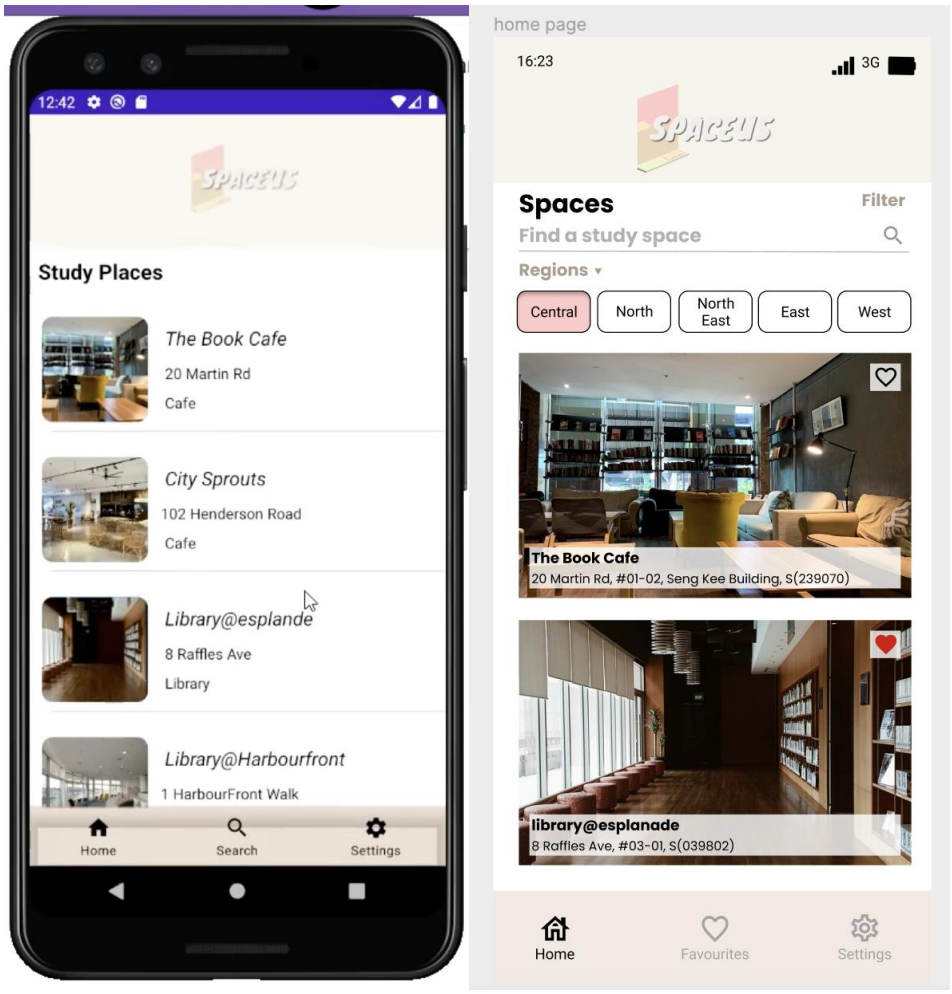


Fig 38. Final look of Home page in Android Studio Emulator (left) VS design prototype in Figma (right)

The Settings page for our final application also has a different design as compared to the prototype design in Figma as we were not able to replicate the exact layout and

design. Besides not being able to replicate the exact design in Figma, we also felt that the change in design was necessary in order to not make the Settings page confusing.

In the updated design prototype in Figma (after user testing and survey was done), we felt that the My Profile section page (in Figma) of the Settings page might confuse users and there was not much relevant information that could be added to the My Profile section. As such, the team decided to not include a huge portion of the page for other profile details; we only displayed profile details such as the users' username, email, contact number and gender. When the users decide to update their profile details, the same fields and details will also be displayed to them and the newly inputted details will be saveable upon any changes made. Users will also have the option to cancel the process of editing fields or cancel whatever changes they have made to their profile details.

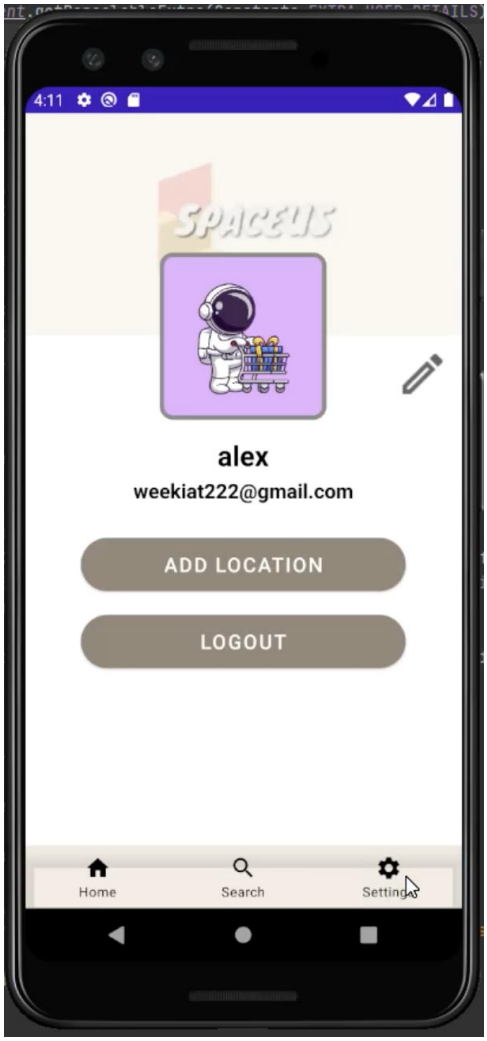


Fig 39. Final look of Settings page of Spaceus application in Android Studio emulator

The Create New Location page is another page we did not design during our prototyping phase as we felt the need to only design pages for main functions i.e. Login,

Create Account and Home page. These functions' pages were prioritised over other pages as they are basic functions of any application; most applications have a Login, Create Account and Home page at the very least. As seen in Fig 3, the process will require the user to fill up the fields in the form before the location can be submitted. The fields we chose are all the basic information needed about the location like its name and address.

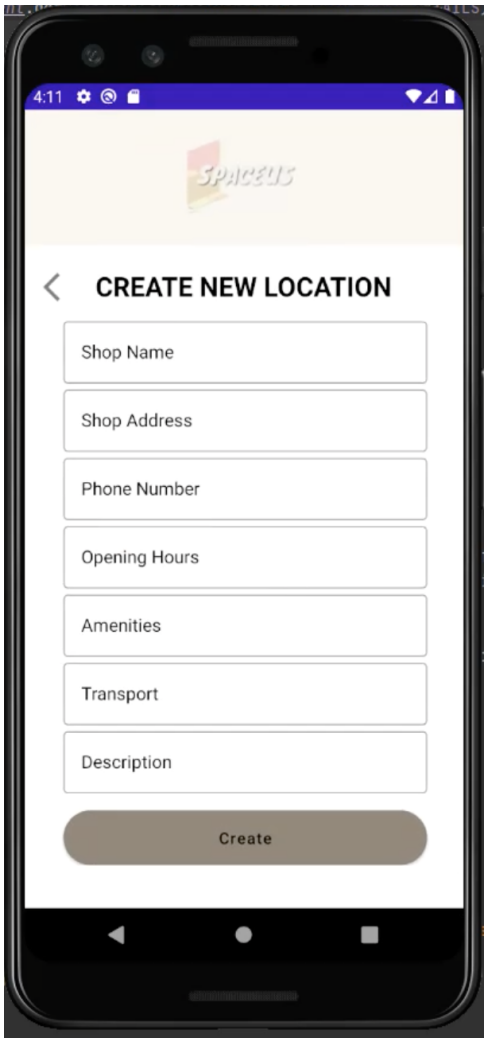


Fig 40. Final look of Create New Location page in Android Studio Emulator

System Development

System development is the process of defining, designing, testing, and implementing a new software application or program. We used Android Studio to develop Spaceus. Android Studio is the official Integrated Development Environment (IDE) for Android application development. In Android Studio, we used Kotlin as our programming language as it is a modern statically typed programming language used by more than 60% of professional Android developers. It can help to increase productivity, developer satisfaction, and code security. Besides Kotlin, we also used Firebase as our database to store our users' details and location details. Firebase is a platform developed by Google for creating mobile and web applications. It provides developers with various tools and services to help them develop high-quality applications, expand the user base and make profits.

Step 1: Create Splash Page, Login and Register Page

First and foremost, we create a project and name it "spaceUs". After that, we need to create some activities which are SplashActivity, LoginAcitivty, ForgotPasswordActivity, RegisterActivity, TermActivity, and PrivacyActivity.

For the SplashActivity, the main function is used to display some information about our company logo, company name, etc. First, we need to edit the activity_splash.xml and add our logo image. After editing the XML, we need to write the code inside the SplashActivity.kt. We need to make sure that the status bar can be hidden to make it full screen and we also use the postDelayed method to send a message with a delayed time. The postDelayed method can be seen in the diagram below:

```
class SplashActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_splash)

        // This is used to hide the status bar and make the splash screen as a full screen activity.
        @Suppress( ...names: "DEPRECATION")
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.R) {
            window.insetsController?.hide(WindowInsets.Type.statusBars())
        } else {
            window.setFlags(
                WindowManager.LayoutParams.FLAG_FULLSCREEN,
                WindowManager.LayoutParams.FLAG_FULLSCREEN
            )
        }

        //we use the postDelayed(Runnable,time)method to send a message with a delayed time
        @Suppress( ...names: "DEPRECATION")
        Handler().postDelayed(
            {
                //Launch the Main Activity
                startActivity(Intent( packageContext: this@SplashActivity, LoginActivity::class.java))
                finish()
            },
             delayMillis: 2000
        )
    }
}
```

Fig 41. postDelayed method

For the RegisterActivity, we have used the Firestore database which is the key feature of Firebase to store our user details. Firestore database caches the data that your app is using, so the app can write, read, listen, and query data even when the device is offline. We connected our project with the Firestore, so we can see the user details on the Firebase website. Besides, we also used the Firebase Authentication to create a register for a user with an email and password. After doing the connection, we have created a data class called “User” in order to contain the fields for accessing them. After that, we have created a function called “validateRegisterDetails” to validate that all the fields and checkboxes are not empty. We have written unit tests to test if the confirmed password is equal to the password or not. After done the unit tests, we have removed the unit tests code. The validateRegisterDetails function can be seen in the diagram below:

```
private fun validateRegisterDetails():Boolean{
    return when {

        TextUtils.isEmpty(et_username.text.toString().trim { it <= ' ' }) -> {
            showErrorSnackBar("Please enter username.", errorMessage: true)
            false
        }

        TextUtils.isEmpty(et_register_email.text.toString().trim { it <= ' ' }) -> {
            showErrorSnackBar("Please enter email.", errorMessage: true)
            false
        }

        TextUtils.isEmpty(et_register_password.text.toString().trim { it <= ' ' }) -> {
            showErrorSnackBar("Please enter password.", errorMessage: true)
            false
        }

        TextUtils.isEmpty(et_confirm_password.text.toString().trim { it <= ' ' }) -> {
            showErrorSnackBar("Please enter confirm password.", errorMessage: true)
            false
        }
        et_register_password.text.toString().trim{ it <= ' '} != et_confirm_password.text.toString()
            .trim{ it <= ' '} ->{
            showErrorSnackBar("Password and confirm password does not match.", errorMessage: true)
            false
        }

        !term.isChecked -> {
            showErrorSnackBar("Please agree terms and condition.", errorMessage: true)
            false
        }

        else -> {
            //showErrorSnackBar(resources.getString(R.string.register_Successful),false)
            true
        }
    }
}
```

Fig 42. validateRegisterDetails function

Besides, we also created a class called “FirestoreClass”. In the FirestoreClass, we create a function called “registerUser” and pass the parameters which are RegisterActivity and User class. The main function for the registerUser function is to set the user details in the Firestore database. We also have used unit tests to test if the registration is a success or a failure. After all the work can run as normal, we have removed the unit tests code. The registerUser function can be seen in the diagram below:

```
fun registerUser(activity: RegisterActivity, userInfo:User){
    mFireStore.collection(Constants.USERS)
        //Document ID for users fields.Here the document it is the User ID.
        .document(userInfo.id)
        .set(userInfo, SetOptions.merge())
        .addOnSuccessListener { it: Void!
            activity.userRegistrationSuccess()
        }
        .addOnFailureListener { e ->
            activity.hideProgressDialog()
            Log.e(activity.javaClass.simpleName, msg: "Error while registering the user.",e)
        }
}
```

Fig 43. registerUser function

We created another function called “registerUser” in RegisterActivity.kt to run the registration. If the validate function is valid, it will create an instance and a register for a user with an email and password. If the registration is successfully done, it will redirect the user to the login page, else it will print the error message. The registerUser function can be seen in the diagram below:

```
private fun registerUser(){

    //Check with validate function if the entries are valid or not.
    if(validateRegisterDetails()){

        showProgressDialog( text: "Please Wait ...")
        val email: String = et_register_email.text.toString().trim{it <= ' '}
        val password: String = et_register_password.text.toString().trim{it <= ' '}

        //Create an instance and create a register a user with email and password.
        FirebaseAuth.getInstance().createUserWithEmailAndPassword(email,password)
            .addOnCompleteListener(
                OnCompleteListener<AuthResult> { task ->

                    //If the registration is successfully done
                    if(task.isSuccessful){
                        //Firebase registered user
                        val firebaseUser:FirebaseUser = task.result!!.user!!
                        val user = User(
                            firebaseUser.uid,
                            et_username.text.toString().trim{ it <= ' ' },
                            et_register_email.text.toString().trim{ it <= ' ' }
                        )

                        FirestoreClass().registerUser( activity: this@RegisterActivity,user)
                        val intent = Intent( packageContext: this@RegisterActivity, LoginActivity::class.java)
                        startActivity(intent)
                        //FirebaseAuth.getInstance().signOut()
                        //finish()
                    }else{
                        hideProgressDialog()
                        showErrorSnackBar(task.exception!!.message.toString(), errorMessage: true)
                    }
                })
    }
}
```

Fig 44. registerUser function

Lastly, we need to call out the registerUser function when the user clicks the register button. We also link the term activity and privacy activity when the user clicks the text

view. Users also can back to the login page by clicking the text view at the bottom. The click listener for each button or text view can be seen in the diagram below:

```
textViewLogin.setOnClickListener{ it: View!
    //val intent = Intent(this@RegisterActivity,LoginActivity::class.java)
    //startActivity(intent)
    onBackPressed()
}

btnRegister.setOnClickListener{ it: View!
    registerUser()
}

textView_terms.setOnClickListener { it: View!
    val intent = Intent( packageContext: this, TermsActivity::class.java)
    startActivity(intent)
}

textView_terms2.setOnClickListener { it: View!
    val intent = Intent( packageContext: this, TermsActivity::class.java)
    startActivity(intent)
}

privacy_policy.setOnClickListener { it: View!
    val intent = Intent( packageContext: this, PrivacyActivity::class.java)
    startActivity(intent)
}
```

Fig 45. Click listener for each button

For the LoginActivity, we created a function called “validateLoginDetails” to validate that all the fields are not empty. After that, we create another function called “getUserDetails”.

In the getUserDetails function, the main function is to get the user details in the Firestore database. The getUserDetails function can be seen in the diagram below:

```
fun getUserDetails(activity: Activity){
    //Here we pass the collection name from which we wants the data.
    mFireStore.collection(Constants.USERS)
        // The document id to get the Fields of user.
        .document(getCurrentUserID())
        .get()
        .addOnSuccessListener { document ->
            Log.i(activity.javaClass.simpleName,document.toString())

            val user = document.toObject(User::class.java)!!

            val sharedPreferences = activity.getSharedPreferences(Constants.MYSHOPPAL_PREFERENCES, Context.MODE_PRIVATE)
            val editor : SharedPreferences.Editor = sharedPreferences.edit()
            editor.putString(Constants.LOGGED_IN_USERNAME,"${user.username}")
            editor.putString(Constants.EMAIL,"${user.email}")
            editor.apply()

            when(activity){
                is LoginActivity ->{
                    activity.userLoggedInSuccess(user)
                }
            }

            //END
        }
        .addOnFailureListener { e ->
            when (activity){
                is LoginActivity ->{
                    activity.hideProgressDialog()
                }
            }

            Log.e(activity.javaClass.simpleName, msg: "Error while getting user details",e)
        }
}
```

Fig 46. getUserDetails function

The userLoggedInSuccess function is called out inside the getUserDetails function. The main function is when the user successfully logs in to our app, it will redirect to the dashboard activity. The userLoggedInSuccess function can be seen in the diagram below:

```
fun userLoggedInSuccess(user: User){
    hideProgressDialog()

    val intent = Intent( packageContext: this@LoginActivity, UserProfileActivity::class.java)
    intent.putExtra(Constants.EXTRA_USER_DETAILS,user)
    startActivity(intent)
    finish()
}
```

Fig 47. userLoggedInSuccess function

After that, we created another function called “logInRegisteredUser” in LoginActivity.kt. If the validate function is valid and the user’s email and password are matched with the Firebase Authentication, it will run the getUserDetails function, else it will print the error message. The logInRegisteredUser function can be seen in the diagram below:

```
private fun logInRegisteredUser(){

    if(validateLoginDetails()){

        //show the progress dialog.
        showProgressDialog( text: "Please Wait ...")

        //Get the text from editText and trim the space
        val email = et_login_email.text.toString().trim{it <= ' '}
        val password = et_login_password.text.toString().trim{it <= ' '}

        //Log-In using FirebaseAuth
        FirebaseAuth.getInstance().signInWithEmailAndPassword(email, password)
            .addOnCompleteListener{ task ->

                if(task.isSuccessful){
                    FirestoreClass().getUserDetails( activity: this@LoginActivity)
                }else{
                    hideProgressDialog()
                    showErrorSnackBar(task.exception!!.message.toString(), errorMessage: true)
                }
            }
    }
}
```

Fig 48. logInRegisteredUser function

Lastly, we need to call out the logInRegisteredUser function when the user clicks the login button. If the user forgot their password, they can click the “Forgot Password” text to the ForgotPasswordActivity. Users also can go to the registration page by clicking the text view at the bottom. The click listener for each button or text view can be seen in the diagram below:

```
override fun onClick(view:View?){
    if(view != null){
        when (view.id){

            R.id.ForgotPassword -> {
                val intent = Intent( packageContext: this@LoginActivity, ForgotPasswordActivity::class.java)
                startActivity(intent)
            }

            R.id.btnLogin -> {
                logInRegisteredUser()
            }

            R.id.textViewCreate -> {
                val intent = Intent( packageContext: this@LoginActivity, RegisterActivity::class.java)
                startActivity(intent)
            }
        }
    }
}
```

```
ForgotPassword.setOnClickListener(this)
btnLogin.setOnClickListener(this)
textViewCreate.setOnClickListener(this)
```

Fig 49 & 50. Click listener for each button

For the ForgotPasswordActivity, users can enter their email to reset their password. If the email is empty, it will show the error message, else it will check if the user email matches the firebase authentication or not. If the email is valid, then it will send an email to the user to reset their password and redirect to the login page. The forgot password method can be seen in the diagram below:

```
btnSubmit.setOnClickListener{ it: View!
    val email: String = input_forgotEmail.text.toString().trim{it <= ' '}
    if(email.isEmpty()){
        showErrorSnackBar("Please enter email.", errorMessage: true)
    }else{
        showProgressDialog( text: "Please Wait ...")
        FirebaseAuth.getInstance().sendPasswordResetEmail(email)
            .addOnCompleteListener { task ->
                hideProgressDialog()
                if(task.isSuccessful){
                    Toast.makeText( context: this@ForgotPasswordActivity,""Email sent successfully to reset your password! "",Toast.LENGTH_LONG).show()
                    finish()
                }else{
                    showErrorSnackBar(task.exception!!.message.toString(), errorMessage: true)
                }
            }
    }
}
```

Fig 51. Forgot password method

Step 2: Create dashboard activity

First and foremost, we use the bottom navigation bar for our home fragment, search fragment, and settings fragment. This is because it allows the user to switch to different

fragments easily and make the user aware of the different screens available in the app. So that, we create a menu called “bottom_nav_menu” and put the three items which are home, search and settings in the menu. After that, we add the menu in the activity_dashboard.xml. Besides, we also need to create three fragments which are HomeFragment, SearchFragment, and SettingsFragment. Now we need to code the DashboardActivity to connect everything to the application. First, create a function called “setCurrentFragment” that takes a Fragment as an argument and sets it in our activity_dashboard.xml. After that, we add a click listener to the items of the bottom navigation bar, so we can display the corresponding fragment when the user clicks the items. The DashboardActivity and setCurrentFragment function can be seen in the diagram below:

```
class DashboardActivity : BaseActivity() {
    private lateinit var bottomNavigation : BottomNavigationView
    private lateinit var binding: ActivityDashboardBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityDashboardBinding.inflate(layoutInflater)
        setContentView(binding.root)

        bottomNavigation = findViewById(R.id.nav_view)
        val homeFragment = HomeFragment()
        val searchFragment = SearchFragment()
        val settingsFragment = SettingsFragment()

        if (savedInstanceState == null) {
            val fragment = HomeFragment()
            supportFragmentManager.beginTransaction().replace(R.id.container, fragment, fragment.javaClass.simpleName)
                .commit()
        }

        bottomNavigation.setOnNavigationItemSelectedListener { it: MenuItem
            when(it.itemId){
                R.id.navigation_home -> setCurrentFragment(homeFragment)
                R.id.navigation_search -> setCurrentFragment(searchFragment)
                R.id.navigation_settings -> setCurrentFragment(settingsFragment)
            }
            true ^setOnNavigationItemSelectedListener
        }
    }

    private fun setCurrentFragment(Fragment: Fragment) {
        supportFragmentManager.beginTransaction().apply { this: FragmentTransaction
            replace(R.id.container,Fragment)
            commit()
        }
    }
```

Fig 52. DashboardActivity and setCurrentFragment function

For the HomeFragment, we use RecyclerView to display our location details. RecycleView is a ViewGroup added to the android studio as the successor of ListView. First, we need to add recyclerView to fragment_home.xml. Now we create a new layout

resource file called “list_item.xml” for designing our CardView layout. After that, we create a Kotlin class called “Location” to hold the information of every item which we want to show in the recycler view. We also need to create an Adapter Class. In this class, we need to write the three important functions which are onCreateViewHolder(),onBindViewHolder(), and getItemCount().

- onCreateViewHolder(): This function is to set the views to display the items.
- onBindViewHolder(): This function is used to bind the list items.
- getItemCount(): This function is to return the count of items present in the list.

```
class Adapter (val listItem:ArrayList<Locations>) : RecyclerView.Adapter<Adapter.RecycleViewHolder>(){
    inner class RecycleViewHolder(itemView: View):RecyclerView.ViewHolder(itemView){
        val itemImage: ShapeableImageView = itemView.findViewById(R.id.item_image)
        val heading: TextView = itemView.findViewById(R.id.item_title)
        val detail: TextView = itemView.findViewById(R.id.item_detail)
        val category: TextView = itemView.findViewById(R.id.item_categories)

    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): RecycleViewHolder {
        val view:View = LayoutInflater.from(parent.context).inflate(R.layout.list_item, parent,  attachToRoot: false)
        return RecycleViewHolder(view)
    }

    override fun getItemCount(): Int {
        return listItem.size
    }

    override fun onBindViewHolder(holder: RecycleViewHolder, position: Int) {
        val item = listItem[position]
        holder.itemImage.setImageResource(item.itemImage)
        holder.heading.text = item.headings
        holder.detail.text = item.detail
        holder.category.text = item.category

    }


}
```

Fig 53. Adapter class

Now, we need to code the HomeFragment to display all the items. In the HomeFragment, we need to get the recycler view by its id and create a vertical layout manager. After that, we need to create an array list of class Locations. The variable called “adapter” will pass the ArrayList to our adapter. Lastly, we will set the adapter with the recycle view. The

function called “test data” is to store the location details and this function will call out in the onCreateView() function.

```
class HomeFragment : Fragment() {

    lateinit var recycle1:RecyclerView
    private val list = ArrayList<Locations>()
    private val adapter:Adapter = Adapter(list)

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {

        var v =inflater.inflate(R.layout.fragment_home, container,  attachToRoot: false)

        recycle1 = v.findViewById(R.id.rv_item)
        recycle1.layoutManager = LinearLayoutManager(activity)

        list.clear()
        testData()

        val adapterr = Adapter(list)
        recycle1.adapter = adapterr
        adapter.notifyDataSetChanged()
        recycle1.setHasFixedSize(true)

        return v
    }

    private fun testData(){
        list.add(Locations(R.drawable.book_cafe, headings: "The Book Cafe", detail: "20 Martin Rd", category: "Cafe"))
        list.add(Locations(R.drawable.citysprouts, headings: "City Sprouts", detail: "102 Henderson Road", category: "Cafe"))
        list.add(Locations(R.drawable.esplanade, headings: "Library@esplande", detail: "8 Raffles Ave", category: "Library"))
        list.add(Locations(R.drawable.hf, headings: "Library@Harbourfront", detail: "1 HarbourFront Walk", category: "Library"))
        list.add(Locations(R.drawable.mangawork, headings: "MangaWork", detail: "291 Serangoon Rd", category: "Cafe"))
        list.add(Locations(R.drawable.orchard, headings: "Library@orchard", detail: "277 Orchard Road", category: "Library"))
        list.add(Locations(R.drawable.rabbitandfox, headings: "Rabbit&Fox", detail: "160 Orchard Rd", category: "Cafe"))
        list.add(Locations(R.drawable.sixlettercoffee, headings: "Tó Letter Coffee", detail: "259 Tanjong Katong Rd", category: "Cafe"))

    }
```

Fig 54. HomeFragment class

For the settings fragment, users can add new locations by clicking the add location button. First, we need to create a new activity called “LocationActivity". We have created a data class called “Location” to contain the fields for accessing them. In the LocationActivity, we created a function called “validateLocationDetails” to validate that all the fields are not empty. After that, we create another function called “addLocation " inside the Firestore class. In the addLocation function, the main function is to set the location details in the Firestore database. The locRegisterationSuccess function is called out inside the addLocation function. The main function of locRegisterationSuccess is when the location successfully adds to our app, it will display a message to users. The addLocation function and locRegistrationSuccess function can be seen in the diagram below:

```
fun addLocation(activity: LocationActivity,locationInfo: Location){
    mFireStore.collection( collectionPath: "locations")
        .document(locationInfo.locName)
        .set(locationInfo, SetOptions.merge())
        .addOnSuccessListener { it: Void!
            activity.locRegistrationSuccess()
        }
        .addOnFailureListener { e->
            activity.hideProgressDialog()
            Log.e(activity.javaClass.simpleName, msg: "Error while registering the location.",e)
        }
}
```

```
fun locRegistrationSuccess(){

    //Hide the progress dialog
    hideProgressDialog()

    Toast.makeText( context: this@LocationActivity,"Add location successfully. Our admin will verify the de...", Toast.LENGTH_SHORT).show()
}
```

Fig 55 & 56. addLocation (top) and locRegistrationSuccess (bottom) function

Besides, we created another function called “registerLocation” in LocationActivity.kt to run the registration. If the validate function is valid, it will store the run the addLocation function and store the location details in the Firestore database. After that, it will redirect the user to the dashboard activity.

```
private fun registerLocation() {
    if(validateLocationDetails()){
        showProgressDialog( text: "Please Wait ...")
        val location = Location(
            et_locName.text.toString().trim{ it <= ' ' },
            et_locAddress.text.toString().trim{ it <= ' ' },
            et_phoneNumber.text.toString().trim{ it <= ' ' },
            et_hours.text.toString().trim{ it <= ' ' },
            et_amenities.text.toString().trim{ it <= ' ' },
            et_transport.text.toString().trim{ it <= ' ' },
            et_desc.text.toString().trim{ it <= ' ' },
        )
        FirestoreClass().addLocation( activity: this@LocationActivity,location)
        val intent = Intent( packageContext: this@LocationActivity, DashboardActivity::class.java)
        startActivity(intent)
        showErrorSnackBar("Thanks for registering!", errorMessage: false)

    }
}
```

Fig 57. registerLocation function

Lastly, we need to call out the registerLocation function when the user clicks the create button. Users also can go back to the dashboard activity by clicking the icon on the left-hand side. The click listener for each button or text view can be seen in the diagram below:

```
ivPrevious.setOnClickListener{ it: View!
    val intent = Intent( packageContext: this@LocationActivity, DashboardActivity::class.java)
    startActivity(intent)
}

CreateBtn.setOnClickListener{ it: View!
    registerLocation()
}
```

Fig 58. Click listener for each button

In the settings fragment, users also can click the edit icon to edit their profile. In this case, we need to create a new activity called “UserProfileActivity”. In the UserProfileActivity, the username and email cannot be modified, but users can add the profile image, gender, and contact number. First, we created a function called “validateUserProfileDetails” to validate that all the fields are not empty. Second, we create a function called “updateUserProfileData” in FirestoreClass to update the user details. After that, we create a function called “updateUserProfileDetails” to run the updateUserProfileData function. The updateUserProfileDetails function can be seen in the diagram below:

```
private fun updateUserProfileDetails(){
    val userHashMap =HashMap<String,Any>()

    val mobileNumber = et_phoneProfile.text.toString().trim{it <= ' '}

    val gender = if(rb_male.isChecked){
        Constants.MALE
    }else{
        Constants.FEMALE
    }

    if(mUserProfileImageURL.isNotEmpty()){
        userHashMap[Constants.IMAGE] = mUserProfileImageURL
    }

    if(mobileNumber.isNotEmpty()){
        userHashMap[Constants.MOBILE]=mobileNumber
    }

    userHashMap[Constants.GENDER]= gender

    userHashMap[Constants.COMPLETE_PROFILE] = 1

    FirestoreClass().updateUserProfileData( activity: this,userHashMap)
}
```

Fig 59. updateUserProfileDetails function

Lastly, we add a click listener to the save button, so users can update their profile details in the firestore database. The click listener for the save button can be seen in the diagram below:

```
R.id.btn_save ->{

    if(validateUserProfileDetails()){

        showProgressDialog( text: "Please Wait ...")

        if(mSelectedImageFileUri!=null)
            FirestoreClass().uploadImageToCloudStorage( activity: this,mSelectedImageFileUri)
        else{
            updateUserProfileDetails()
        }
    }
}
```

Fig 60. Click listener for save button

For the search fragment, the layout is similar to the home layout, but we have added the filter button and search bar. The click listener for the save button can be seen in the diagram below:

```
v.btn_filter.setOnClickListener{  it: View!
    val intent = Intent(requireContext(), FilterActivity::class.java)
    startActivity(intent)
}
```

Fig 61. Click listener for save button

Flaws in the technical design and implementation

1) Settings fragment
    - When the users click the edit icon, the program will crash. We found that the problem is when calling UserProfileActivity from SettingsFragment, it cannot get the user details from the firestore database. To confirm the problem, we have tried to call the UserProfileActivity from the LoginAcitivity, it can add the profile image and update their details. Due to our lack of knowledge of the Kotlin language, the problem still cannot be solved.

2) Search fragment
    - The search bar cannot be displayed successfully. We have tried a lot of ways to let the Search bar be displayed, but all cannot work. Due to our lack of knowledge of the Kotlin language, the problem still cannot be solved.

3) Specific Location Activity
    - We cannot display the specific location layout when the user clicks the location details. Due to our lack of knowledge of the Kotlin language, the problem still cannot be solved.

4) Filter activity
    - There are not any functions when users click each button in the filter activity. We have tried a lot of solutions but it doesn’t work in our program. Due to our lack of knowledge of the Kotlin language, the problem still cannot be solved.

Testing During develop application

Register an account (positive)

Test Scenario ID	testRegister01				
Test Case Description	Test user success register account				
Prerequisite	-				
No	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**Test Result**
1	Create an account by enter email, username,password, and confirm password	***Username:*** “user001” ***email:*** testuser001@gmail.com ***password:*** test123 ***Confirm password :*** test123	Your account has been created.	Your account has been created.	Pass

Register an account (negative)

Test Scenario ID	testRegister02				
Test Case Description	Test user unsuccessful register account				
Prerequisite	-				
No	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**Test Result**
1	Create an account by enter exist email	***Username:*** "test002" ***email:*** testuser001@gmail.com ***password:*** test123 ***Confirm password :*** test123	The email address exists. Please use another email address.	The email address exists. Please use another email address.	Pass

2	Create an account by entering a username, password only.	***Username:*** "user003" ***Email:*** *NA* ***password:*** test123 ***password confirmation:*** test123	Please enter an email.	Please enter an email.	Pass
3	Create an account by entering an email, password only.	***Username:*** *NA* ***email:*** testuser003@gmail.com ***password:*** test123 ***Confirm password :*** test123	Please enter your username.	Please enter your username.	Pass

Login an account (positive)

Test Scenario ID	testLogin01				
Test Case Description	Test user successful login account				
Prerequisite	A valid account				
No	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**Test Result**
1	Enter valid email and password, and click the login button.	***Email:*** testuser001@gmail.com ***Password:*** test123	Login successful, redirect to the home fragment.	Login successful, redirect to the home fragment.	Pass

Login an account (negative)

Test Scenario ID		testLogin01			
Test Case Description		Test user unsuccessful login account			
Prerequisite		-			
S.No	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**Test Result**
1	Enter invalid email and click the login button.	***Email:*** testuser002@gmail.com ***Password:*** test123	The email or password is not correct. Please check again.	The email or password is not correct. Please check again.	Pass
2	Click the login button without entering email or password.	***Email:*** testuser002@gmail.com ***Password: -***	Please enter the password.	Please enter the password.	Pass

Update user profile (positive)

Test Scenario ID		testUpdate01			
Test Case Description		Test user successful update profile			
Prerequisite		Login to your own account.			
No	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**Test Result**
1	Go to settings fragment and click the edit icon.	***Contact Number:*** *01121832781* ***Gender:****Male*	Your account has been updated.	Your account has been updated.	Fail
2	Go to UserProfileActivity upload image, contact number, and gender.	***Image:*** ***"****pic01.png"* ***Contact Number:*** *0116688688* ***Gender:****Male*	Your account has been updated.	Your account has been updated.	Pass

Update user profile (positive)

Test Scenario ID		testUpdate02			
Test Case Description		Test user unsuccessful update profile			
Prerequisite		Login to your own account.			
No	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**Test Result**
1	Update profile by entering gender only.	***Contact Number:*** *-* ***Gender:****Male*	Please enter the contact number.	Please enter the contact number.	Pass
2	Update profile by insert image and gender only.	***Image:*** ***"****pic01.png"* ***Contact Number:*** *-* ***Gender:****Male*	Please enter the contact number.	Please enter the contact number.	Pass

Analysis

SWOT analysis

Strengths

- Users can search for study locations without needing to spend time looking for them.
- It can help make use of space that is left empty during off-peak hours.
- Allow for spaces to be used effectively as users can share not just information about the space but also their preferences (E.g User A does not mind sharing seats, so User B and C can share a table).
- Users can choose the venue according to the reading environment they want. For example, the venue can be a coffee shop, library, tutoring centre, etc.
- Users can filter up with some requirements. For example, Users can choose a venue equipped with a projector or large monitor, tables, etc.
- Users can reserve venues in advance.
- A google form to enable users to comment on the study location. So that users can review the feedback and do their selections.
- Users can meet more friends through the app. For example, Users can initiate activities to read together in the same place.
- Good community. Users can work together to share new locations with each other.

Weaknesses

- The venue in the app is different from the actual venue. For example, incomplete equipment, noise, and pollution.
- There is no choice in some remote places. For example, users in remote areas must go to the urban area, and they will waste time and energy.
- User comments are incorrect. For example, when the user arrives at that place, they find that the place doesn't have the equipment they want.
- Some owners of cafes/stores being used as a place to study may find that people studying at their outlet obstructs business.

Opportunities

- Offering flexible learning opportunities to students.
- Business cooperation can happen if there is a positive impact on revenue, and a chance to offer real rewards to users.
- Offers a local option for finding study places, which is not something widely available/known for Singapore.
- Many students find it difficult to find study locations and often have to travel a lot/wake up early to find a spot. Very attractive option.

Threats

- Destroyed by some competitors. For example, competitors will leave some incorrect information to mislead users.
- Not many places to choose from.
- Relies on users to give accurate data. However, as academics can be considered competition with other users, it may be difficult for them to always have accurate data due to inherent bias.
- Many business owners may find fault with the app if they find that it obstructs their business.

PESTLE Analysis

A PESTLE study is a methodology for analyzing the primary elements influencing an organization from the outside (Political, Economic, Sociological, Technological, Legal, and Environmental). It provides information about the external issues that affect the application.

Political factors:

In this group, we include the political factors in the country and how the government policies may affect the app's perspective. The project is based in Singapore. The country is governed by a stable administration that allows for free commerce, with its political risk being relatively minimal. According to the Political and Economic Risk Consultancy (PERC), the country has the lowest political risk on the continent. There is no government monopoly on study spot locators. Additionally, there are also no hurdles for app owners that may prevent the app from being launched in the future as no applications identical to this have been produced so far.

Economy:

Singapore has a thriving free-market economy and it is evolving at a rapid rate. The country has the greatest per capita income in ASEAN. Our app is primarily aimed at students who may choose to study outside for a variety of reasons. This software is adaptable to individual users' needs, providing practical filters for study places. Thus, it would aid in reducing time spent to locate study places. Furthermore, this software can also help to increase publicity for lesser-known study places, resulting in increased revenue and recognition for the respective businesses.

Sociological:

Singapore's literacy rate is quite high as primary schooling is required. According to an essay released by the University of Arizona, there are advantages to studying outside. Namely, how it reduces tension and improves mood, allowing users to concentrate on their job. In this day and age of rigorous learning, studying outside may help students concentrate better, aiding in improving health and increasing natural vitality. This is where our app comes in handy, to make users' lives easier by assisting them in finding appropriate study locations.

Technological:

Singapore is a technologically advanced country that is progressive and will not inhibit the implementation of a new technological solution. This program may assist in locating places based on how the user prefers to travel, and it also filters out results based on the user's needs. Future software updates could include the capacity to form new friend groups based on what users are studying.

Legal:

We check if our activities are legitimate in Singapore. So far, this app has not violated any rules. The privacy policy page informs consumers about the details of privacy and data processing activities. We've also discussed how the information gathered would be used. This program does not require any sensitive information from the user, complies with cookie regulation, with the Terms and conditions included in the app.

Environment:

This software has no negative environmental consequences. The application includes only public transportation travel routes and makes no mention of the use of private transportation. As the use of private transportation would contribute to the rise of carbon emissions, we strongly recommend users to commute via public transportation to reduce environmental pollution.

Ways to measure project success

1. Schedule

The effectiveness of project management is frequently evaluated by whether or not you adhere to the original schedule. The scheduling helps us get clear on our project purpose, which is vital in making our project successful. In this case, we will update our project schedule every week. For example, we use Gantt charts to update our project schedule so that the important tasks and deadlines can be viewed visually. This would help us with our productivity, allowing us to easily delete or delegate tasks. Which parts are most important? Which part must be done first? Is our project delivered on time? The lists will help us to answer these questions.

2. Quality

A quality review at the end of a project phase is always a good option. We can check whether the quality of our project meets our expectations. If not, we can adjust our plan in time to achieve the desired result we want. It's best to find out immediately before the project goes too far, because it may be too late to do anything by then.

In this case, we have adjusted the registration form to increase security. For example, we use the confirm password method to ensure our customer didn't mistype their password.

3. User Satisfaction

In this case, we hope to get user feedback. This is because we can learn what users like and dislike through feedback. So that, we can improve our project and services to keep them in top condition. Ultimately, this will lead to better business, better user experience, etc. After all, satisfied customers are likely to use our app again. This would help increase customer loyalty, and serve as our biggest advocates.

Evaluation

Examining our Software Development Journey

In order for us to evaluate our software development journey, we need to carefully review every step taken and our endpoint. Looking back on the process of development, we can simplify it into three key steps: Brainstorming, design, and coding.

The first step is brainstorming, which covers the development of the idea, the analysis, and the improvement of the idea. While ideas are a product of creativity, there are also ways to streamline the brainstorming process. For example, one of the methods we could have used was brainwriting. Each of the group members writes down three ideas each within a set time period, for example, 5 minutes. Then, everyone swaps their set of ideas with someone else's, and then they write down three more ideas. These can either be new concepts or be built on the set they took. After the ideas have been passed around the entire group, the sets are collected and the new ideas are reviewed by the group. This would have improved the brainstorming and generated multiple ideas for our group to use as our foundation.

The second step is design, which is the process of converting rough designs into actual wireframes, wireframes into prototypes, and conducting usability testing on the prototypes. Our design process was efficient, however, we would have much to gain from additional user testing. Having a constant stream of feedback from users as we develop the product would help us to cut out unnecessary features earlier and work on functions that are truly desirable to our target audience. This can be seen from the earlier rounds of testing, where our product's design changed due to criticism from users. For example, the removal of the about us button, and the changes to the navigation bar.

The final step is coding, which involves learning Android Studio and Kotlin, implementing our idea with them, and creating a prototype. The time needed for the coding step to be completed was too long. It may have been wiser for us to either work with something familiar like a website or a simpler app design. A table showing the comparison of advantages between the coding of a mobile app and coding a mobile website for our project has been provided below.

Table of comparison

Mobile Website	Mobile App
Wider audience	Improved user experience
Easier to implement	Better integration
Familiarity	Offline access

Wider audience - A website can be accessed from multiple devices, e.g. iPhone, Android, laptop.

Improved user experience - Users can set preferences in the app, creating a more personalised experience.

Easier to implement - A website would make it easier for our team to code the entire project with all the functions we want.

Better integration - An app is able to use the device's inbuilt hardware, e.g. GPS, camera, Bluetooth.

Familiarity - Our team is familiar with websites, so we would spend less time learning new things and we would be able to focus on mostly coding.

Offline access - Apps are capable of storing data offline, enabling users to view data without Internet access.

Both websites and apps provide different benefits to our project. With no clear disadvantage, it becomes necessary for us to choose what best suits our project. After considering these factors and working through this project, our team still believes that an app is a better fit. This is because the integration offered by an app would allow us to access hardware features and provide a better user experience in general.

Market Research

Case study 1: Waze app

Waze app is a mobile app, on both iOS and Android, that enables users to plan their travel routes and change them while on the way, according to real-time updates on traffic situations. It takes crowdsourced traffic information and helps users to adapt their travel route to other factors, like traffic, police cameras, or speed limits. Appendix 2 contains screenshots of the app.

Functions:

- Uses icons to flag things like police, traffic cameras
- Users able to save places under “Favorites”
- Trips can be scheduled under “Planned drives”
- Crowdsourced information on roads
- Real-time updates on the map
- Custom maps
- Also flags location of users
- Zoom feature
- Colour coding for crowd level
- Collaborates with stakeholders to improve data

What can we learn to use in Spaceus?

- Use icons to flag things like F&B, toilets, vending machines
- Custom maps for study locations with live updates from community information
- Location flagging for users, study groups, can help to organize a crowd
- Colour code for crowd level

Disadvantages of the above:

- Overuse of icons can result in clutter.
- Custom maps are unreliable without up-to-date data due to recent changes. (Tables shifted for the event, restricted seating due to covid, unorthodox furniture making it difficult to accurately “calculate” space.)
- Not all users are willing to share data like location or even study locations.
- Colour codes may not be as useful due to study areas being different from roads.

Case study 2: Fern app

Fern App is a web-based app that relies on crowdsourced information for study locations.
The app has a search bar to search for locations and a filter results button to filter for results based on user selection. The filters are based on 4 options: Wifi, Noise, Seating, and Power Outlets. There is also an option, Search by Map located near the filter results, that enables users to find a location on the map. Users can create a user profile and add a list of places in our profile for easier navigation. Appendix 3 contains a screenshot of the home page.

Functions:

- Search function with filters
- Able to search for nearby study locations around the user
- Add new locations for studying
- Users can create profiles so that their preferences and favourite locations are saved

Pros:

- Uses symbols to represent filters, which makes it intuitive to understand.
- A simple interface is easy to navigate
- Compact design maximizes the map view

Cons:

- Not intuitive to use as not all functions are clear due to compact design
- Not capable of resizing itself to fit a wide range of screens
- Limited functionalities
- Does not load quickly

Evaluating against case studies

Both case studies are examples of crowdsourced data leading to an output in the app. Similarly, we hoped our idea would match up to these case studies. However, upon a careful review of our product, there are some points that we can learn from.

Firstly, not all of our features work. While it might seem like a non-essential and almost obvious point to bring up, there are a few things we need to address. Bugs have always led to poor user experience and thus it is necessary that we keep our product in a functional state. Since we have some features that do not work, we can consider this as a major fault that will lead to a bad user experience. This means that in comparison to a working product, our project may seem inadequate. For us to have a fair evaluation of the working components, most other comparison points will have to take the current state of our project into consideration as well.

Secondly, we made the app intuitive for local users. Users are able to grasp what needs to be done without any instructions. This is enhanced by the minimalistic design of the app, which isolates key functions into easily accessible areas, pushing the main features to users and separating additional ones into a different area. This is comparable to Waze and possibly better than Fern, where users can navigate the different functions without going through multiple pages. A large majority of our functions are available on the navigation bar and this shortens the task flow.

Thirdly, our project only has a few functions. This is good because it helps to keep the focus on our key purpose of providing study spaces and it also reduces the clutter on the app. This is in line with our minimalistic design, maximising the features while keeping the interface clear of extra functions. However, additional features may need to be added in the future, since users may find that their use of the app is impeded by the lack of functions. For example, in our first usability testing, some feedback from users mentioned that they would like custom seating maps of the places in the app. This will help them better visualise crowding and their study space management, allowing them to make wiser choices when it comes to where and when to study outside.

Lastly, our current project is unable to harness the phone’s hardware. The main advantage provided by an app is the ability to utilise the phone’s hardware to perform additional functions, for example, using the phone’s GPS to find the user’s current location and to give directions to the user. However, our team was unable to implement this due to the technical difficulty we faced when we were learning mobile programming. This can be resolved in future updates to the code, where we intend to create a custom list of locations catering to the user’s current location.

Conclusion/Summary

The final version of Spaceus has been tested and evaluated, and throughout this journey of creating an application, we have had some successes and some failures. Through our weekly meetings to plan and strategize our steps to develop Spaceus, we have been able to collaborate over different circumstances and it has also allowed us to learn and gain new soft and technical skills along the way.

Technical successes and failures

Throughout the development of our application, our team has met with numerous problems along the way. Some of these problems turned to successes as we were able to overcome and solve them while there are a few issues we came across that ended up as failures as we were not able to solve them.

Success

The biggest success we managed to achieve is being able to create this application from scratch. As mentioned before and throughout our proposal, our team did not have any prior knowledge of the Kotlin language nor did we have any experience in developing applications or using Android Studio before. This lack of knowledge and experience made it challenging for us when we first began our development of the application.

Everyone in the team had to learn some of the basics of Kotlin quickly to prevent any major delays in the development of our application. Having done some exploration with Kotlin at the start, we were then able to start coding.

With the current functionalities that we have been able to implement, Spaceus is able to allow users to find places to study and work at and add new locations into the database to expand the locations we have.

Failure

Some failures we faced were not being able to implement all the functions we originally planned to include in the application and not being able to recreate the app the way we designed it initially in Figma. Many efforts were made to try to replicate the design in Figma and include as many functions as we could, but we were still not able to build the application as well as we would like.

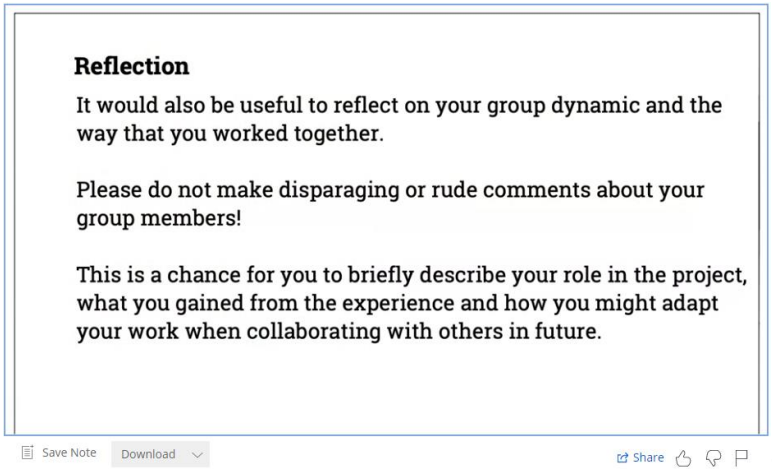
Impact

With Spaceus, we hope that users will find it useful in providing them with an easier way to find locations to study/work at in Singapore. We also hope that users will be able to rely on Spaceus whenever they need to find a suitable study spot to do their work.

Future plans

With the experience gained from building Spaceus, we hope to establish and build Spaceus properly and deploy it so that the application can be properly used. Further improvements and development would need to be done and added to the functions that have not been implemented. Through more research and practice, the functions we initially planned to include in Spaceus can be properly integrated and implemented.

Reflection

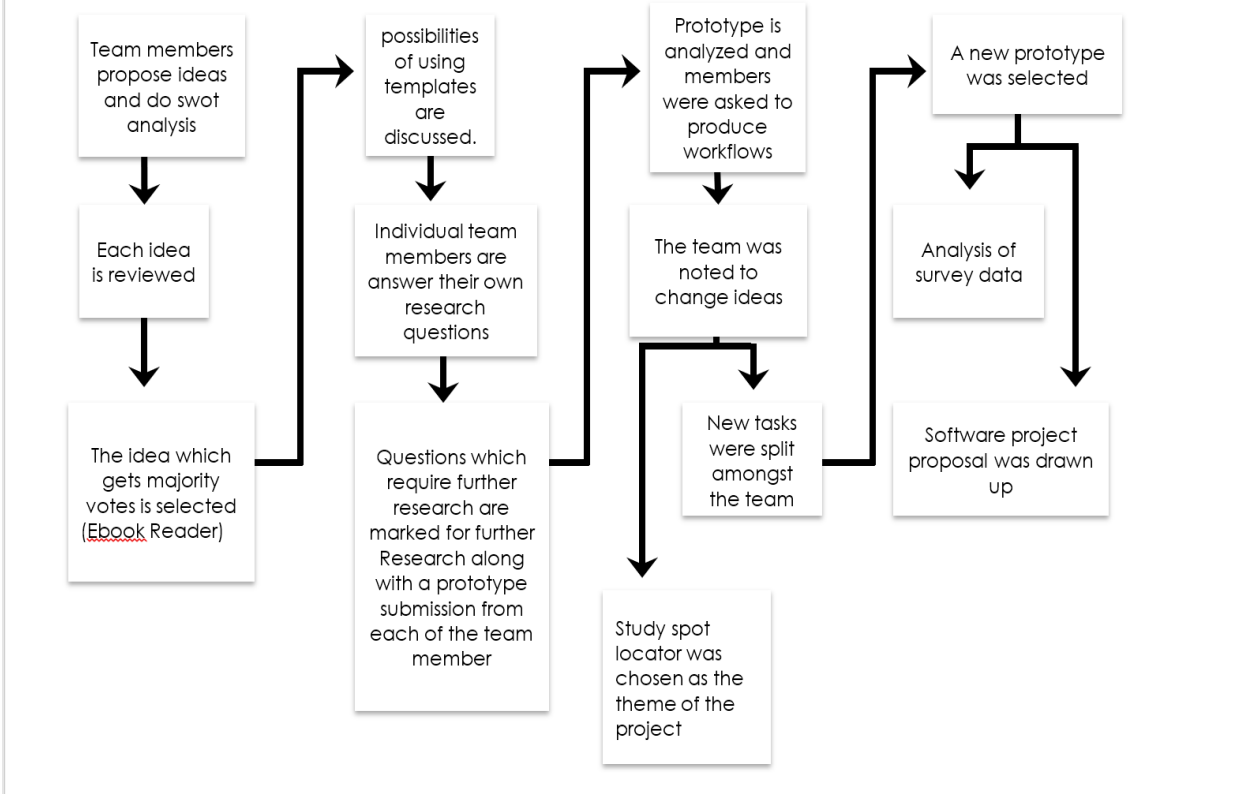


-remove this when writing your own reflections.

Appendices

Appendix 1

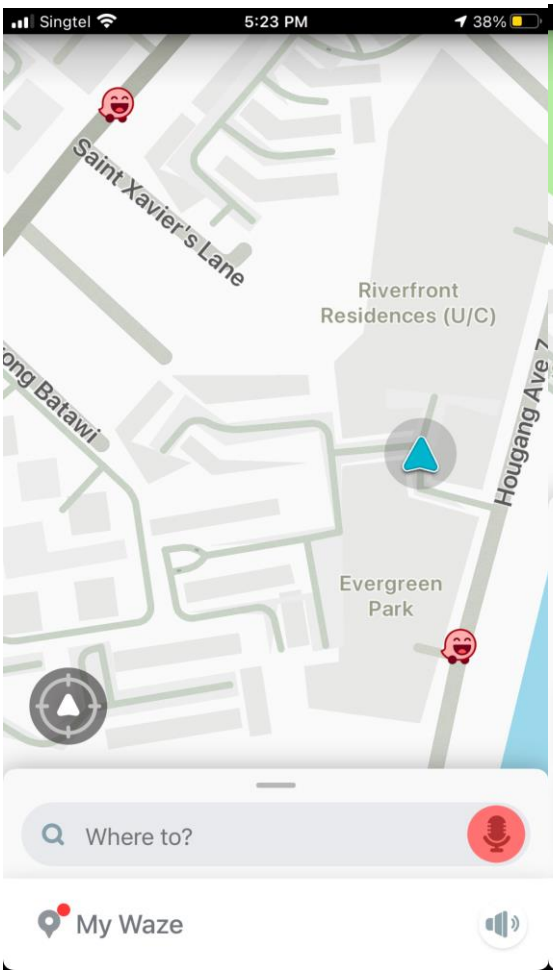
Workflow diagram



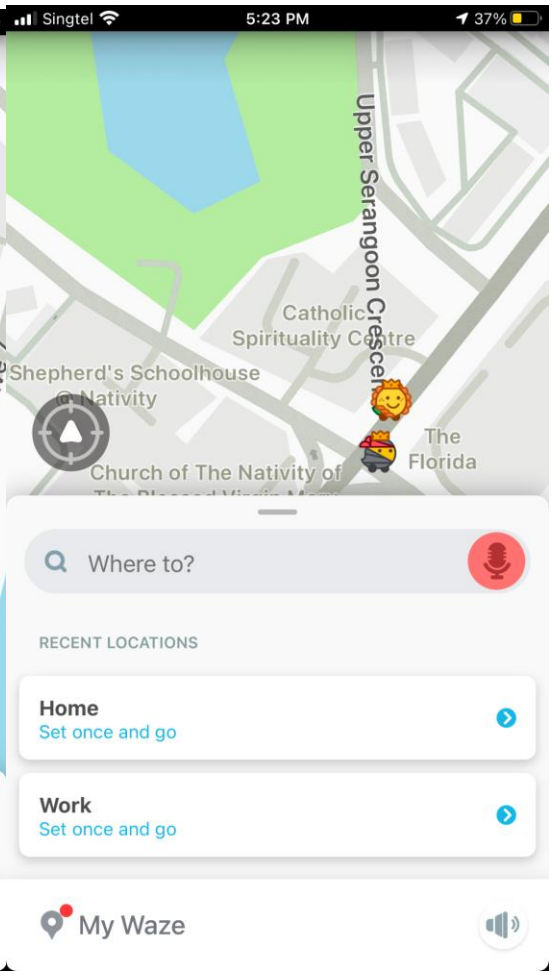
Appendices

Appendix 2

Waze app Home Screen



Waze app Search function

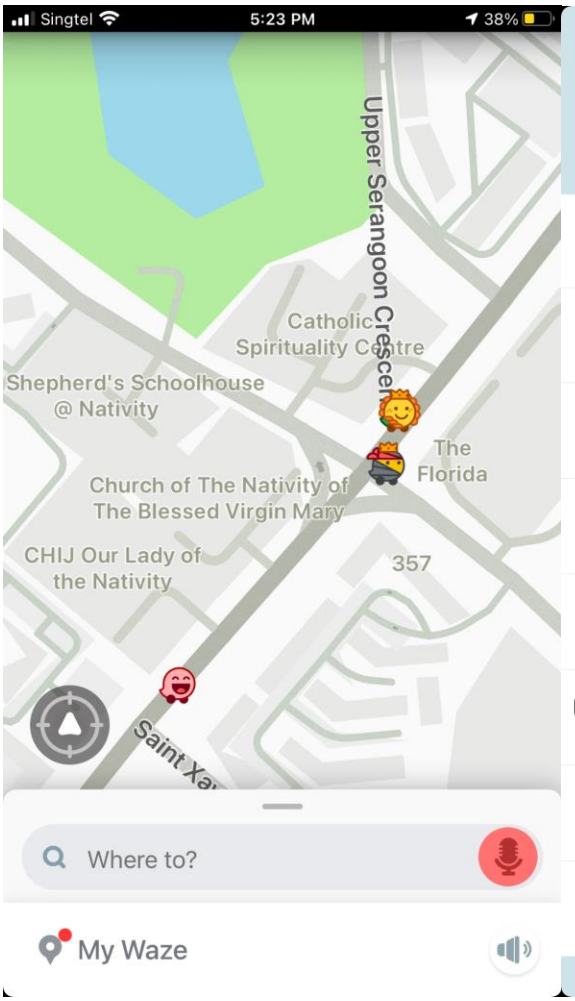


Source: Waze App

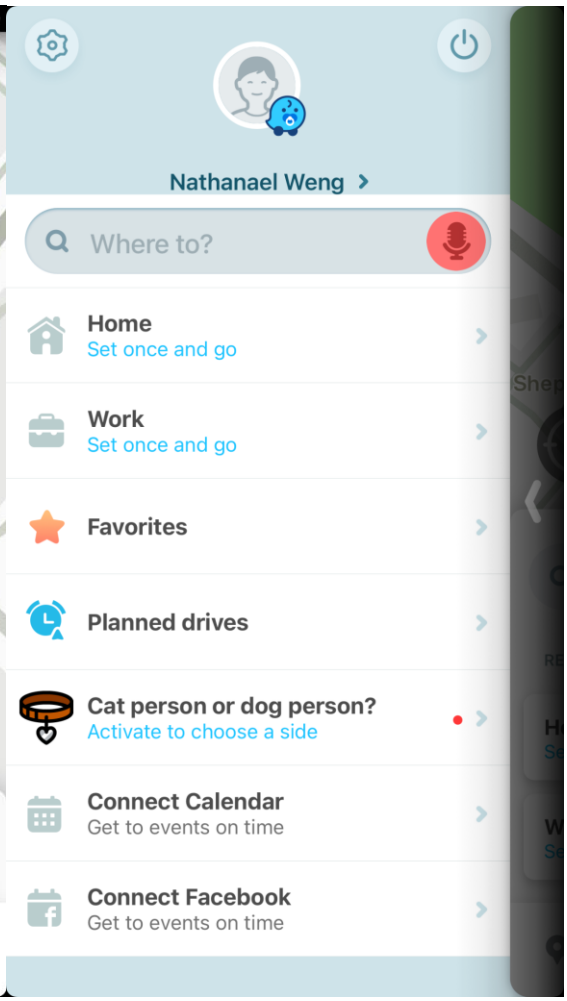
Appendices

Appendix 2

Waze app User Icons



Waze app Menu Tab

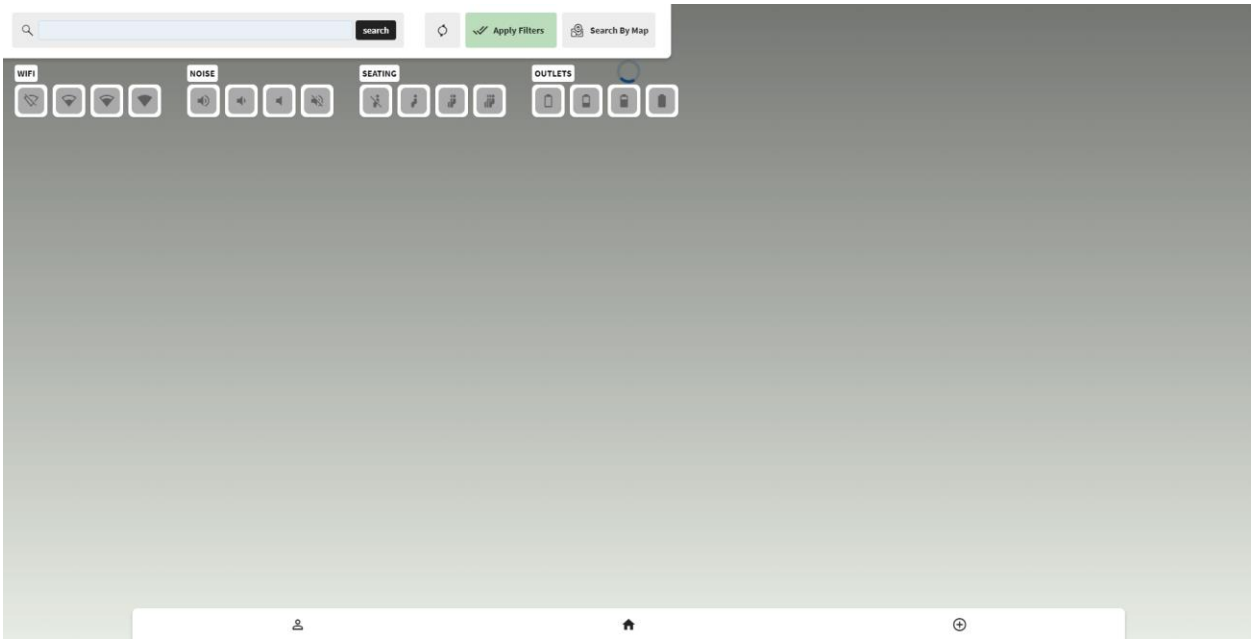


Source: Waze app

Appendices

Appendix 3

Fern app home screen



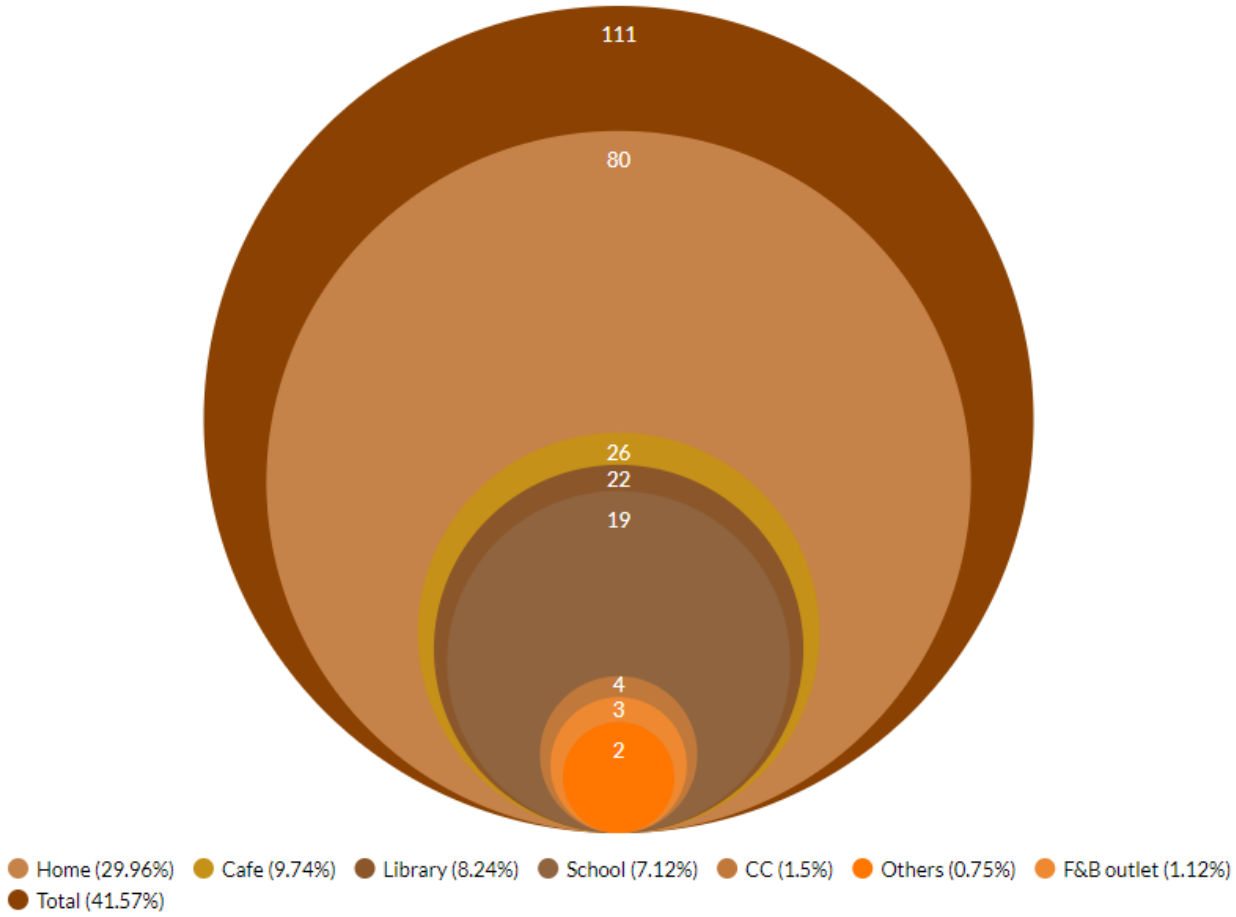
Source: https://app.fernapp.co/SearchLocations/Home

Appendices

Appendix 4

Reorganized survey data

Fig 1: Users listing their usual studying places



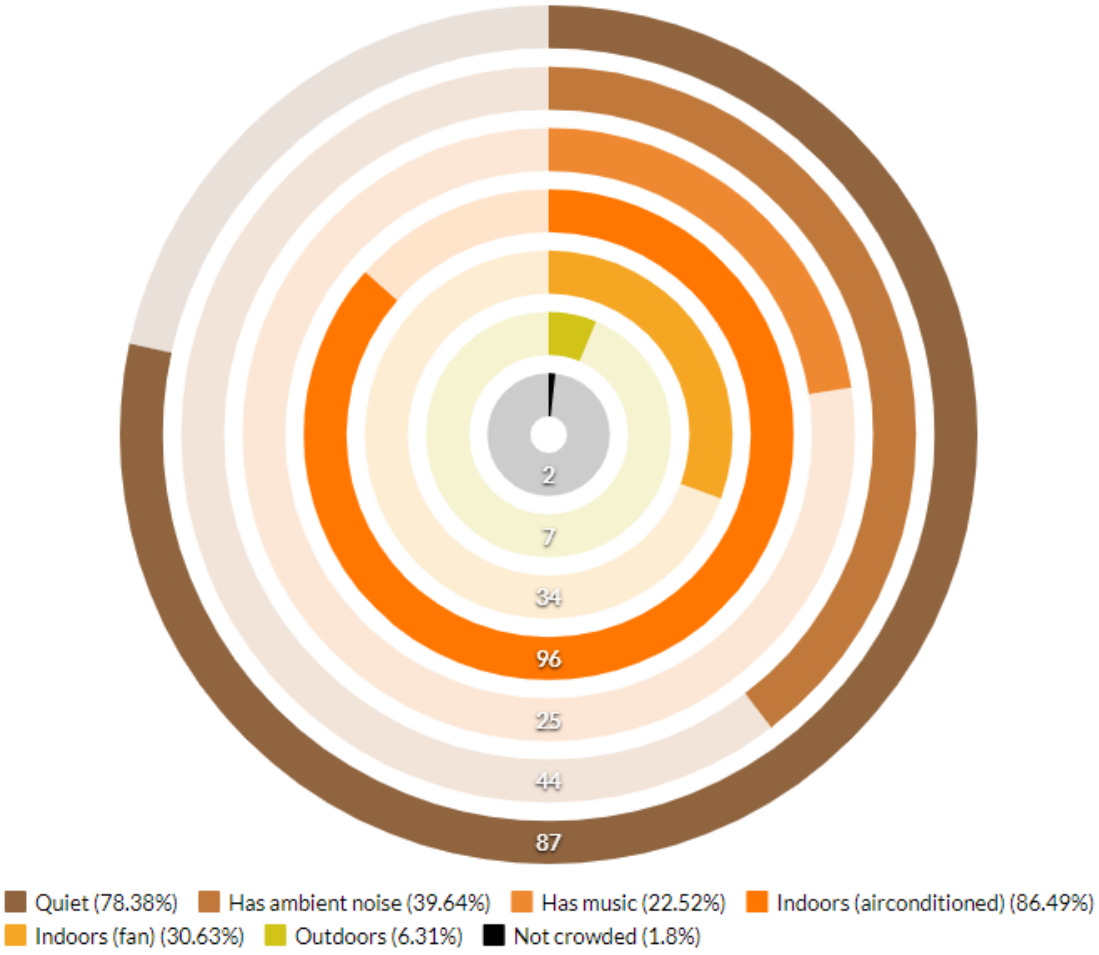
Source: https://docs.google.com/spreadsheets/d/1pwGTJcxzteMUlVlQUg716n-7Bd4Xh5lk3LemQADHRAg/edit?usp=sharing

Appendices

Appendix 4

Reorganized survey data

Fig 2: Users listing factors that make a conducive study environment



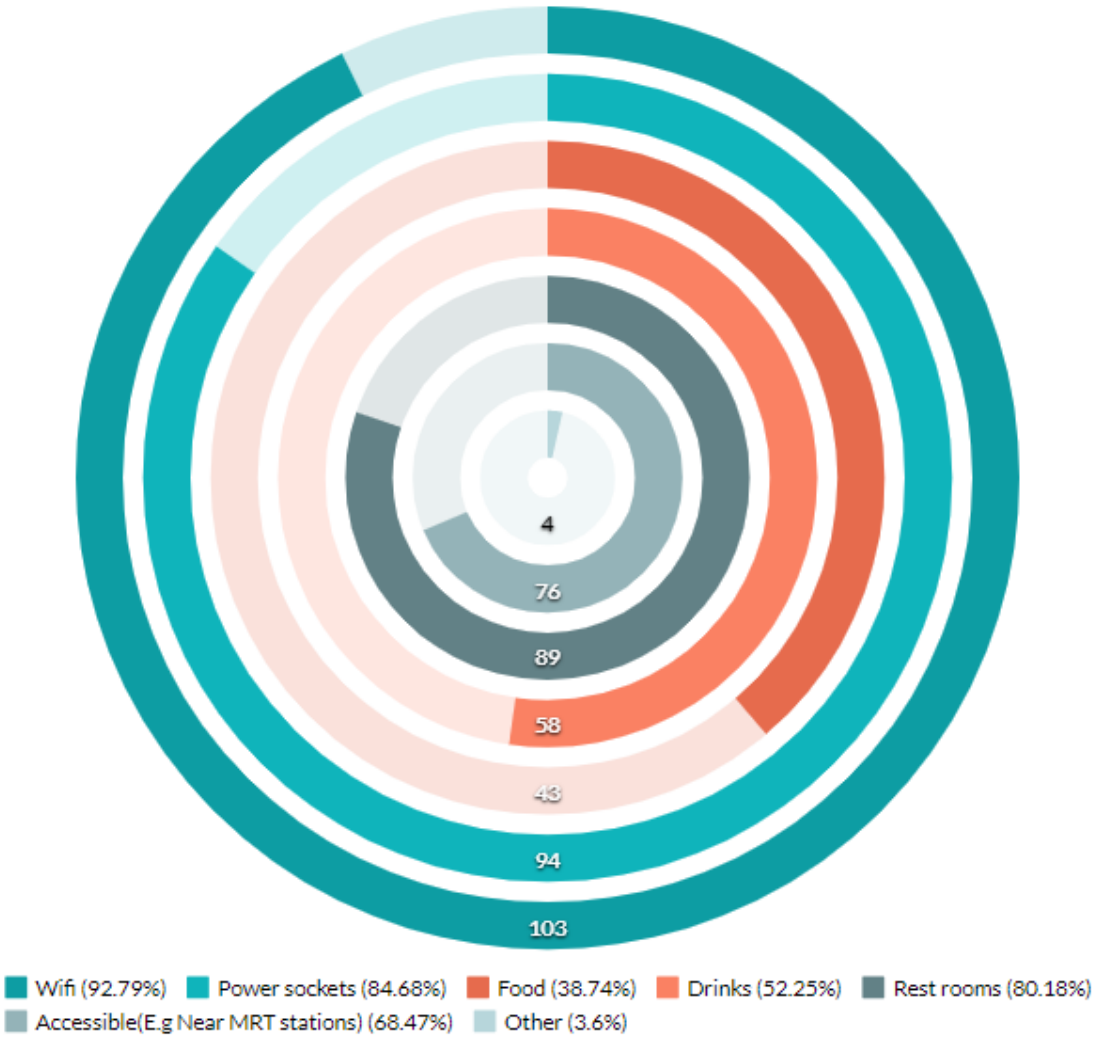
Source: https://docs.google.com/spreadsheets/d/1pwGTJcxzteMUlVlQUg716n-7Bd4Xh5lk3LemQADHRAg/edit?usp=sharing

Appendices

Appendix 4

Reorganized survey data

Fig 3. Essential features of a study location

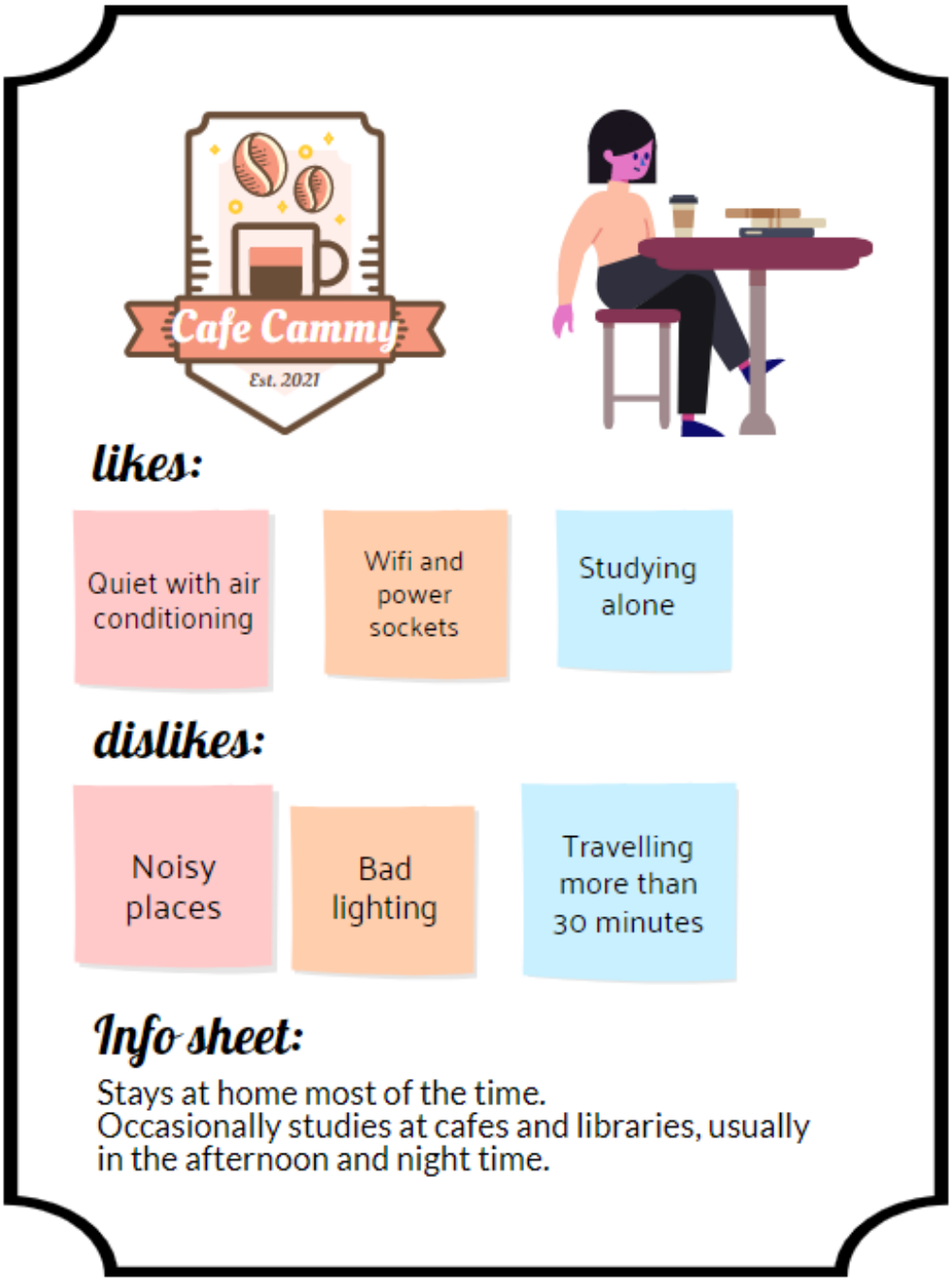


Source: https://docs.google.com/spreadsheets/d/1pwGTJcxzteMUlVlQUg716n-7Bd4Xh5lk3LemQADHRAg/edit?usp=sharing

Appendices

Appendix 5

User persona: Cafe Cammy



Appendices

Appendix 6

Links to task flow diagrams in Miro:

Login and Register Account: https://miro.com/app/board/o9J_lx0iwNg=/
Forgot Password: https://miro.com/app/board/o9J_lw7G3gI=/
Create New Location: https://miro.com/app/board/o9J_lw89L90=/
Update Profile: https://miro.com/app/board/o9J_lwDN3cc=/

References

Iubenda.com. *Laws and regulations every app developer should know - and how to comply.* [online]. Available at: <https://www.iubenda.com/en/help/14787-laws-regulations-every-app-developer-should-know>

Planningtank.com. 2020. *What is STEEPLE analysis?* [online]. Available at: <https://planningtank.com/market-research/steeple-analysis>

Pestle Analysis.com. 2015. *PESTLE Analysis of Singapore.* [online]. Available at: <https://pestleanalysis.com/pestle-analysis-of-singapore/#Environmental_Factors>

Thomas, N. (2019). *How To Use The System Usability Scale (SUS) To Evaluate The Usability Of Your Website - Usability Geek*. [online] Usability Geek. Available at: <https://usabilitygeek.com/how-to-use-the-system-usability-scale-sus-to-evaluate-the-usability-of-your-website/>

Uagc.edu. 2019. *Studying Outside vs. Inside | The Pros & Cons.* [online]. Available at: <https://www.uagc.edu/blog/studying-outside-vs-inside-the-pros-cons>