Official Documentation of

Connect like never before.

Click here to download our APK now

Project Overview

Motivation

With the rise of the digital age, e-meeting people have become the new norm. These interactions are created through platforms like web-based games and online forums. What if we crafted an application targeted at facilitating such interactions for like-minded individuals? This platform can cater to enthusiasts in cooking, sport, general wellness, or just about anything! Such an application will augment the 'search process' for interest buddies, finding a savvy way to link people who are looking to ping ideas with each other.

Currently, most popular applications centred around linking like-minded people are dating-related. While there are existing 'friendship apps' in the global market, none of them are prevalent in our local context. We are motivated to develop an app enabling people to strike up productive conversations, allowing them to personalise and maximise their utility of our platform.

Project Aim

We aim to create a social app called P!ng, targeting Singaporeans from all age groups. We hope to connect like minded individuals based on their passion and interests, while offering the flexibility of scalability.

User Stories

1. As an individual who possesses multiple interests, I will be able to craft a profile representative of my interests and background.
2. As an inquisitive individual scouring for new interests, I can easily identify broad channels to pique my interest in a topic.
3. As a well-informed individual looking for a fruitful discussion on a certain topic/specific expertise, I can identify small focus groups/partners to ping ideas.

   (The scalability of the app provides users with the flexibility to achieve both Points 2 and 3 simultaneously)

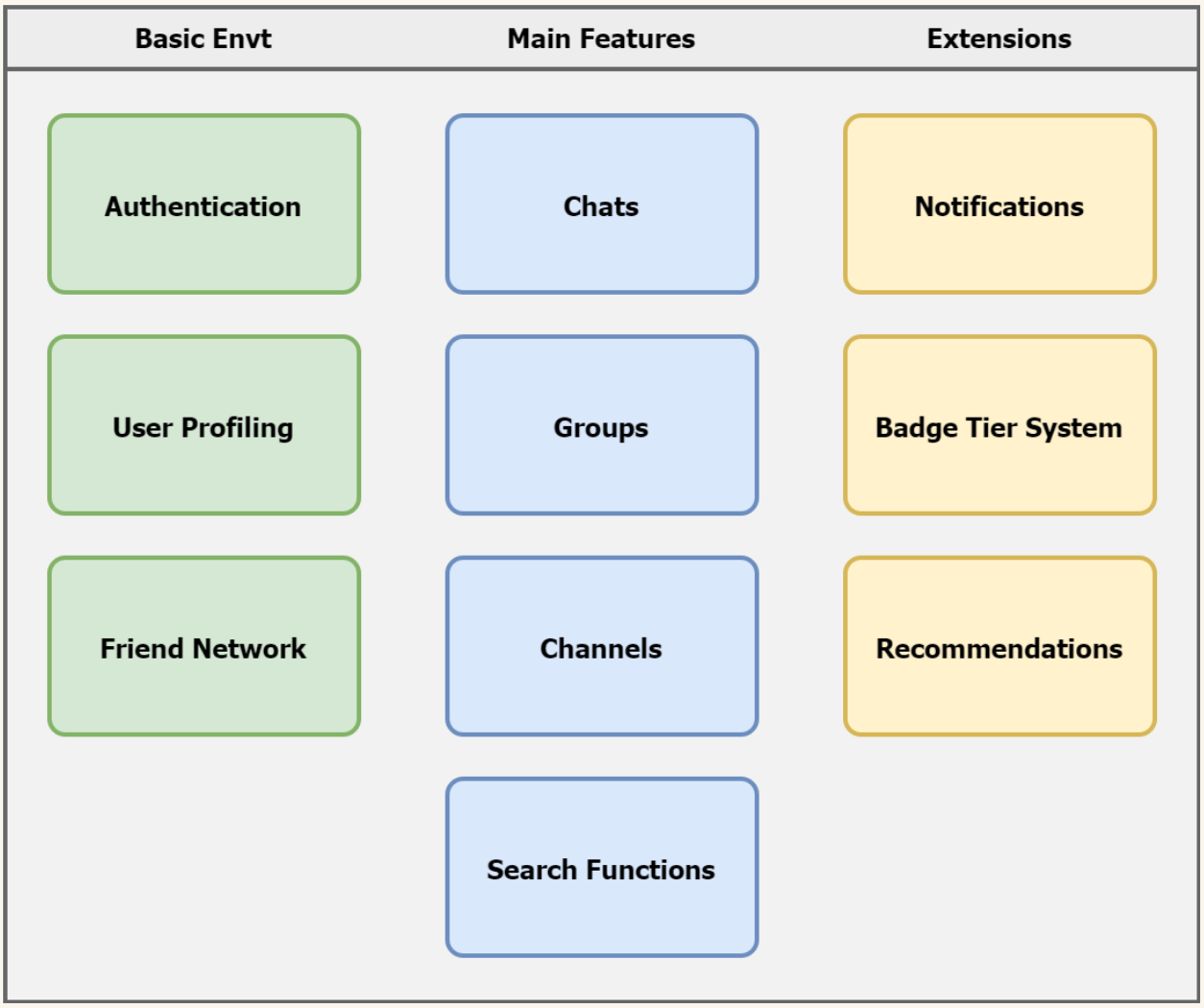
4. As a subject matter expert, I can gain credibility through tiered badges, using my knowledge in specific fields to manage and contribute publicly in channels.
5. As an entrepreneur/business, I can utilise P!ng to conduct market research and gain insight on the latest trends.

Tech Stack

- **React Native + Redux:** For mobile development and front-end purposes
- **Google Firebase (Authentication, Firestore, Cloud Functions, Cloud Storage):** For back-end purposes, authentication and storage of data
- **Git + Github:** For repository and source-code control

Application Features

P!ng offers the following features which have been grouped into 3 main categories:

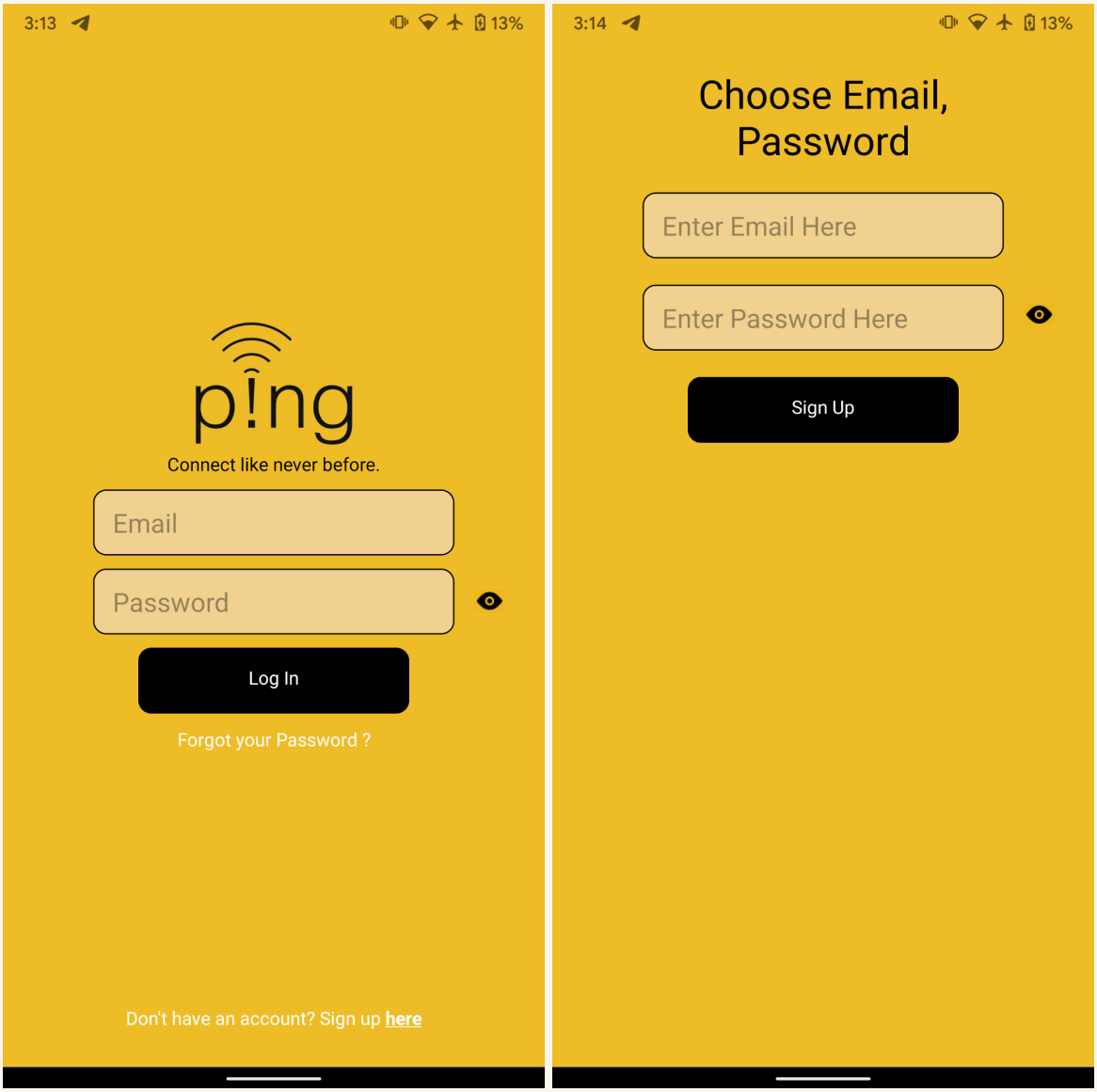


Basic Environment

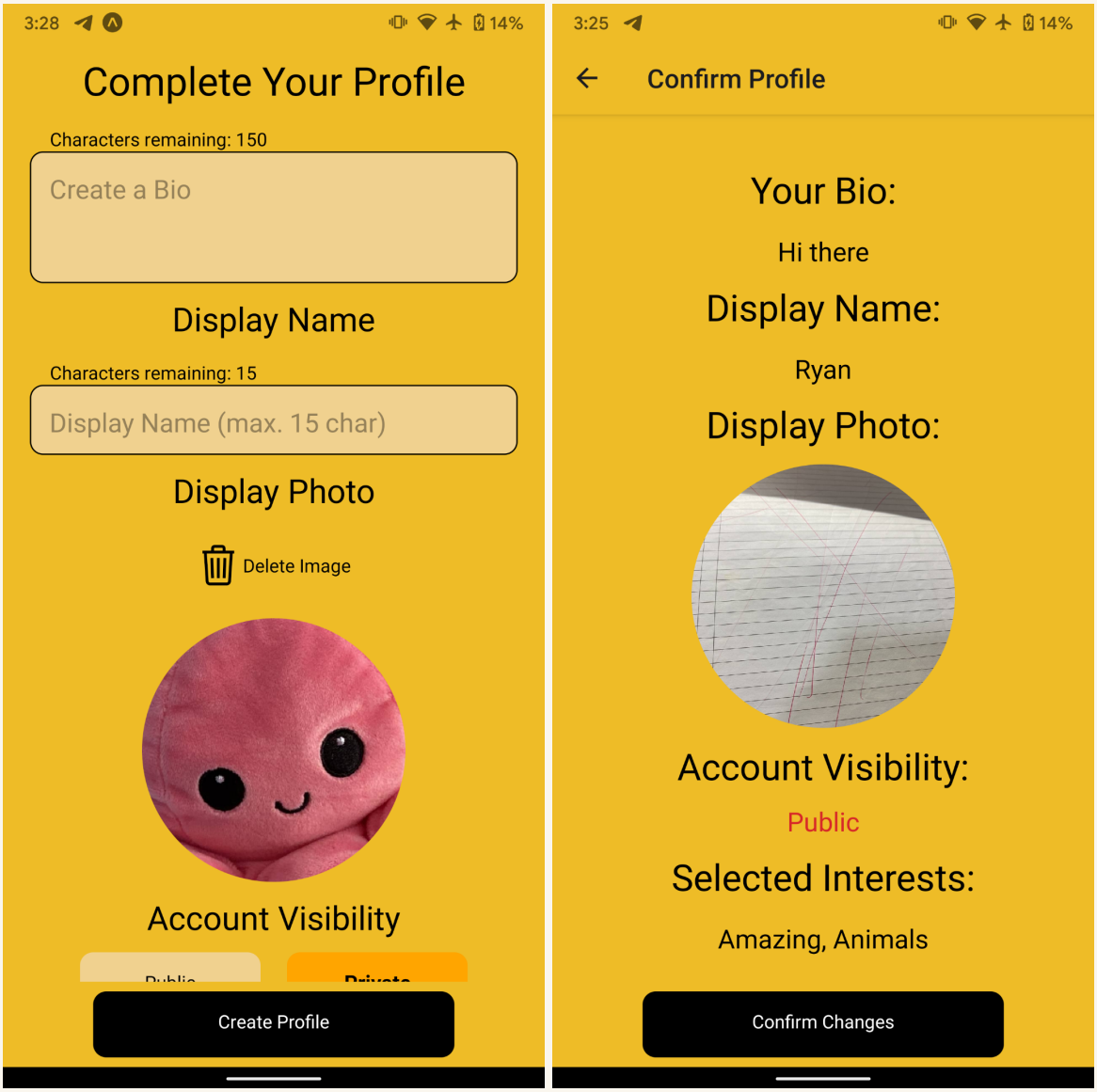
Authentication / Registering

The Login screen appears when P!ng is opened and allows for new users to sign up. It also allows for password recovery if an existing user has forgotten his/her password.

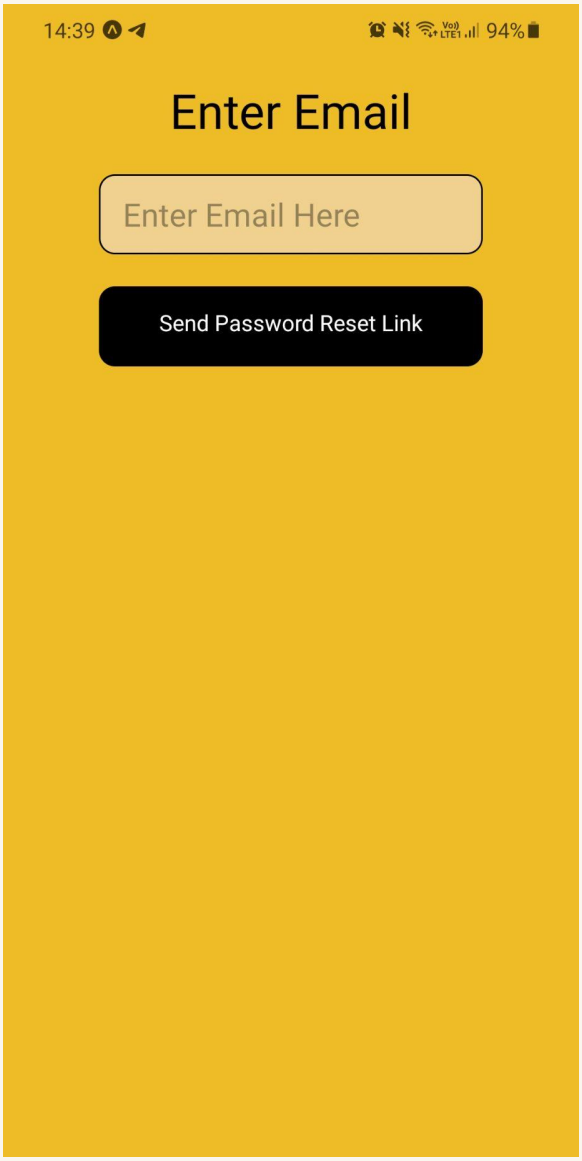
Upon registration, a user has to verify his/her account via email verification before reentering the app later on.



Each user has to complete their profile before using P!ng. He/she has to fill in a biography (max. 50 characters), a display username, and interest topics. There is also an option for account visibility which determines if the user's profile information can be accessed from a global search.



On the Forgot Password screen, the user will have to key in a valid email linked to an existing account, where they will receive a link to reset a password.

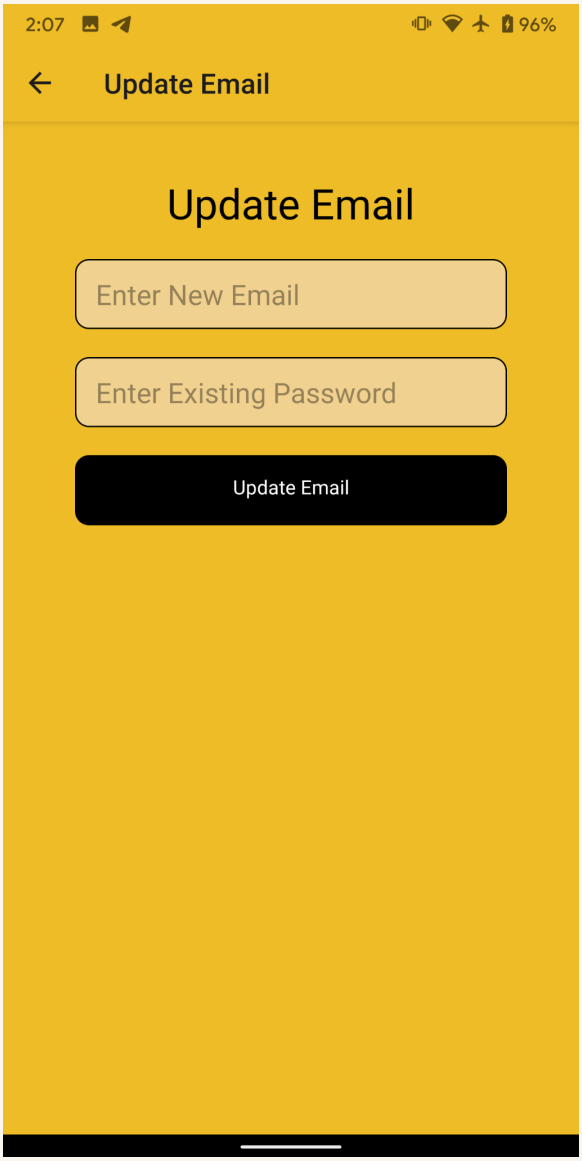


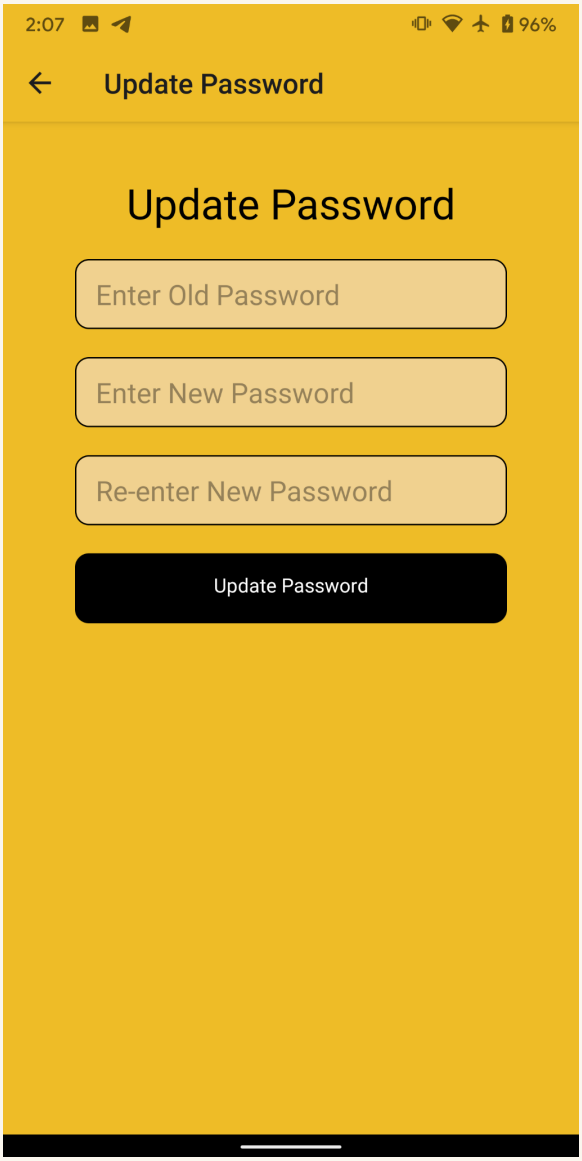
Settings

The settings menu provides personalisation options and important information for the user.

It appears as a navigation drawer.

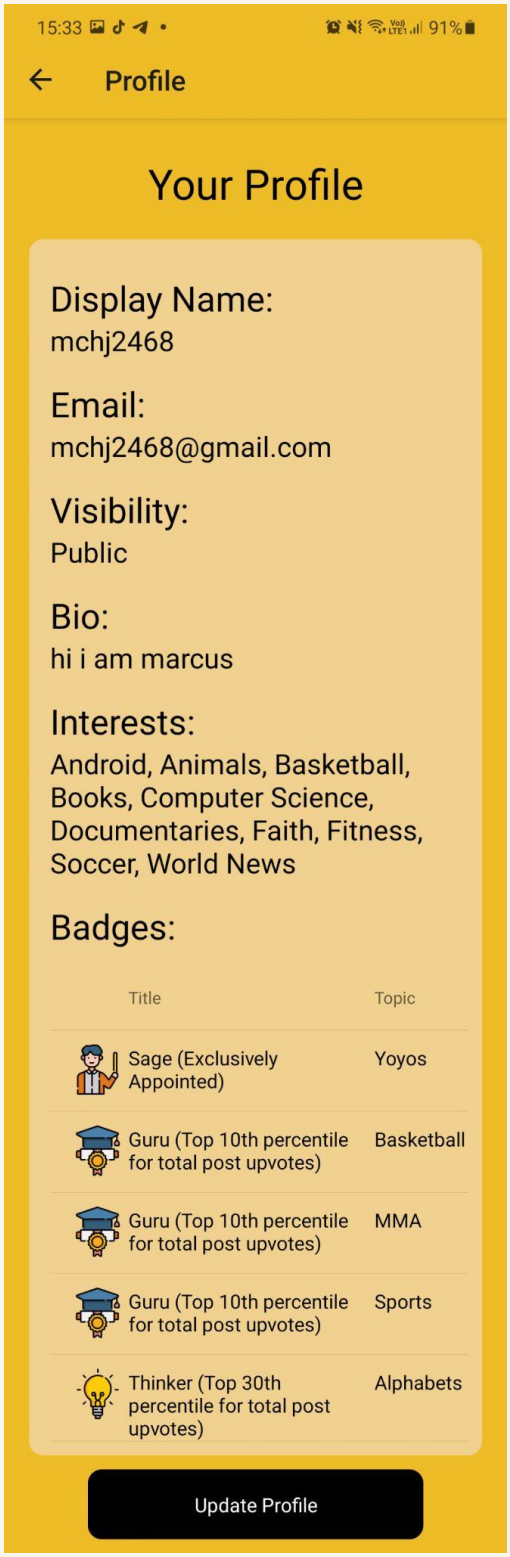
The user can change their email or password in the Security section. New emails must be valid and new passwords must be minimally 6 characters long. A confirmation email will be sent out to users after the details have been updated.

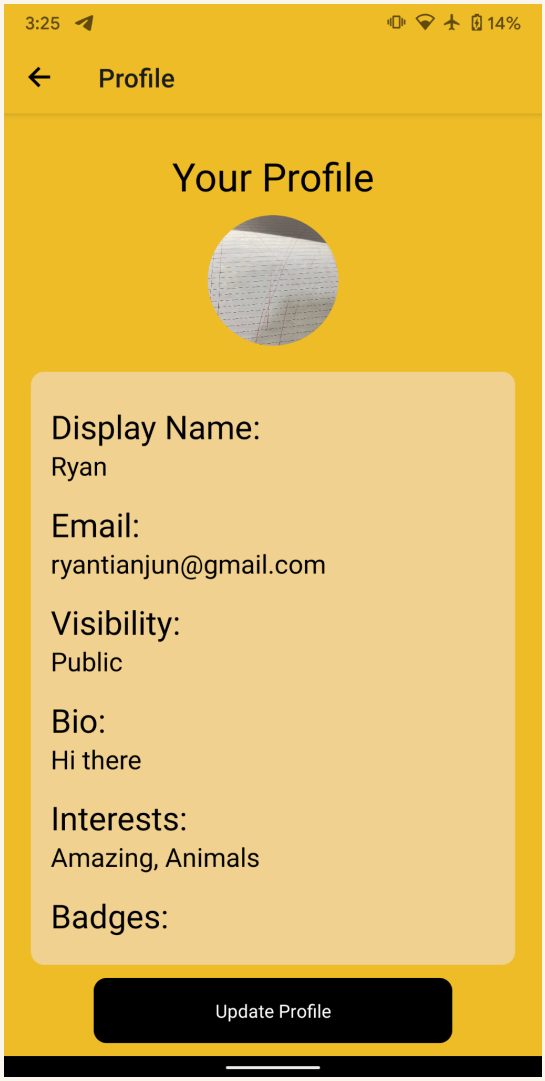




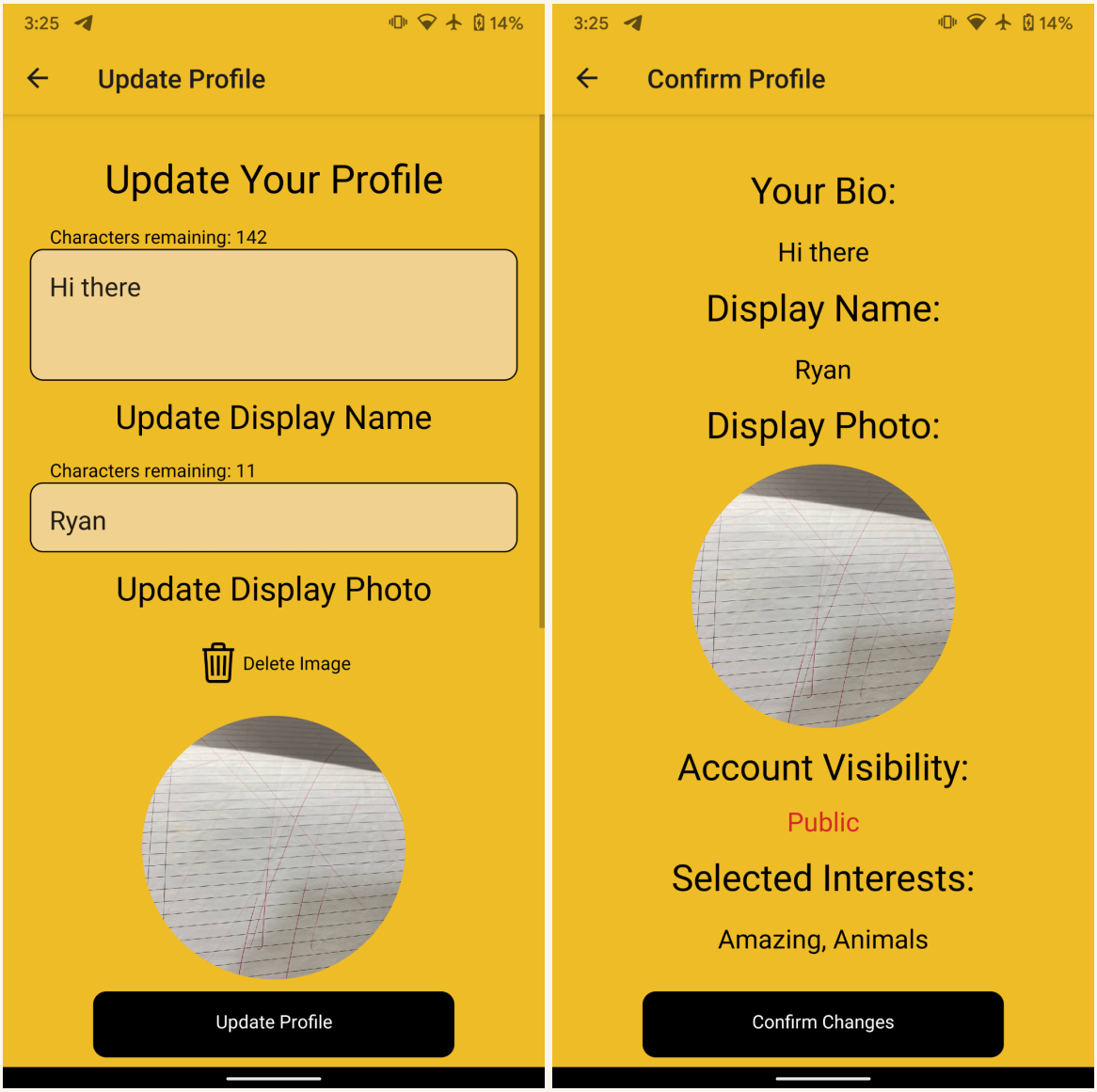
User Profiling

Below are samples of the View Profile Page.





Users can update their biography, display name, visibility and/or interests anytime. They will, however, not be able to alter their badges (which are awarded by the app of course).

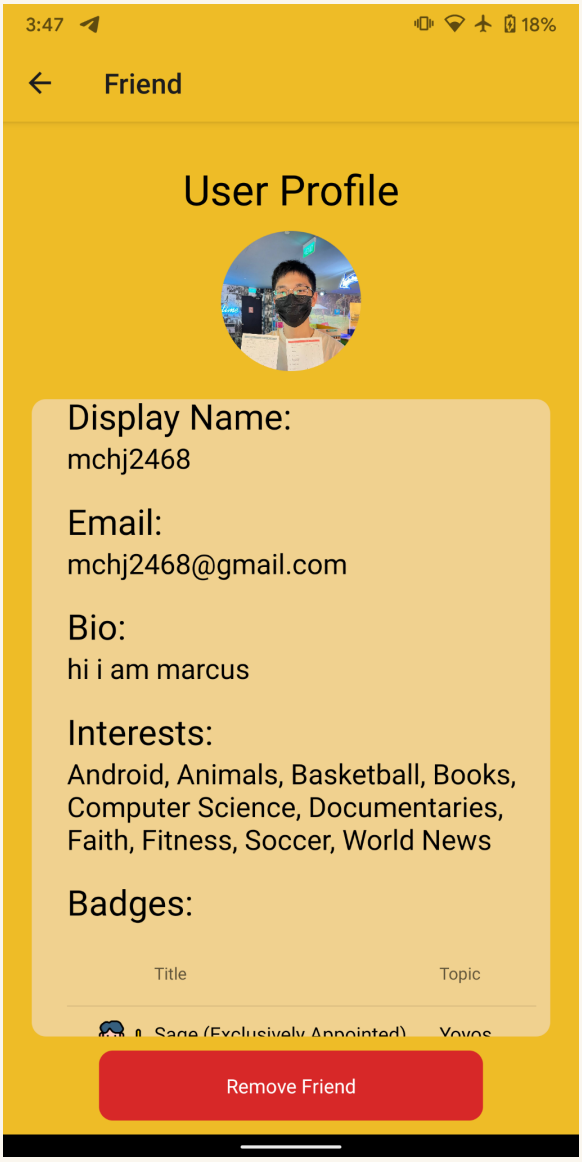


Friend Network

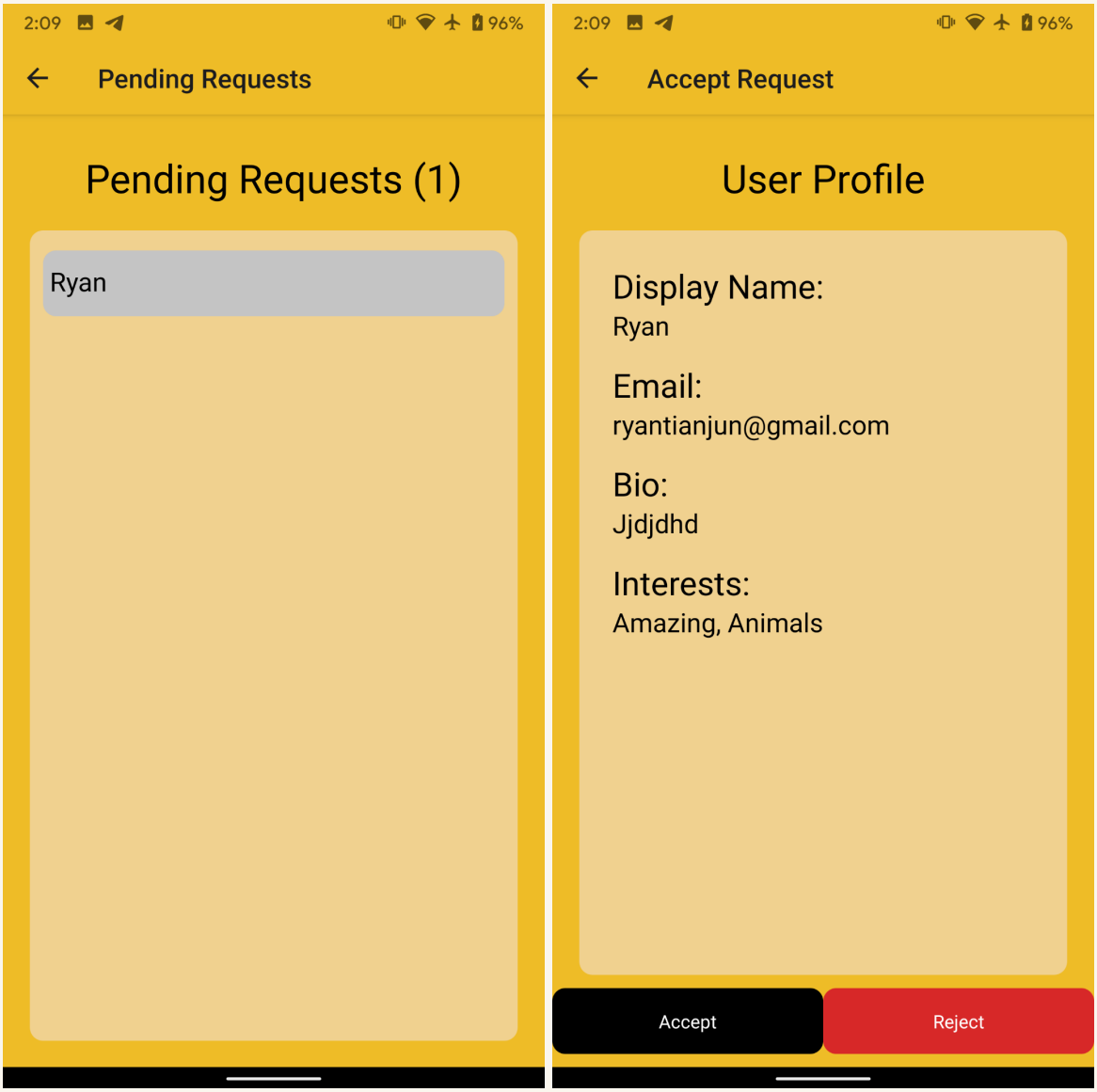
P!ng allows users to have meaningful interactions with new people. The Friends feature is our way of helping people keep track of these bonds so that they can ping their friends again anytime!

To befriend another user, users may add them by searching them up in the Search Bar. This can be done with their display name.

The profile visibility will depend on the user's public/private status.



If the user is private, you will have to wait for him/her to accept your friend request. This can be done at Settings -> Friends -> Pending Requests.

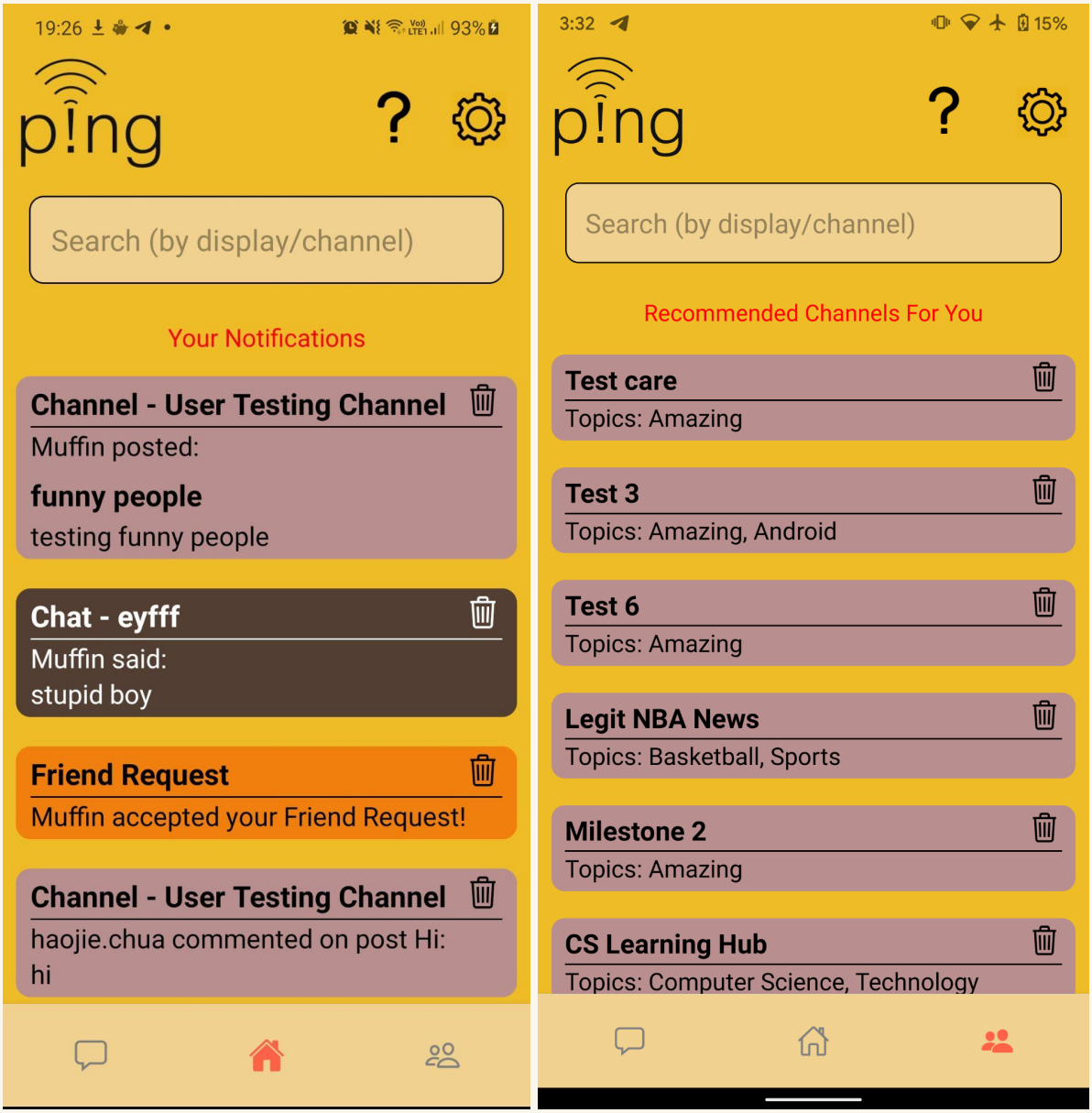


Once you are friends with another user, you may add them in chats, groups and even view their badges! Of course, you can also remove any user as your friend by viewing their profile at Settings -> Friends.

Main Features

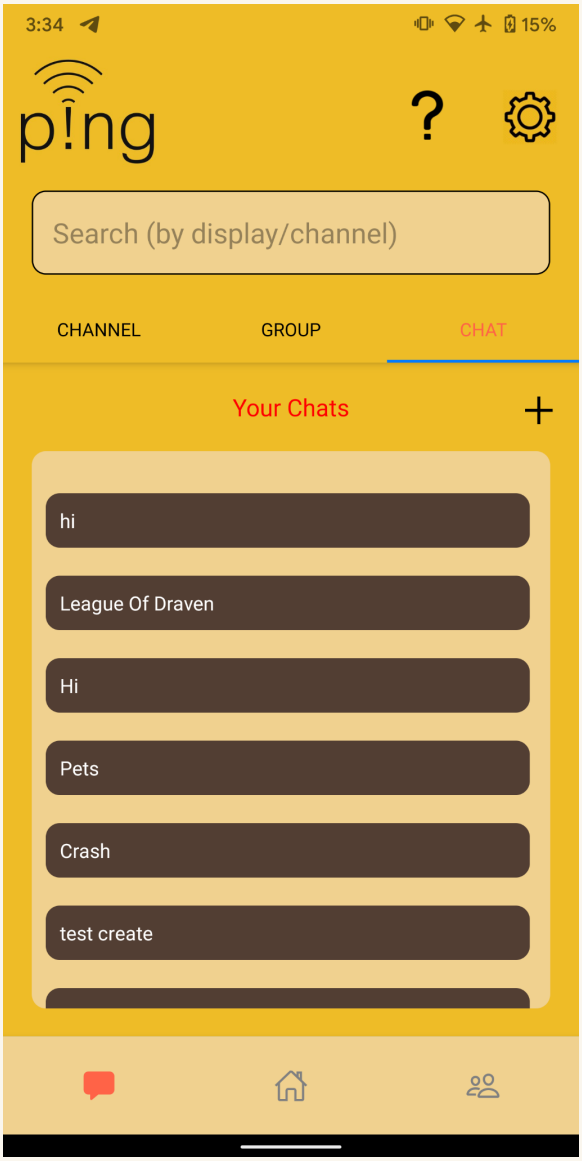
The main selling point of P!ng is its scalability, providing users with the freedom to interact on a variety of levels. Our 3 levels of communication, Chats, Groups and Channels, each serve a specific purpose in the app's ecosystem. These features are accessible as tabs under the messaging icon.

The Home icon leads to a page for Notifications, whereas the rightmost icon leads to the Recommendations page.



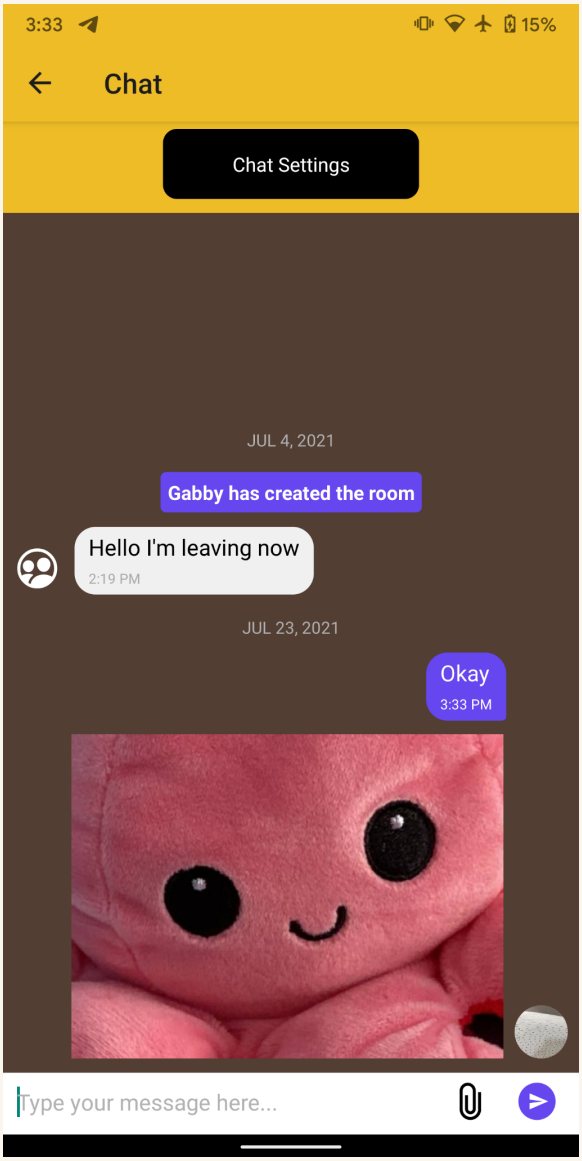
Chats

Chats are simplistic, protected rooms for two users to have a private conversation with each other. Each chat is tagged to certain topics, acting as subjects for the users to centre their conversations upon.

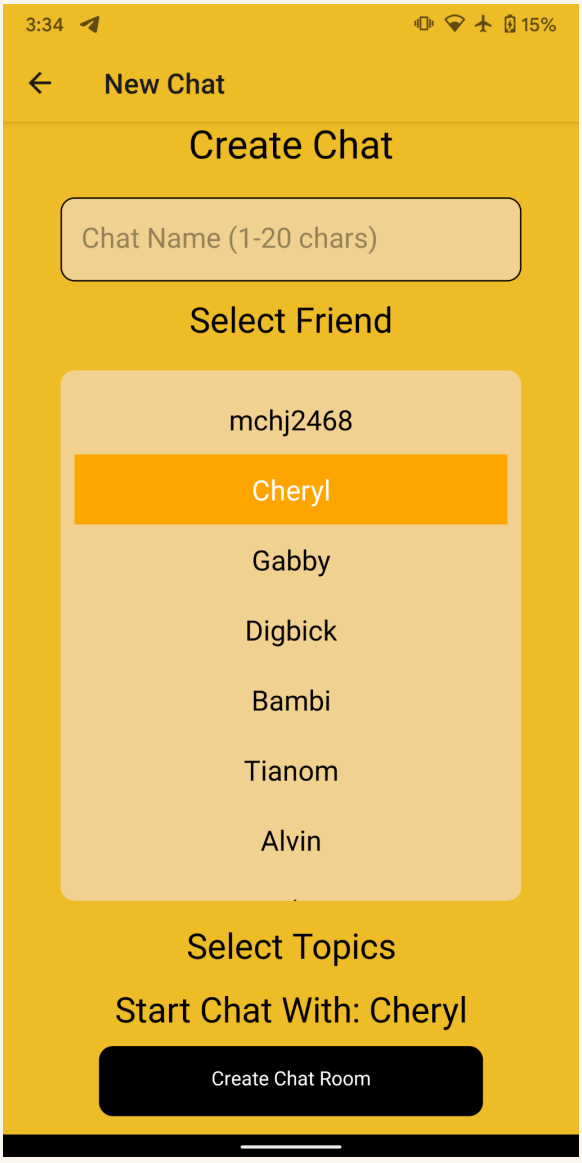


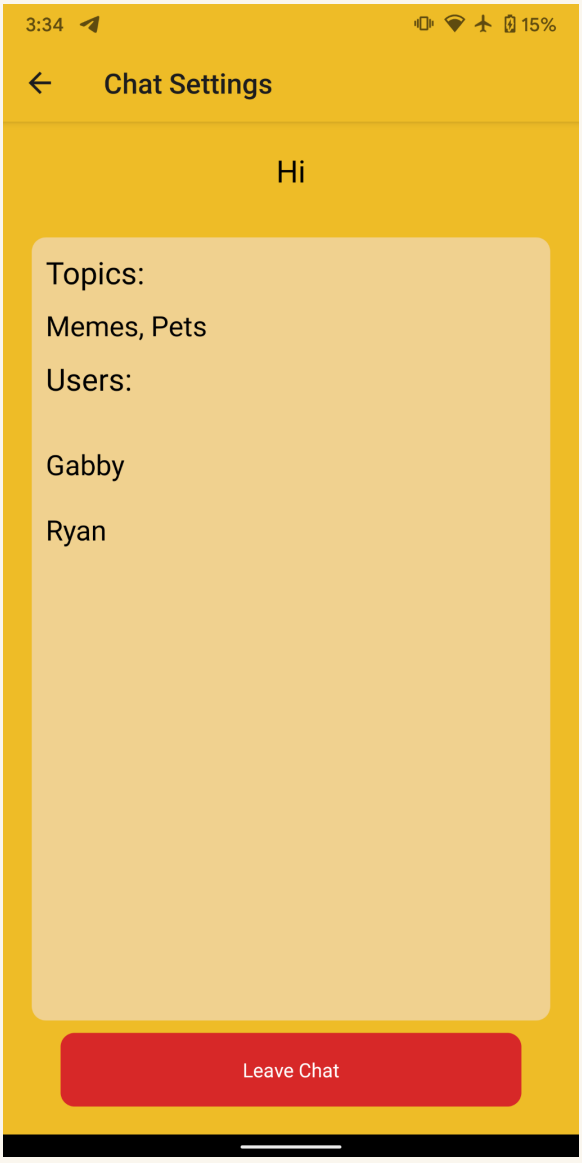
Inside a Chat Room, there is a clean interface for the two users to exchange text messages.

Users may tap on the image for a wider, clearer view of it.



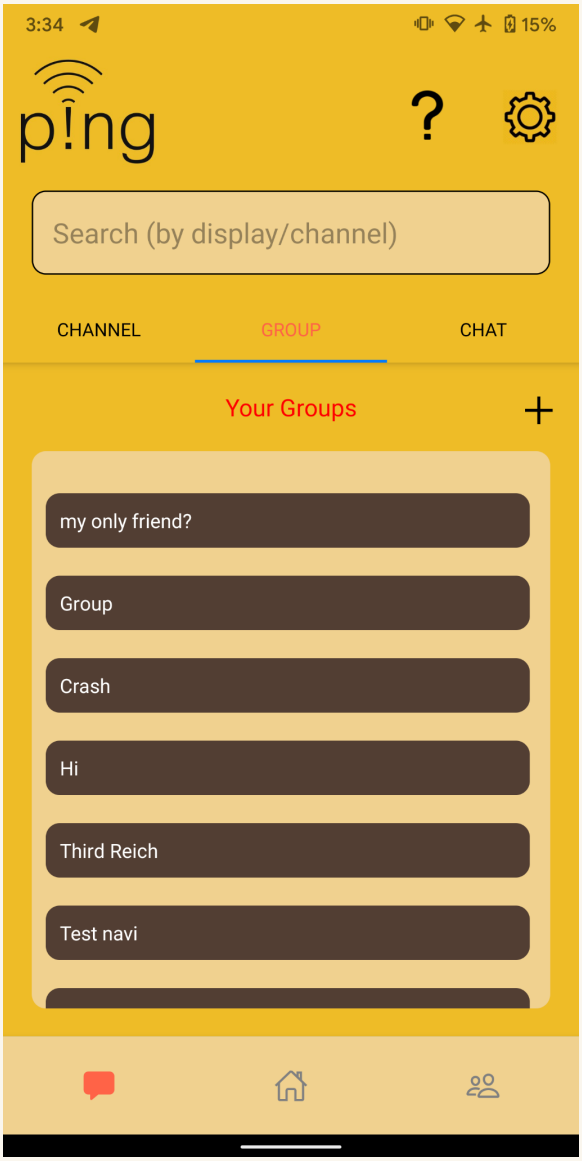
When a user creates a chat, he/she picks one friend to chat with. There is also an option for the chat to be tagged to topics for discussion, however this is optional. These settings can then be viewed from the Chat Settings screen accessible for any chat room.



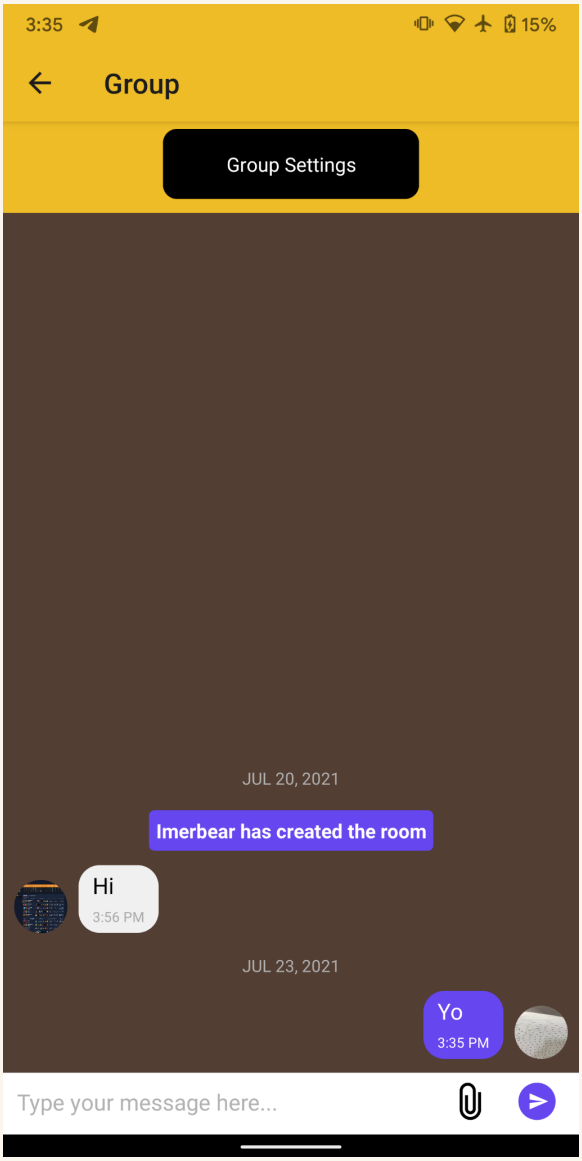


Groups

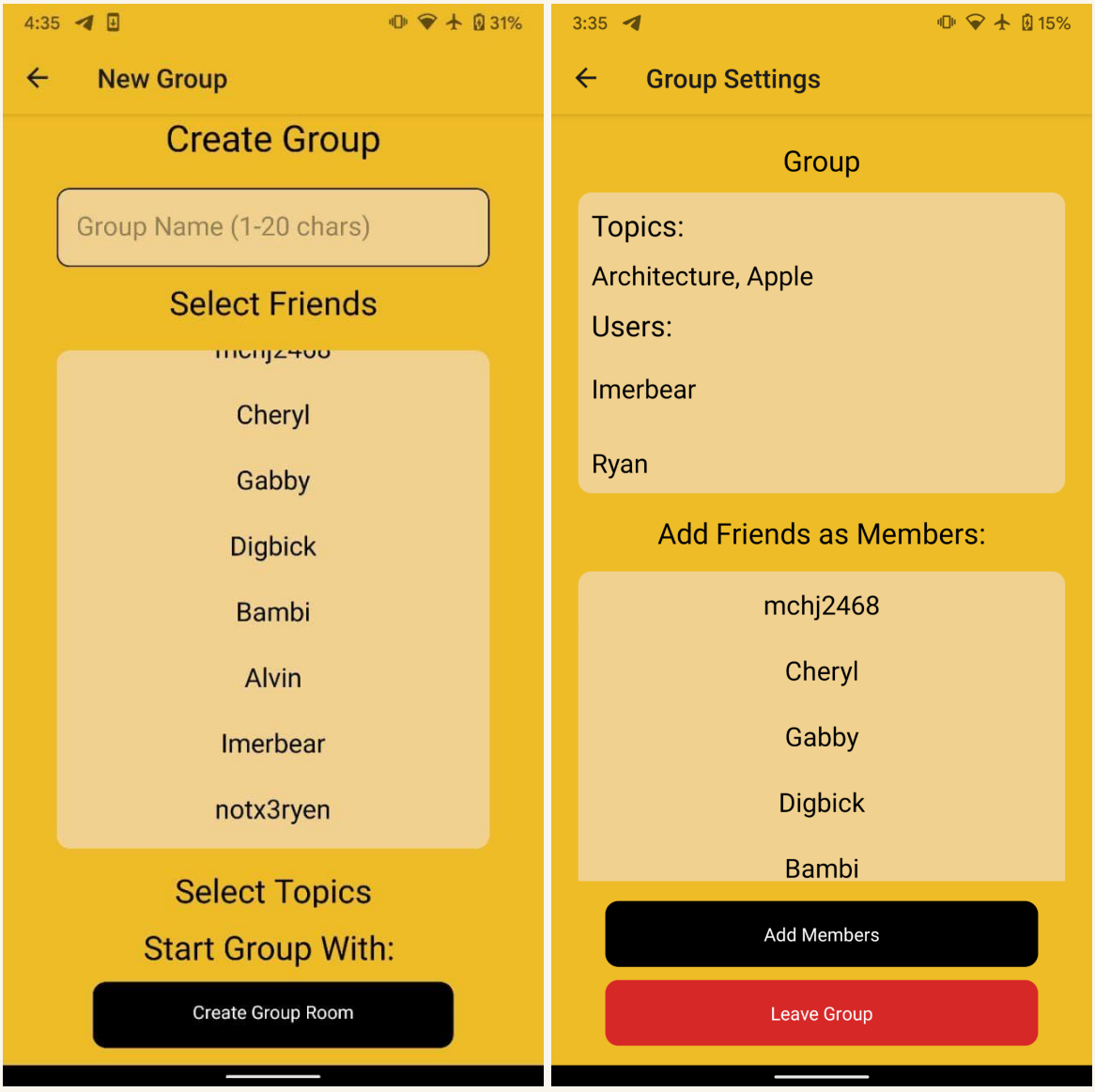
Groups build on the functionalities of Chats, but help to facilitate discussions of several like-minded users. You may create a group with any number of friends inside, and any member in a Group is able to add his/her own friends to a Group room.



The Group Room looks similar to that of Chat, with some minor changes to the UI on the way.



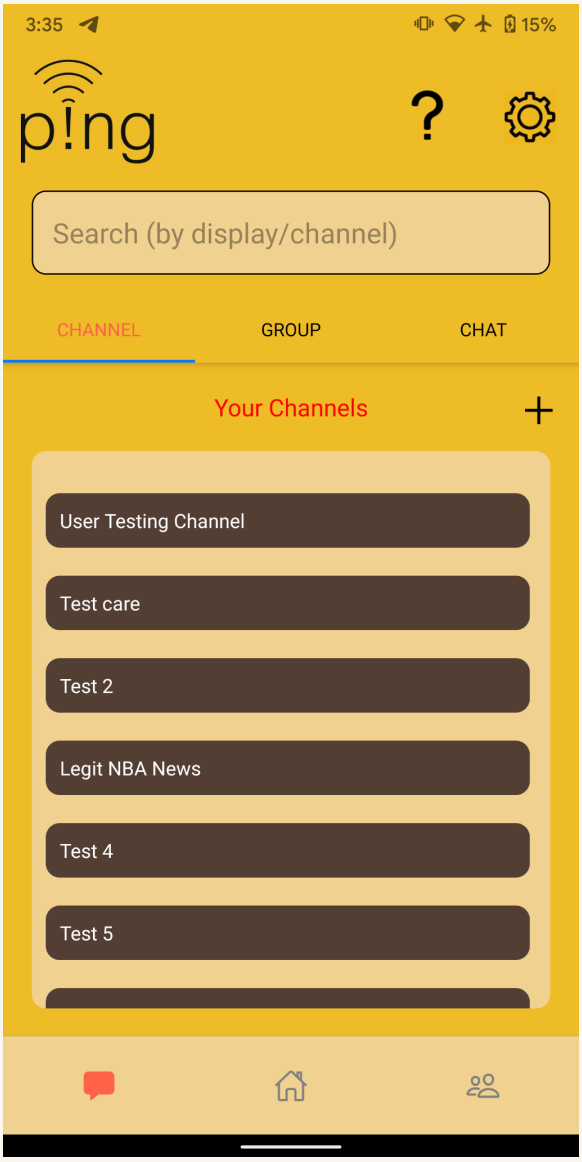
Groups allow the addition of as many friends as required. Each Group is also tagged to some discussion topics. One key difference is that for a Group, any member can add his/her friends to the Group at any point in time. This can be done from the Group Settings screen.



Channels

Channels are like the marketplace for P!ng, where all users can gather for a specific subject matter. Each channel is tagged to 1 or 2 topics, and any user can create a channel.

Channels are public by default, so any user can search and join your channel from the Global Search. Any user is also free to create a post in a channel. Under each post, users can leave comments related to the main post.



Inside each Channel, the Posts are displayed in chronological order.

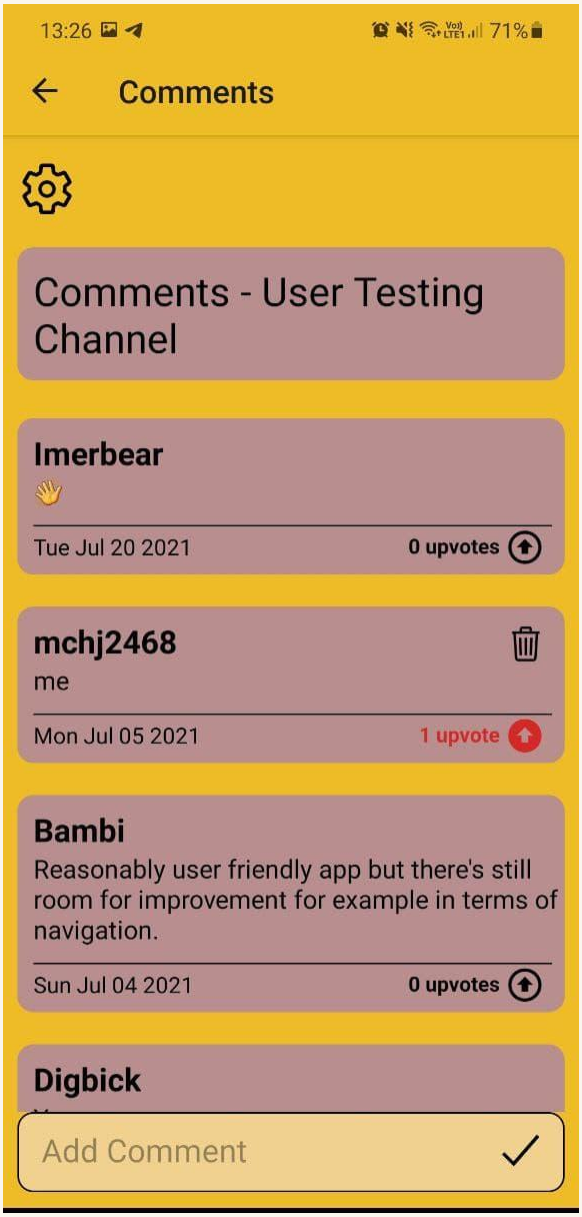
Each Post is limited to 800 characters. Under each Post, users can view/leave comments by clicking on the comments button.

For each post/comment, creators can receive upvotes from others in the channel. Upvotes assist users in achieving topic-related badges, more of which is elaborated upon in the Badges help section.

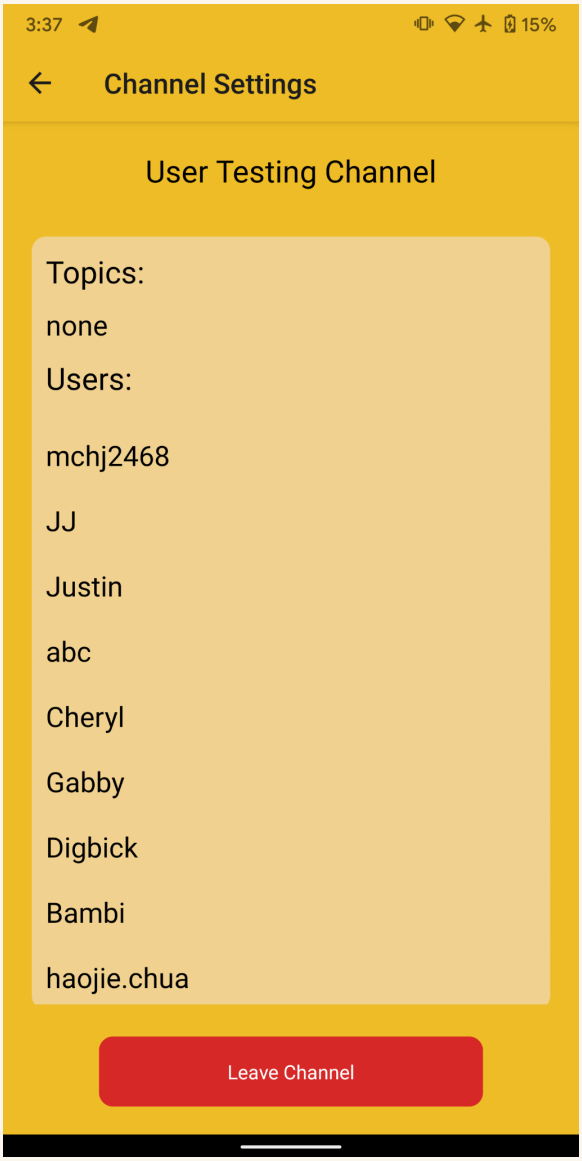
Posts can be created easily by clicking the '+' sign at the top right corner.

In addition, each Post/Comment can be deleted by its creator by simply clicking on the trash icon, where one will be prompted for confirmation.

Below is the Comments section nested inside each Post.



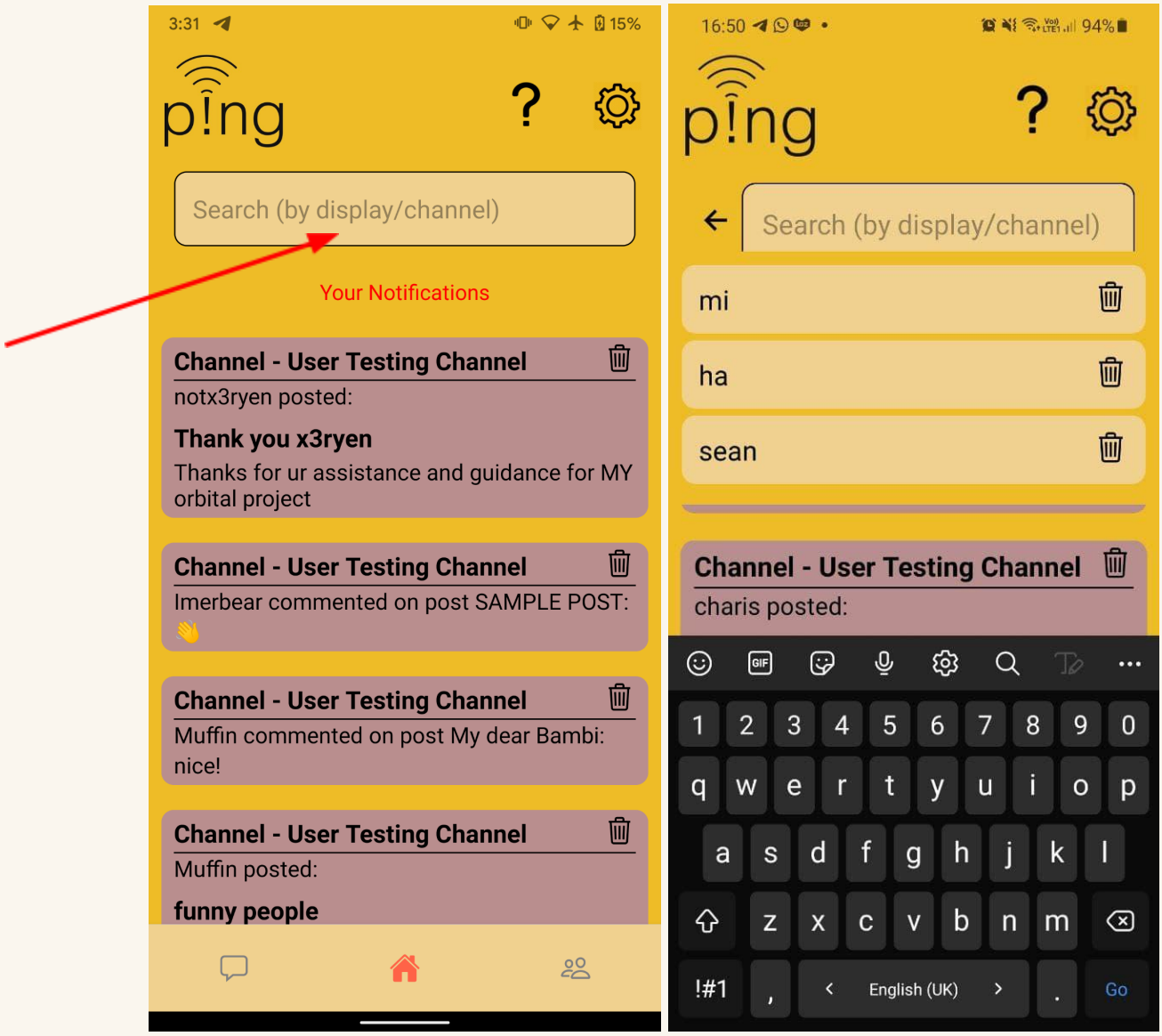
Similar to Chats and Groups, one can view the information for the Channel from its Settings page, and can also exercise the option to leave the Room from this page.



Search Functions

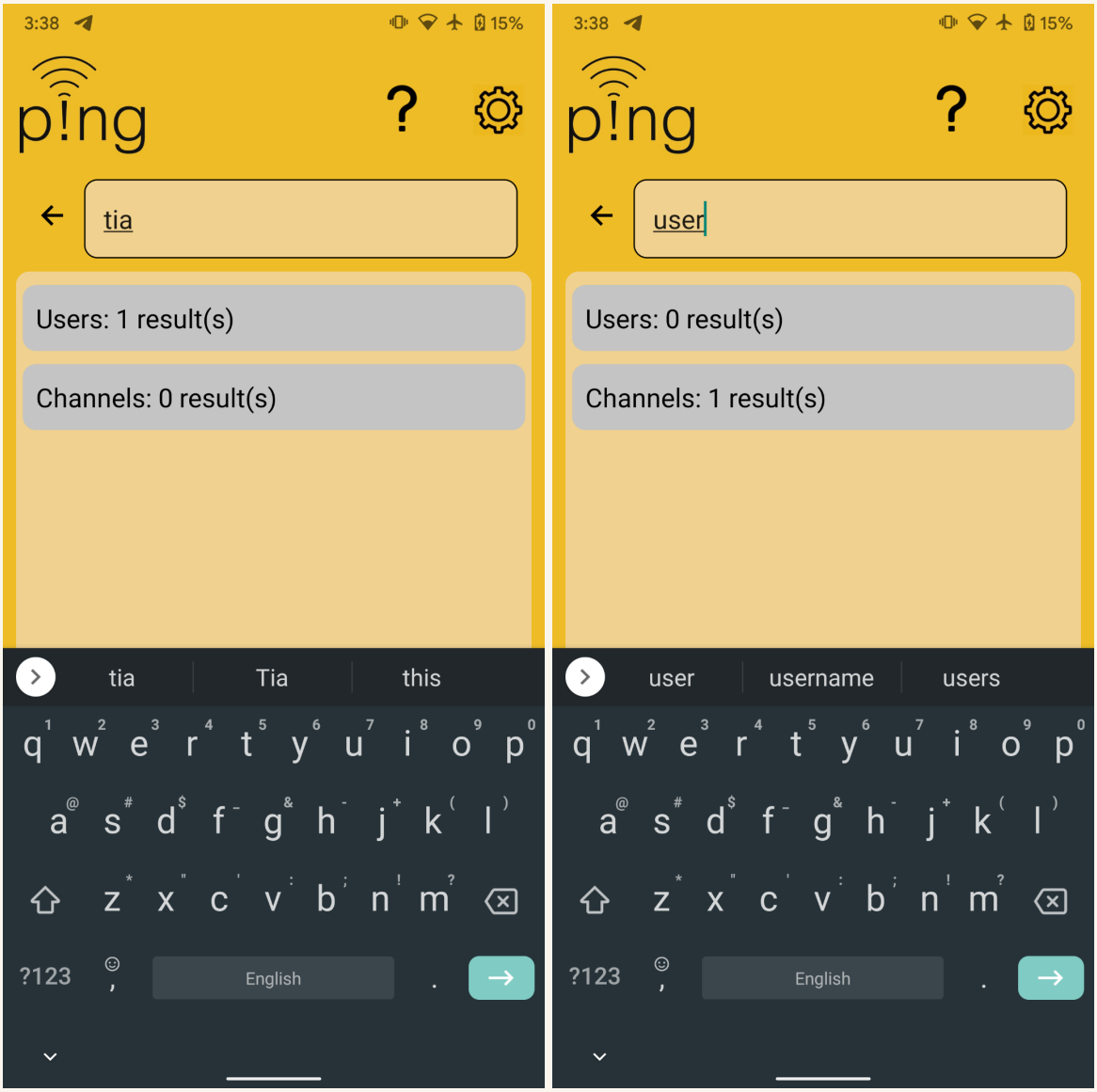
The Search functionality in P!ng opens users up to the global network, where they can reach any user or any channel. It is located at the top of the Home Screen, providing a search which is not case-sensitive.

There is a search history which users can access conveniently, or delete past search records.

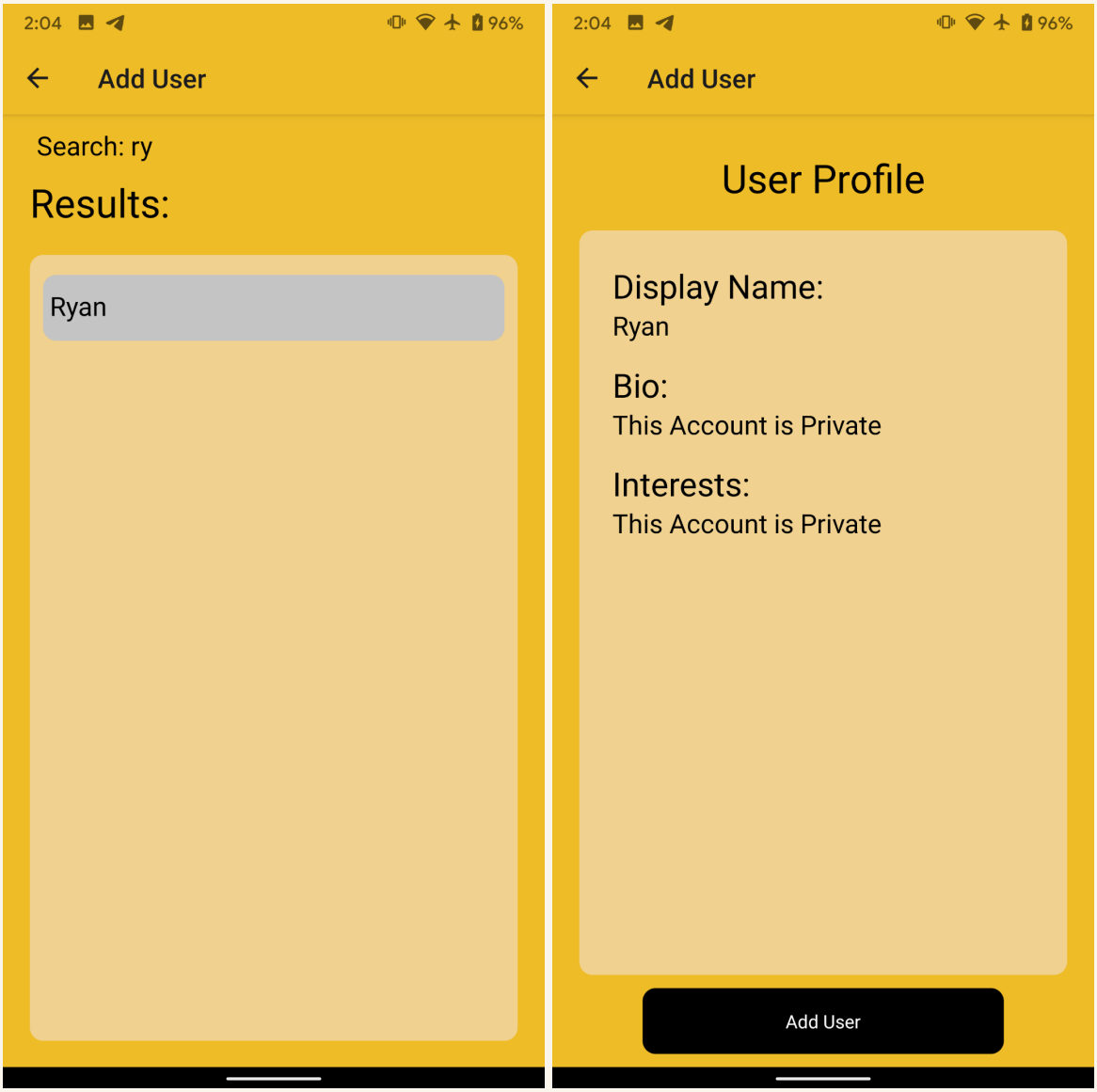


Upon entering a search, search results will be split into users and channels. The user can then narrow down their search efficiently.

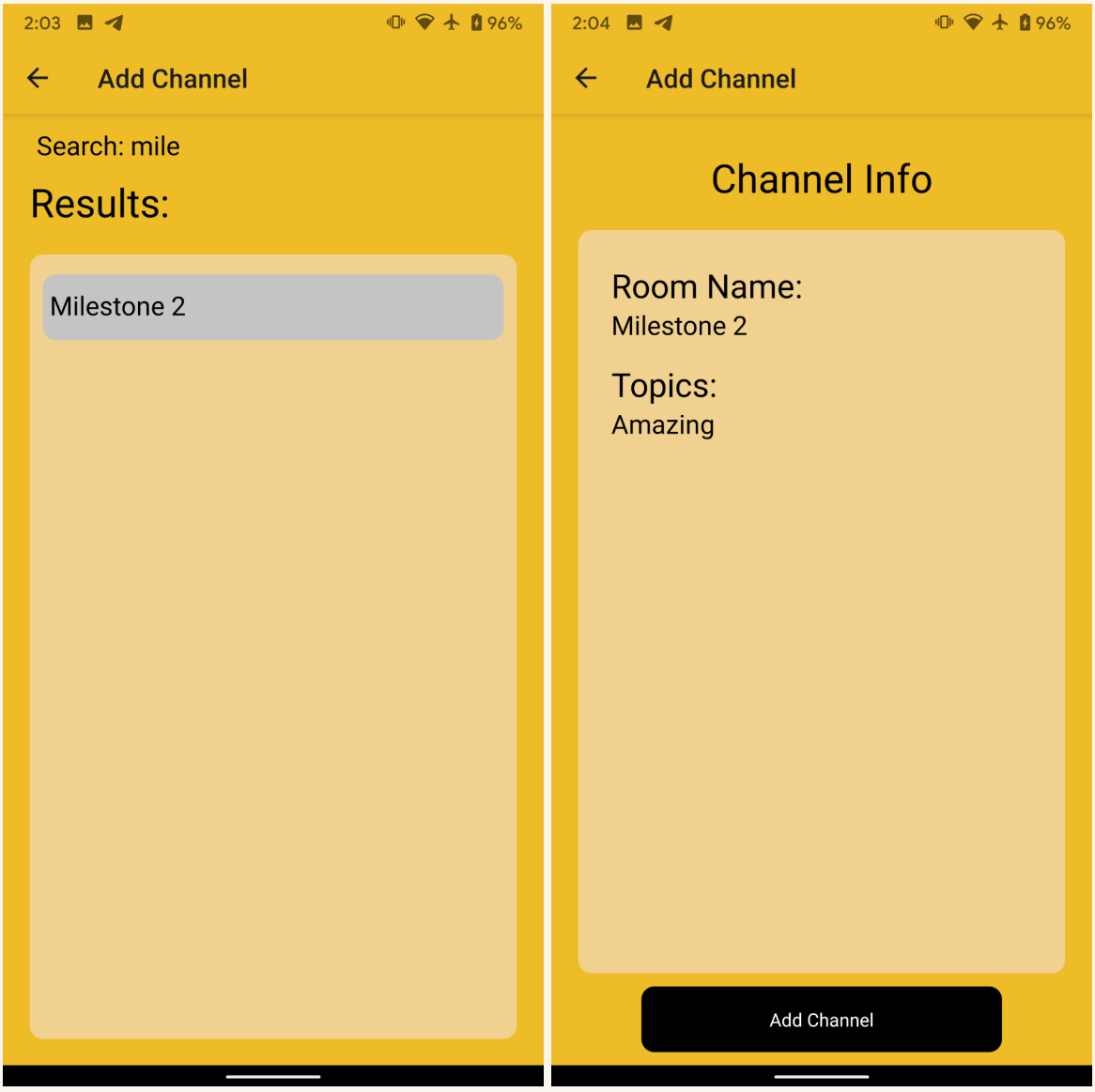
Our search feature is live, so the results are refreshed with every change in text input!



For users which are found through a global search, their bio interests, and badges will only be displayed if their account is public. Else, this information is hidden. The found user may be added as a friend and depending on whether this friend is public or private, you will become friends immediately/send a friend request to the other person.



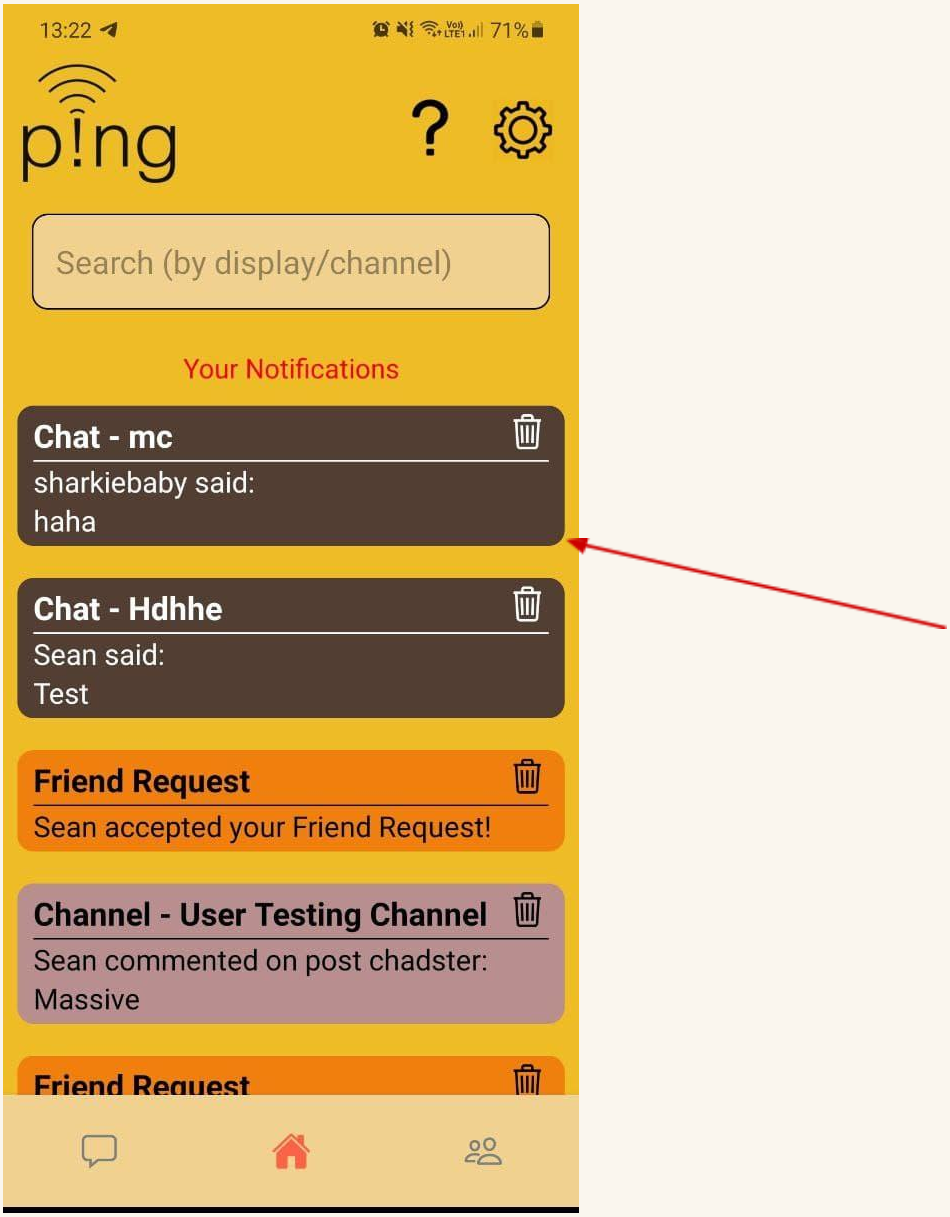
Channels, on the other hand, are public by default. Hence one can join any channel that they find through the global search. The Channel's name and relevant topics will always be readily available through a search query.



Extensions

Notifications

The notifications can be viewed from our Home screen the minute a user logs in. Notifications are color-coded and provide information about new messages, badges or friends. They can also be cleared easily by the user to prevent cluttering. Each notification is live, and clicking on it will link the user to the relevant feature.



Badge Tier System

Badges are used as a differentiator for users. To reward users with stellar performances and help newer users find credible sources of information, badges are used to create a tier system.

Badges are earned by gaining upvotes on posts/comments you create. For example, if I have 5 upvotes in total from Channel X which is on Travel, and 11 upvotes in total from Channel Y on the same topic, I will have amassed 16 upvotes for this particular topic.

It is important to note that badges are topic-specific. Users can view their badges by going to Settings -> Profile.

Currently, these are the badge tiers in descending order, with the topmost being the highest.

Sage: Awarded by the creators to the most exceptional users of P!ng.*
Guru: Awarded to users in the top 10% in a particular topic.^
Thinker: Awarded to users in the top 30% in a particular topic.^

*Sage badges are awarded based on the discretion of the support team.

^Percentiles are calculated based on a server-side algorithm that refreshes every few minutes. There might be a slight lag for the changes to be confirmed!

Some Rules:

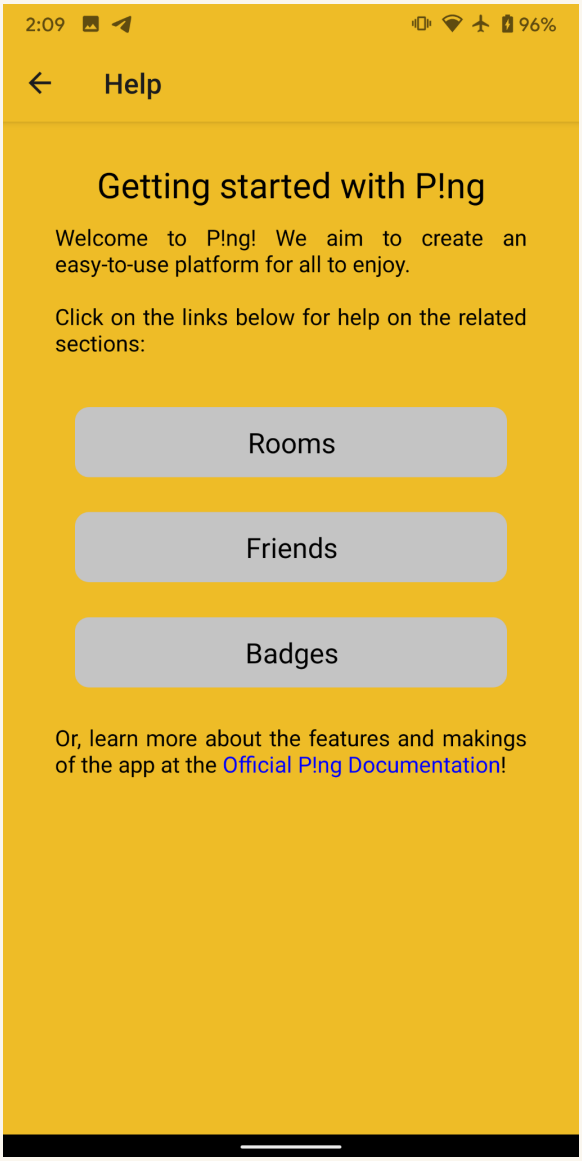
- You can only have one badge per topic, so once you have upgraded to a higher tier, your previous tier of badge for that topic will be removed.

- Once you have attained a certain tier, you will not be downgraded.

Our team will be looking to add more interesting badges to award our ever-growing pool of users! Stay tuned :)

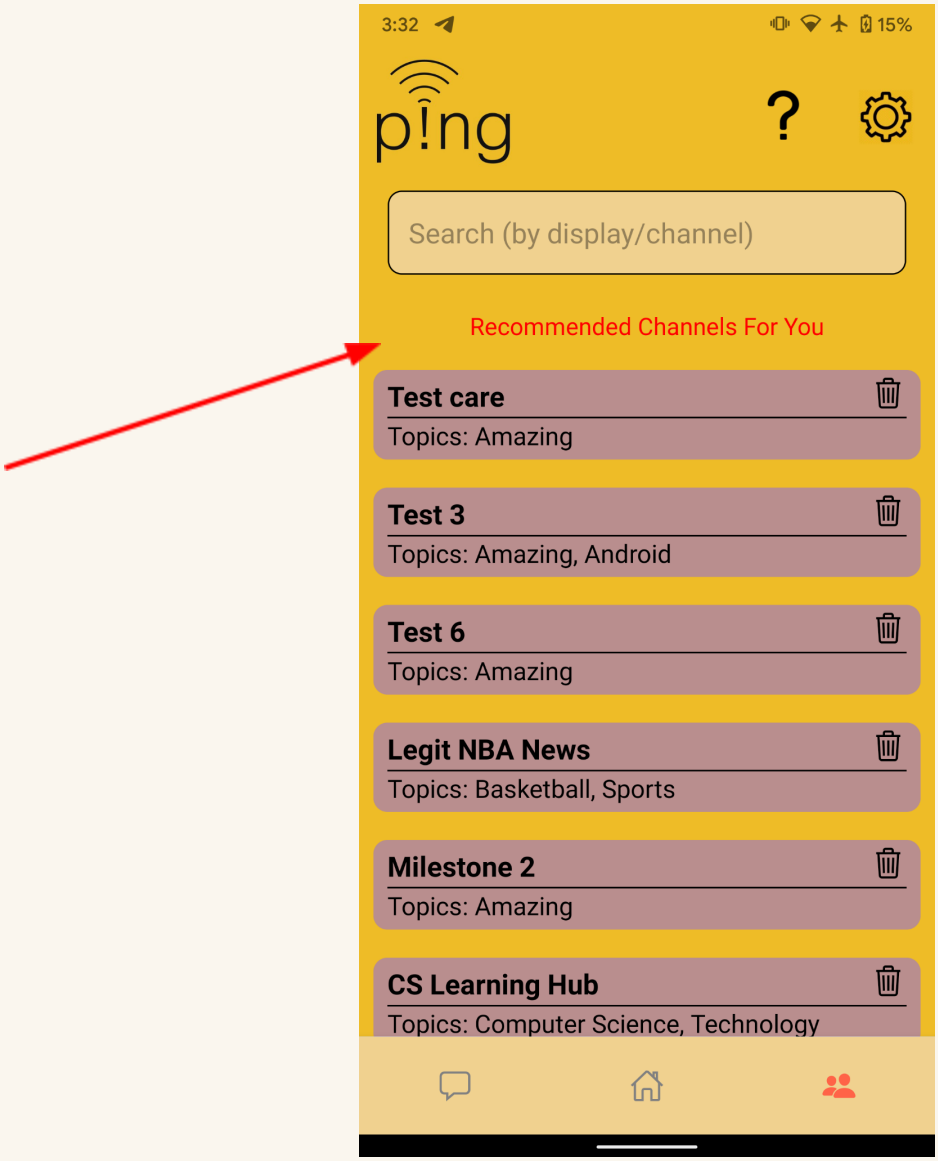
Help Pages

To create a better user experience for first-time users, there is a help page under Settings. Users will be linked to separate pages with more information about the main P!ng features, as well as the official README (this document).



Recommendations

Accessible from the Home Screen, recommendations are channels which are advertised to each user based on their profile. They will be recommended channels based on their interests. Recommendations can be deleted too, if desired.



Software Engineering Practices

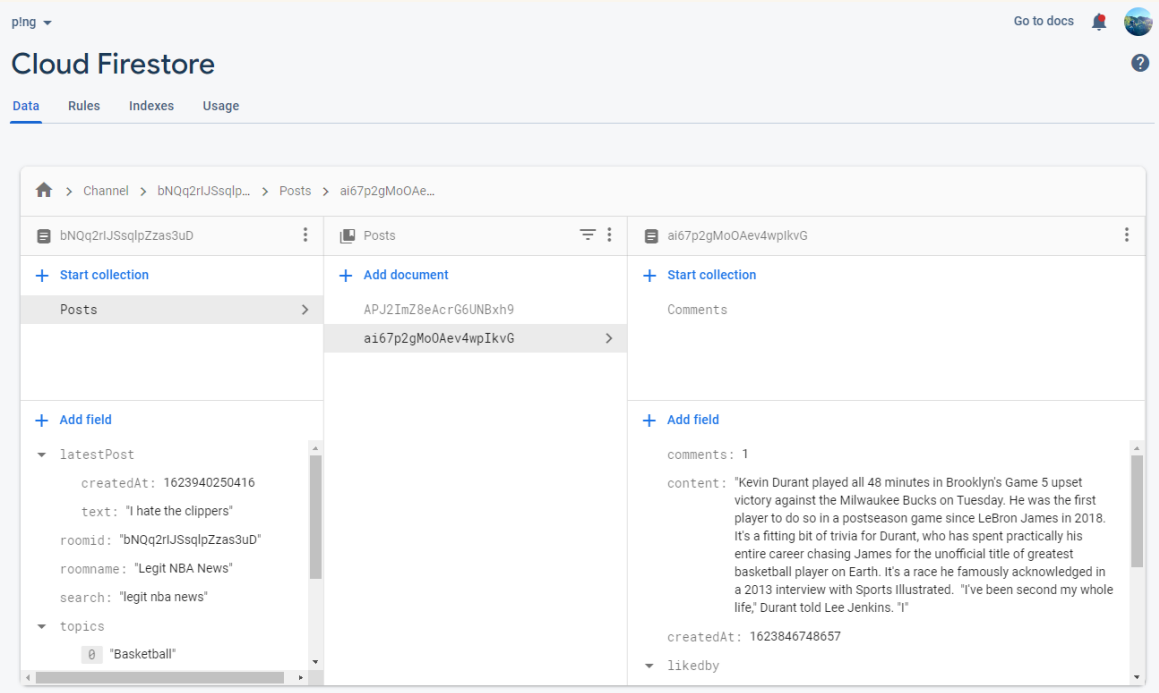
Requirement Gathering - Choosing our Tech Stack

While deciding on the framework for our app and with cross platform support in mind, we came down to two options: Flutter vs React Native. We wanted a framework that would work on both IOS and Android that was modern yet flexible. Eventually, we chose React Native over Flutter due to React's maturity as a platform, having been on the market longer than Flutter. The code we wrote could be easily reused for both web app and desktop app development and there are many tutorials out there for React Native. Additionally, React Native uses JavaScript, which we were more familiar with compared to Dart (by Google) used by Flutter. This saved us substantial learning time and allowed us to focus on picking up other skills. Combined together with Expo, choosing React Native was the framework of our choice.

After collecting ideas, we discovered two major needs.

We needed a relational database that allowed us to make quick reads and writes. The queries should be customisable and support filtering of documents by their fields. The 2 obvious choices were MongoDB and Firebase, but we settled on Firebase because of its user-friendliness making it easy to learn. More precisely, we used **Cloud Firestore** instead of the Realtime Database.

Firestore features 'richer, faster queries and scales further than Realtime Database'. The structure of the database gave us the flexibility to nest collections which was precisely what we needed. This was prominent in allowing us to have structures like Channels → Posts → Comments:



More evidence of this can be found in our Data Model.

Furthermore, the Google ecosystem as a whole provided us with some useful tools which we needed to integrate into our P!ng too. For example, we made use of the **Google Auth** tool to handle authentication for our app.

Other than a database, we also needed a method to pass our users' state between screens as they navigated through P!ng. Such a tool will help us render the correct profile or chat data on each page. We could use this to decide when our data in the backend needed to be refreshed too. Upon research and discussion with our mentor, we decided to use **Redux**, a predictable state container for JavaScript apps. It had great synergy with React Native and reduced our troubles being an all-in-one state data manager.

Aside from these needs, we realised that our extensions will require something more than simple queries from our Firestore database. For our badges, we needed to constantly update the benchmark scores for users to move up the tiers. This required us to run an extremely large scheduled function on the backend. For push notifications, we needed to attach listeners to our notifications collection. These listeners would then have to trigger a function to push notifications to the user. Going back to Google Services we found an apt solution for both these problems in the form of **Cloud Functions**. Cloud Functions could be scheduled and also triggered by using event listeners, hence we used this to our advantage.

p!ng

Go to docs

Functions

Dashboard | Health | Logs | Usage

Function	Trigger	Region	Runtime	Memory	Timeout
postNoti	document.create GlobalNoti/{NotiIds}	us-central1	Node.js 14 Beta	256 MB	60s
scheduledUpda…	every 5 minutes	us-central1	Node.js 14 Beta	256 MB	60s
updatePostNoti	document.delete GlobalNoti/{NotiIds}	us-central1	Node.js 14 Beta	256 MB	60s

Implementation

The central feature of P!ng is the scalable rooms, in the form of chats, groups and channels. Our initial plan was to code up an entire chat API to serve this purpose. However, we faced teething problems and constraints in trying to implement this. We discovered that while we could not use Firebase Cloud Messenger service, we could use React Native's Gifted Chat API as a solution. Gifted Chat allowed us to simplify chats and groups to serve their precise purpose of connecting people in a no-frills manner. The distinction was made for channels, for which we coded the entire backend by ourselves. We did this for channels as we needed a lot of control over the interface so as to implement nested messages (in the form of comments and posts) and upvoting.

There were several challenges that we faced throughout the implementation. Using Cloud Firestore provided us with quick queries but the drawback was we ran into many issues with asynchronous code. With more experience along the way we managed to resolve all of these problems through systematic debugging. Collaboration between both members was vital throughout this process.

Despite the fast reads and writes from our database, a greater challenge was retrieving all the relevant data in a split second and re-rendering our page upon any changes to the database. To fix all these render issues, we had to make full use of React Native Hooks and strategic placement of our queries on different screens. This helped us to achieve smooth refreshes and quick updates for the user.

Lastly, due to the nature of React Native, navigation and passing data (props) between screens proved to be a challenge. It was especially tricky when the screens were in different stacks (nested or not). Data was not passed properly and we ran into many undefined state errors while navigating. To overcome this issue, we grouped relevant screens into the same stack, and nested stacks in a logical and efficient manner, while passing props only when necessary. This simplified the logic for an otherwise complex process.

Design

Our two primary goals for our UI/UX were usability and information architecture. We also strived to improve the visual and interaction design as secondary aims.

Information architecture was particularly important given the numerous sub-functionalities in our app. We needed clean navigation that was not laggy. **Usability** was important as we wanted our app to target all age groups, so it had to be dummy-proof with obvious prompts along the way.

Before commencing on our code, we did **wireframing** using Figma so that we could envision our idea. The initial thought process was to have 3 main screens in Home - Rooms, Notifications and Just For You. Within Rooms, we implemented 3 tabs (Chats/Groups/Channels) which could be toggled by swiping horizontally on mobile. With our Figma prototype as a blueprint, we stuck to this design throughout the entire course of development.

To achieve our UI/UX goals, we stuck to some key principles in the app design:

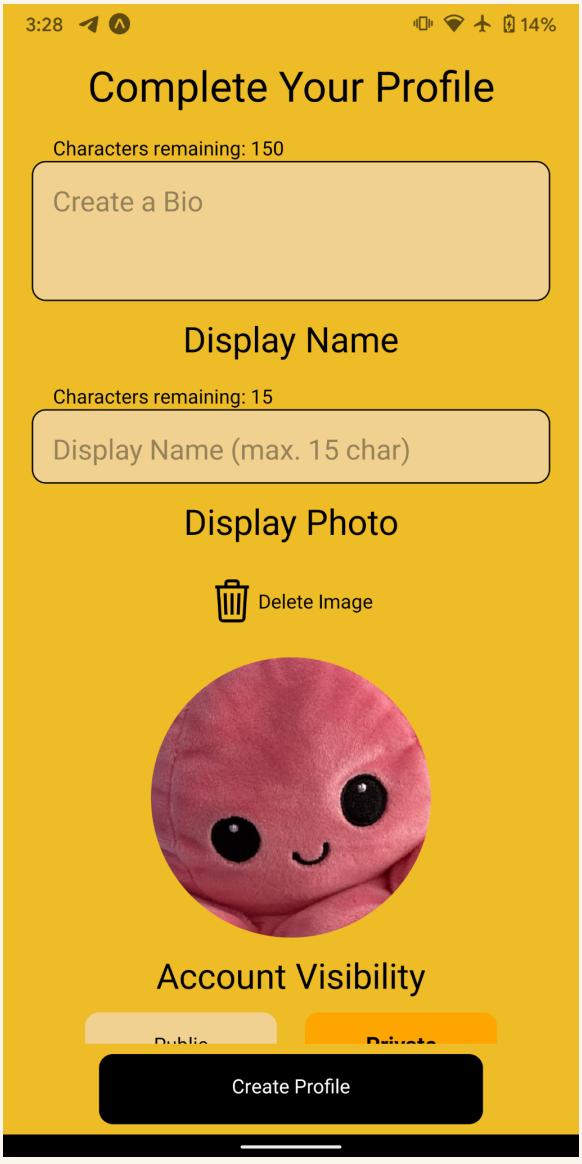
1. **Hierarchy** is best illustrated with the Site Map in the following section. The main navigation menu contained a search bar on top of the main sections, which allows the user to get more specific down the hierarchy. Within each page, we followed a strict visual hierarchy where the headers were the most prominent followed by content like tables or buttons in the body.

2. **User Control** involved reasonably maximising the functionalities given to users. We did this in multiple ways. Firstly, alerts were given when users made any 'illegal' actions, like registering an account with an existing email, creating a channel with >2 topics etc. Secondly, we initialised a back button which allowed users to backtrack in the navigation with great ease. Finally, our help page gives users a succinct 'mini tutorial' on how to use all the features in P!ng.

3. **Confirmation** was an important feature for our app which involves a lot of user decisions. For decisions like updating profile information, deleting posts or leaving a chat, the user was given an alert for the chance to review their decisions.

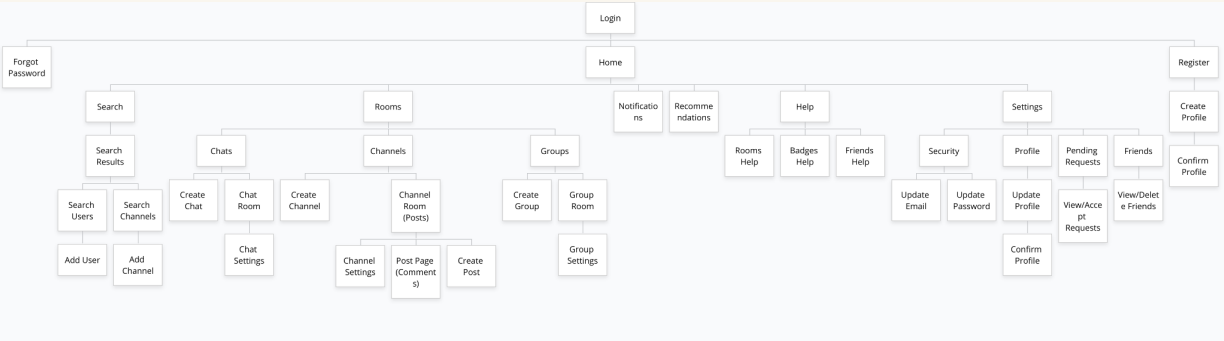
4. **Consistency** was achieved by standardising the colors, font, text sizes and buttons. The same layout was used for all tabs to improve the visual coherence. The color palette that was applied on every screen can be found below.

5. **Accessibility** within an application will reduce the cognitive load of allowing users to maximise enjoyment and utility of the app. We made use of basic and relevant icons throughout the app. In addition, we labeled every text box which required input from users, with placeholder text providing any additional information. The confirm button was also fixed at the bottom of the screen for easy access. Attached is an example from the Register Profile screen:



On a side note, the color scheme was handpicked to be easy on the eyes and enhance readability in different light settings, catering to users of all ages.

Site Map



For a clear PDF version of our Site Map, click on the thumbnail above.

Planning - Data Models

The myriad of features for our app involved dynamically changing data and passing of data between many entities. We needed to organise the data being passed between users, rooms, topic tags, our global state etc.

A useful tool for us to model our data was Entity Relationship Diagrams, or ERD in short. Being a popular structural diagram for database design, we adopted this for our planning. Its main advantage was to show the precise relationship between entities and their attributes, expressing how and where the links were made.

We used a Physical ERD which modelled closely after our actual relational database in Firestore. We assigned each field to a data type (string, map objects, trit etc.) and allocated a primary key for all important documents (doc id) for easy verification. A Physical ERD made our job of organising backend code much cleaner.

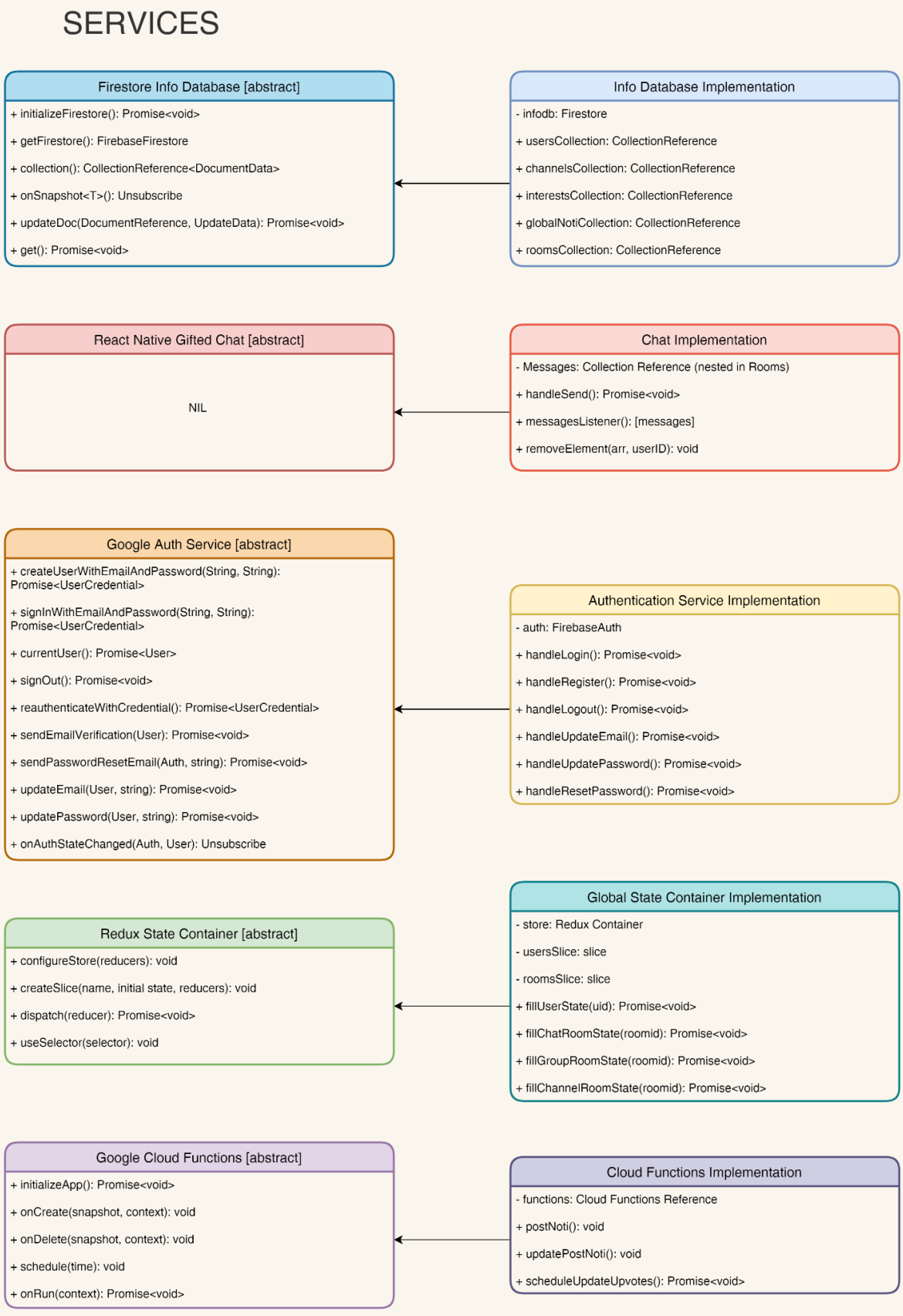
Specifically, this model gave us the useful big picture view that we needed to craft our databases and decide which fields/attributes were essential. This bird's eye view visualisation gave us clarity throughout the project. With the ability to identify mistakes, we could also afford some flexibility by tweaking our models with the ERD, before applying changes to our database in Firestore.

We could also debug our database easily. While Firestore's backend was very user-friendly and allowed us to spot bugs by viewing the information in its hierarchy, we could spot how the bugs had a lead-on effect by studying our ERD. For example, while implementing badges, we discovered that there were inconsistencies in our topics array due to some minor spelling errors. Using our ERD, we managed to trace and debug some other affected attributes which were located in entities like Users and Channels.

Along with our ERD for Objects below, we created a separate data model that elaborates on the Google Services that we used. As mentioned earlier, we made use of Firestore for storage, Auth for authentication and Cloud Functions for running heavy queries behind-the-scenes.

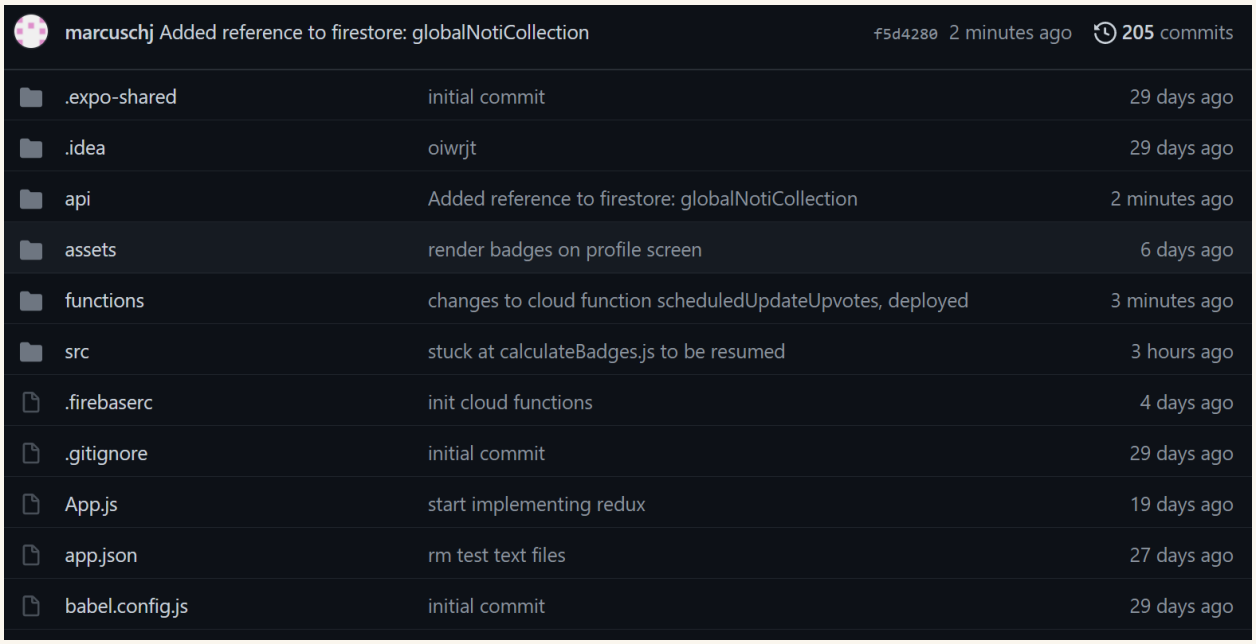
Physical Entity Relationship Diagram (ERD) for P!ng:

Services used in P!ng:



Version Control

Throughout the course of the project, we worked on overlapping parts in our code. We made use of Git and GitHub to merge conflicts seamlessly into our repository. This was done through maintaining separate branches, for working and releasing. We made use of informative commit messages so that we were always aware of the specific changes for each version.



Testing

User and Unit Testing were a huge part of P!ng's development. Between the team, testing is carried out on a daily basis and/or after a feature is implemented.

Methods of testing employed:

1. Alpha testing

   Before every milestone, we thoroughly tested all features that we promised and tried our very best to break the code and uncover bugs. This was primarily done together while writing the README and use cases, as both are highly related to this method of testing. Afterwards, we would work on the discovered bugs and test them again to ensure the bugs are fully patched.

2. Ad-hoc testing

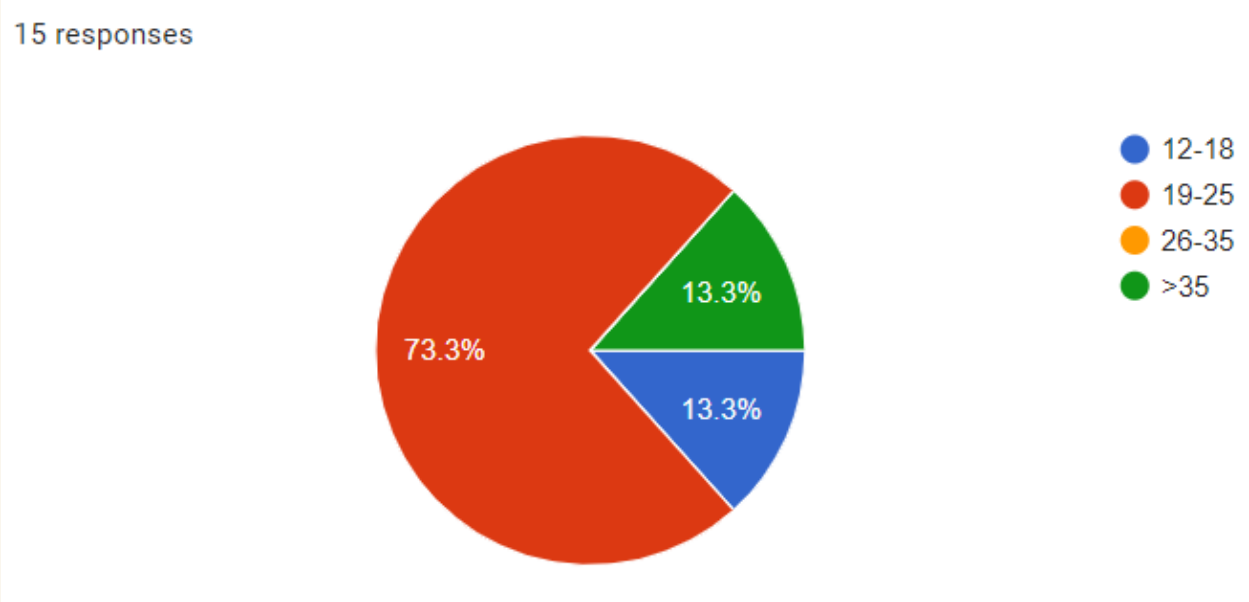
   While we were making the separate components, we would test them on the fly - e.g. whether the channels post was sent to the backend firebase with the correct info, adding console.log (printing to system console) to check if there were any undefined information, or simply checking if the information was flowing smoothly throughout the app, from firebase to redux store to our navigation components. We would then fix the bugs on the fly and for those unsquashed bugs, we would take note of them and fix them later as some bugs may have arised from a separate component.

   For a detailed list of our use cases, click here.

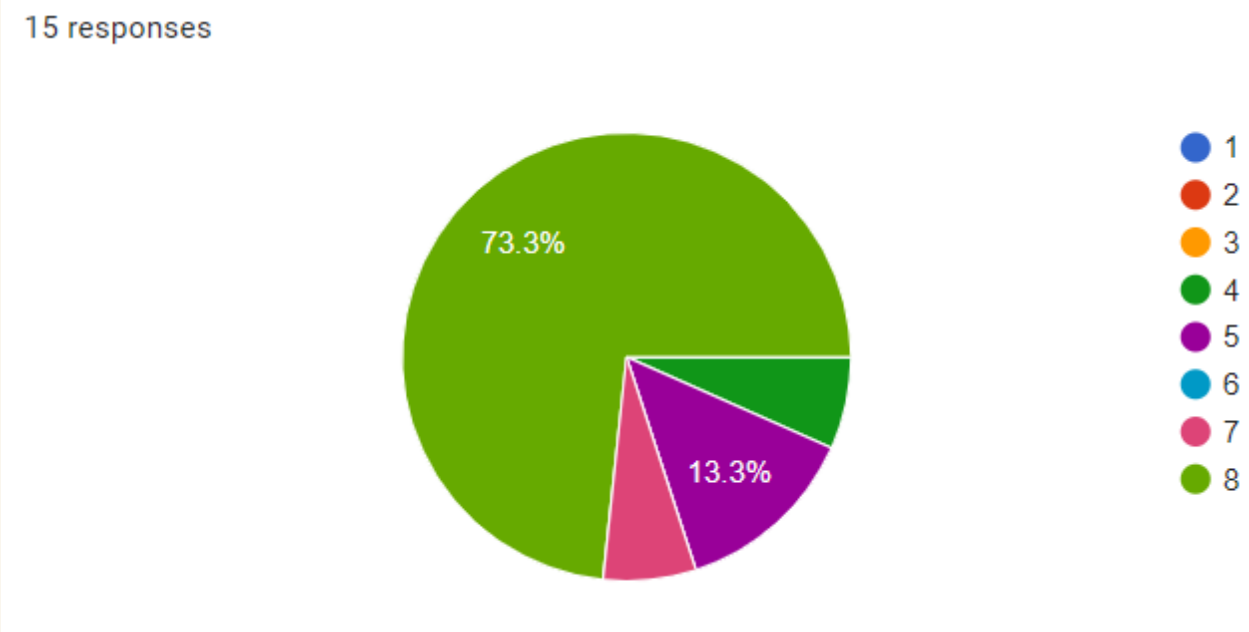
3. Usability testing

   Just after Milestone 2, we conducted a usability test to seek improvement pointers for our UI/UX. A representative group of youth and adults were tested. The participants were given a list of tasks to perform which required them to test out the key functionalities in P!ng. Through the participants' verbalisation of thought process and success indicators for the tasks, we manage to make the user experience much better. At the same time, we managed to identify a few bugs and rectify them. It is important to note that the participants are within the social circle of both app developers.

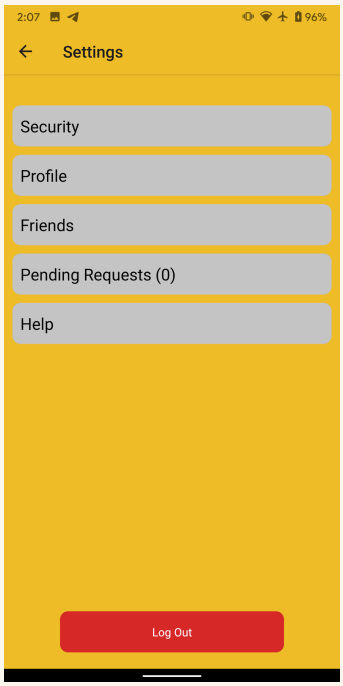
   In summary, we collated a total of 15 responses from a range of age groups:



We chose to differentiate the age groups as different groups may have different responses to the app since they have differing experiences with technology. Most of the participants were students (86.7%) and have never tried the app before. Participants were given a list of tasks (8 tasks) and had to finish the tasks without guidance. While the participants were going about their tasks, we took note of the patterns that emerged (e.g. where they looked to update profile, any mis-clicks). We found that the vast majority of participants were able to complete the tasks with ease:



The tasks that participants may have found more challenging were those that were deeply nested in the navigation stack (e.g. updating profile requires navigation to settings screen -> profile -> update profile -> confirm changes). This led us to change the navigation stack and create a drawer for the settings screen:



Before

After

This was among the numerous changes we made as a result of user testing.

Some of the key findings from the test were:

- Colour palette was distracting
- UI changes to the logo, upvote icon etc.
- Differentiation between chats and groups
- Placement of buttons
- Zoom to view image
- Improved navigation
- App not refreshing
- + Numerous bug fixes

A copy of the Google Form deployed for the usability testing is included here.

4. Back-end testing

   Cloud Functions were used to update the app notifications and also periodically update badges as mentioned above in the README.

   To test Cloud Functions, we used **Firebase Local Emulator Suite**. The suite enabled us to create a mock up database to push and pull information from, and was vital in testing Cloud Functions. Initially we tested Cloud Functions by uploading them directly to Firebase, but that proved to be problematic as we were unsure if the functions were running correctly since the Firebase dashboard does not provide any feedback on Cloud Function errors and there was delay when the functions actually ran and the changes being updated on the app. The local emulator suite showed us real time updates and we were able to debug and test our functions and features before pushing them to our back-end Firebase.

   To check out the testing done on the Suite, click here.

5. Component testing

   For Firestore, we needed to ensure that the documents were being updated correctly in their collections and that the collections were being created in the right documents. We needed to ensure all registered users were able to login and were sent a verification email.

   Upon completion of a major component, for e.g. channels, we would extensively test out the component before integrating it with other components. This was done using our use cases and testing the code extensively. By ensuring that the component fulfils the requirements and is bug-free, we can then safely integrate the component with the other components.

   Component testing was done formally using **Jest**, a JavaScript testing framework that works well with React Native. It was used to test rendered output. Specifically, it does so using Snapshot Testing, a powerful and low-level tool enabled by Jest.

   Critically, **React Native Testing Library** was also used to supplement the component testing. It allows the use of user-centric fireEvent methods to simulate user interactions.

   Our component testing involved simulating user actions, and then testing if the following backend processes were completed properly:

   - Authentication
   - Redux Integration
   - Cloud Firestore Integration
   - Self-created Algorithm for Badges feature

   For a detailed run-through of our automated testing, please watch this video.

Project Timeline

We adhered closely to the timeline below throughout the course of our project:

Agenda	Timeline
Liftoff	13 May
Learning Stage - Formulation of Project Idea - Market Research - React Native and Firebase Courses	Week 1 (17-23 May)
Authentication *Main Functionalities: Login, Register, Update info, Reset password* - Set up backend using Firebase (with Email verification) - Set up UI + Navigation for authentication system	Week 2 (24-30 May)
Evaluation Milestone 1: Ideation - Idea - Features - Design - Development plan - Tech stacks - Proof of concept (Able to login) - Documentation	31 May
User Profiling *Main Functionalities: Profile page, Biography, Interests, Display Name* - Link backend to store user data - Set up UI + Navigation for updating/viewing profile	Week 3 (31 May-6 Jun)
Friend Network *Main Functionalities: Add/Accept Friends, Toggle Public/Private, View Friends List* - Link backend to store friend requests and connections - Set up UI + Navigation for viewing/adding/confirming friends	Week 4 (7-13 Jun)
Chats *Main Functionalities: View Chats, Create Chat, Use Chat* - Integrate Gifted Chat for chat interface + Redux to store room state - Link backend to Gifted Chat for storing rooms and messages nested - Add Chat Room Settings for viewing users/topics, leaving chat - Add Create Chat for user to pick a friend for chatting	

- Set up UI + Navigation for Chats Screen, Chat Room, Settings, Create Chat	
Groups *Main Functionalities: View Groups, Create Group, Use Group, Add friends* - Integrate Gifted Chat for group interface + Redux to store room state - Link backend to Gifted Chat for storing rooms and messages nested - Add Group Room Settings for viewing users/topics, leaving group, adding friends into existing group - Add Create Group for user to pick >1 friends + topics for chat - Set up UI + Navigation for Groups Screen, Group Room, Settings, Create Group	Week 5 (14-20 Jun)
Search Functions *Main Functionalities: Search Friends (by display), View Profiles, Search Channels* - Link backend to query for friends using display / channels by name - Refine search query to render similar results too (e.g. case discrepancies) - Set up UI + Navigation for search bar, search results	
Channels *Main Functionalities: Search/Add Channels, Create Channels, Create Posts/Comments, Upvote Posts/Comments* - Design UI for channel interface - Add Add Post/Comment functionality - Link backend to store channels with nested posts with nested comments - Add Channel Settings for viewing users/topics, leaving channel - Add Create Channel for user to pick 1-2 topics - Add Upvotes feature - Redux to manage changes for upvoting + commits to channels - Set up Navigation for Channels Screen, Room, Settings, Create Channel	Week 6 (21-27 Jun)
Evaluation Milestone 2: Prototyping - Implemented all main functionalities - System testing - Documentation	28 Jun
Notifications *Main Functionalities: Receive Notifs (from Friends/Chats/Groups/Channels/Badges)* - Deploy Cloud Functions to listen for activities requiring notifications - Push notification data to data store + render notifications - Enable clicking on notifications to link to relevant screen - Set up UI + Notifications page	Week 7 (28 Jun-4 Jul)
Badge Tier System	Week 8

Main Functionalities: View Badges, Receive Badges - Deploy Cloud Functions to scan for 10th and 30th percentile upvote scores to attain different badge tiers - Link backend code to award badge and send notification whenever users qualify for any tier - Set up Badges UI	(5-11 Jul)
Recommendations *Main Functionalities: Receive Channel Recommendations* - Devise algorithm to sort channels for recommendations - Link backend code to find recommendations for each user - Set up UI for Just For You page	Week 9 (12-18 Jul) Week 10 (19-25 Jul)
Evaluation Milestone 3: Refinement - Additional features (e.g. attach photos, delete account, edits posts/messages etc)	26 Jul
Full Testing Run	Week 11 (26 Jul-1 Aug)
Troubleshooting Backend	Week 12 (2-8 Aug)
Refining UI - Do full UI testing on every screen - Ensure standardisation between screens - Include help screens as mini manuals for guiding new users	Week 13 (9-15 Aug)
Full Testing Run	Week 14 (16-22 Aug)
iOS Store and Play Store Launch	*TBC*
Splashdown	25 Aug

Project Log

S/N	Activity	Date(s)	Ryan (hrs)	Marcus (hrs)	Remarks
1	Learn Figma	3 May	5	-	
2	Poster Design	3-12 May	-	8	
3	JS Refreshers	4 May	3	5	
4	HTML/CSS courses from freecodecamp.org	7 May	3	3	
5	Studying React Native Docs	8 May	3	-	
6	React JS course from freecodecamp.org	9-10 May	7	7	
7	Learning Software engineering Design patterns	12 May	2	-	
8	MongoDB course	17 May	9	-	
9	Firebase Courses on Udemy	18-22 May	-	10	
10	Learning Expo X React Native setup	20 May	3	5	
11	React Native Workshops	15, 22 May	6	3	Each workshop: 1h prep, 2h actual session
12	App UI Design	23 May	4	4	
13	UI: Login, Register, Forgot Password, Navigations, Settings	24-26 May	14	-	
14	Backend: Firebase Integration and Setup	25 May	-	5	
15	Use Cases + Data Model	26 May	8	8	
16	Backend: Authentication	26-27 May	-	15	
17	UI: Profile, Chat Screens, Navigation	28 May	8	-	
18	Group meeting for chat implementation	30 May	1	1	
	Milestone 1 Total Hours:		**76**	**74**	
1	Study Firestore	31 May	3	-	
2	Study Redux	1 June	4	4	
3	UI + Backend: Profile Screen	2-3 June	13	-	
4	Build Firestore Architecture + Data Structure	2 June	-	5	
5	Integrating Redux	2 June	5	7	
6	Backend: Implement Gifted Chat for Chat room	3-5 June	-	14	
7	Update Navigation	4 June	3	3	

8	UI: Chats	4,8 June	10	-	
9	Backend: Chat Room Settings + Create Chat page	6-8 June	-	18	
10	UI + Backend: Friend Ecosystem	7-9 June	15	1	
11	Debugging	9 June	4	3	
12	UI + Backend: Groups	9-11 June	-	8	85% of steps similar to Chats
13	Channels: backend + UI	10-11,13 June	15	1	
14	Debugging	12 June	7	-	
15	Planning for badge tiers + UI: Badges	13 June	-	7	
16	Documentation and doing README	14-25 June	2	23	
17	Styling fixes, alert styling fixes	14 June, 16 June	3	2	
18	Channels: Posts, Comments, upvotes	15-16 June	10	-	
19	Improvements for Searching (case insensitive etc)	16 June	2	-	
20	Firebase Cloud functions	17 June	5	6	
21	Implementing Badge Functions and Algorithm	18-22 June	-	10	
22	Home Notification system: backend cloud + frontend	17 - 18 June	16	-	
23	Bug fixes	18 June	2	-	
24	Overhauled rendering system for entire app	19 June	3	-	
25	Fixed async issues for entire app (mainly badges)	20, 22 June	11	3	
26	Loading indicators for the entire app	23 June	4	-	
26	App UI Review + Edits	23 June	-	5	
28	Help Screens	23 June	1	3	
29	Prepare app for milestone 2 (Special Channel, notification etc)	24 June	1	-	
30	Enhancements to rooms: View user profiles	24 June	2	-	
31	Videography + Script for Video Submission	25-27 June	2	5	
	Milestone 2 Total Hours:		**143**	**128**	
1	Pin/edit posts	29 June	4	-	
2	Add Loading for screens, fix Friends render	1 July	4	-	

3	Review + Addressing bugs from Peer Evaluation Milestone 2	2-3 July	9	8	
4	Designing User Testing Form and Protocol	3-4 July	-	5	
5	Image Support for Chats and groups	2-4, 6 July	10	1	
6	Live Search + Search history	5 July	5	-	
7	Styling Review and standardisation across all screens	5 July	-	8	
8	Settings Drawer	6-7 July	6	1	
9	User testing	4-8 July	7	7	
10	Addressing bugs from User Testing	9 July	-	6	
11	Profile Picture support for channels	9 July	3	-	
12	Emulator suite testing	9 July	3	-	
13	Learning Jest and React Native Testing Library	10-14 July	-	5	
14	Writing Automated Component Tests	18-22 July	-	14	
15	Numerous bug fixes and improvements	20-22 July	10	6	
16	Delete Account feature	23 July	3	-	
17	M3 README	23-25 July	3	7	
18	Prepare app for M3	24 July	2	2	
	Milestone 3 Total Hours:		**69**	**70**	
	Total Hours:		**288**	**272**	