



WE BELIEVE

README(MS3)

Tan Tze Young

Seah Zi Xiang

Table of Contents

Project Overview.....	4
Motivation.....	4
Aim.....	4
User Stories.....	4
Scope of Project.....	6
Features.....	6
Login and Registration System.....	6
Resetting of Password.....	9
Posting of stock ideas.....	11
Viewing of new blog posts //TODO Update the images.....	11
Posting of new blogs.....	12
Editing of posts.....	13
Upvote of Posts.....	13
Comments of Post.....	14
Searching for fellow users.....	17
Following Other Users.....	18
Unfollowing Other Users.....	19
View Profile.....	19
Privacy of stocks.....	20
Ranking of users.....	22
Updating of Profile Picture.....	23
Realtime updates of market and stocks prices.....	25
Data and Logic Flow.....	29
Authentication.....	30
Home View.....	31
OOP Principles.....	32
Information Hiding.....	32
Code Reusability.....	32
Garbage Collection.....	33
Error Handling.....	33
Software Architecture.....	35
Client-Server architecture.....	35
Event Driven architecture.....	35
N tier architecture.....	36
Software Design Patterns.....	36
Singleton Pattern.....	36
Model View Controller Pattern (MVC).....	37
Version Control.....	37
Branching.....	37
Pull requests and issues.....	38
Testings:.....	39
Unit Testing.....	39
Widget Testing.....	42
Integration Testing.....	43
Authentication Tests.....	43
Features Testing.....	44

User Testing.....	45
Tech Stack.....	46
Milestone 1 Overall Current Progress.....	46
Milestone 2 Overall Current Progress.....	46
Milestone 3 Overall Current Progress.....	47
Timeline and Development Plan.....	47
Current Testing Limitations.....	51
Milestone 3 Links.....	51
Class Diagram.....	51
Work Log.....	51
Poster.....	52
Video Demonstration.....	52
APK.....	52
Milestone 2 Link.....	52
Milestone 1 Link.....	52
Acknowledgement.....	53
External Packages Used.....	53
Sources referred.....	53

Milestone 3 Submission

Project Overview

Motivation

As a value investor investing in the Singapore exchange market, there is a lack of a place where one can find good stock analysis. This sentiment is echoed throughout, with many pointing out how stock analysts are usually compromised by investment banking relationships and institutional groupthink. Retail investor analysis of stocks can be found on blogs online, but these are hard to find and hard to keep track of. Hence, we saw a need for a platform where investors can pitch and consolidate their investment ideas.

Besides that, specifically in the Singapore market, consolidating company information and getting company announcements can be quite a hassle. Furthermore, stock valuation metrics, such as the price over earnings of a stock have to be manually calculated and are not updated in real time, since the price of the stock can fluctuate, resulting in valuation metrics having to be updated each time.

Aim

Thus, we aim to create an app that serves not only as a place for top investors to share their investing ideas but also to help value investors in their investing process. In the app, users will have their own profile and space to write their investment ideas. Users can choose to “follow” other users to get updates on their investment ideas. The app will also get real-time information from the Singapore exchange website, and update any custom valuation metrics a user calculates.

User Stories

1. As a user, I would like to know what are the thoughts of other users on a certain stock so that I can critique and discuss with like-minded people their thoughts on a certain stock
2. As a user, I would also like to see credible and well-researched blogs being recommended to me on the platform so that I would be further incentivized to use this platform
3. As a user, I would like to see the performance of other people’s portfolios on the platform so that I can gauge how good of an investor they are.
4. As a user, I would want the app to be secure so that I can ensure that my data is kept private and confidential as per my settings dictate.
5. As a user, I would want a platform that is easy to use and navigate so that I can spend more time thinking about stock ideas and what to post instead.
6. As a user, I would like timely updates on stock information and other information so that I can always work with the latest information.

7. As a user, I would like a pleasant-looking application so that reading others' blogs is easier.
8. As a user, I would like for there to be a way to search for other specific users so that I can easily find blogs other people recommend to me.

Scope of Project

We Believe is a mobile application that serves as a platform for various investors with all sorts of experiences to come and share their ideas. They can give their thoughts and ideas on the various investment plans.

The features of our application will be detailed below. For each feature, we have split it into three further parts of

1. **[Proposed]** -> **Features for Minimum Viable Product(MVP) by splashdown**
2. **[Current Progress]** -> **Elaboration on the current status of specific feature**
3. **[Additional Features]** -> **Possible add-ons**

Features

Login and Registration System

[Proposed]: Our application needs a standard login and registration system to store the data of the various users. This login and registration system will take in the corresponding email and password.

During the registration process, there will also be an input field for a name such that

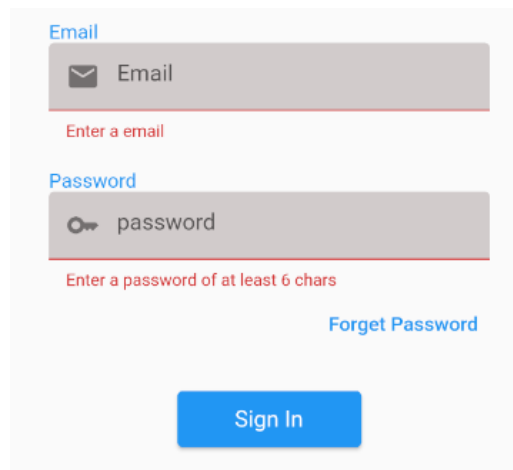
[Current Progress]: Our application needed a standard login and registration system to keep track of the various users and their corresponding data.

The image displays two side-by-side screenshots of a mobile application's login and registration screens. The left screen, titled 'Login', has a blue header bar with the word 'Login' and a 'Register' link. Below the header, there are two input fields: 'Email' with an envelope icon and 'Password' with a key icon. A 'Forgot Password' link is positioned below the password field. At the bottom, there is a blue 'Sign In' button. The right screen, titled 'Registration', has a blue header bar with the word 'Registration' and a 'Log In' link. It features four input fields: 'Email' (envelope icon), 'Name' (person icon), 'Password' (key icon), and 'Re-Enter Password' (key icon). A blue 'Register' button is located at the bottom of the form.

Screenshot of the current design of the app. The left shows the login screen and the right shows the registration screen.

The 2 screens also have a form of validator to ensure that the emails and password meet a certain criteria such as:

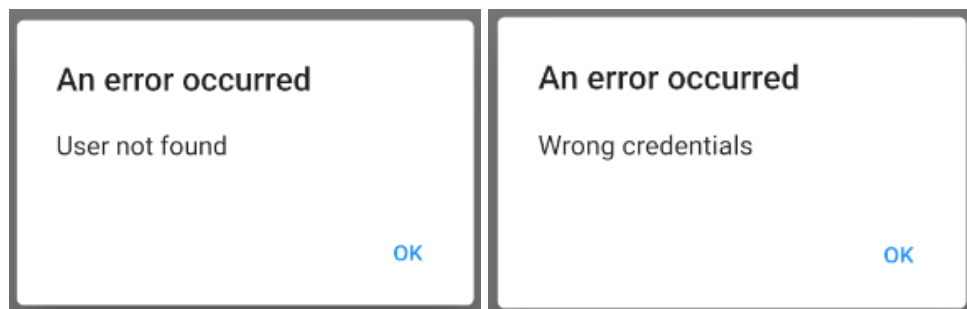
1. Email field cannot be empty
2. Passwords must at least have 6 characters



The screenshot shows a login form with two input fields. The 'Email' field is labeled 'Email' and has a red error message 'Enter a email' below it. The 'Password' field is labeled 'password' and has a red error message 'Enter a password of at least 6 chars' below it. A blue 'Sign In' button is at the bottom. A link 'Forget Password' is also present.

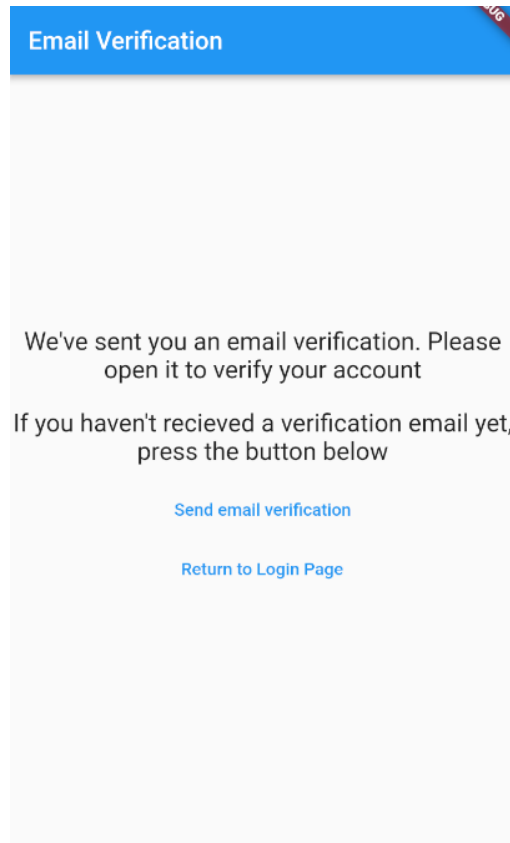
Screenshot of the validator in action

After clicking the Sign In/Register button, a loading screen will show up and should there be any further inconsistency or errors with the input fields, an error dialog box will pop up and inform the user of the possible error.



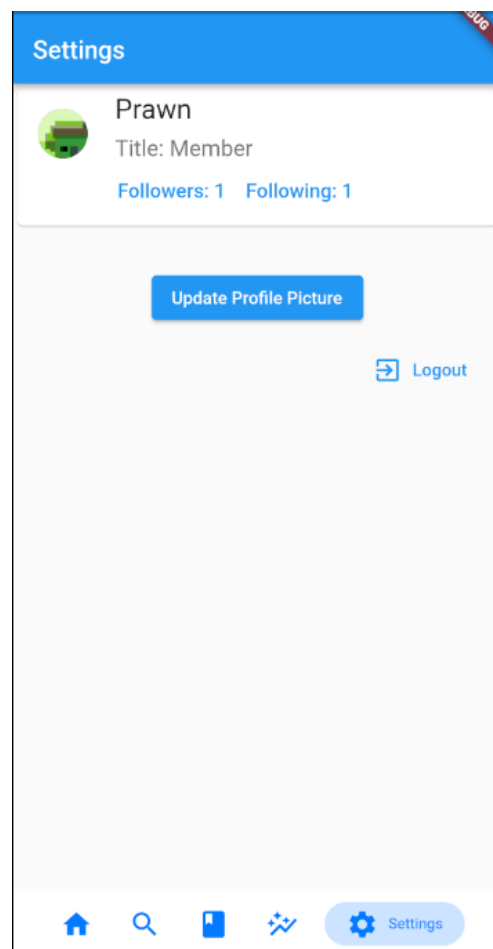
Screenshot of possible error dialog boxes

As for the registration portion, if all goes well, an email verification will be sent to the user's email and they need to verify their email in order to proceed.

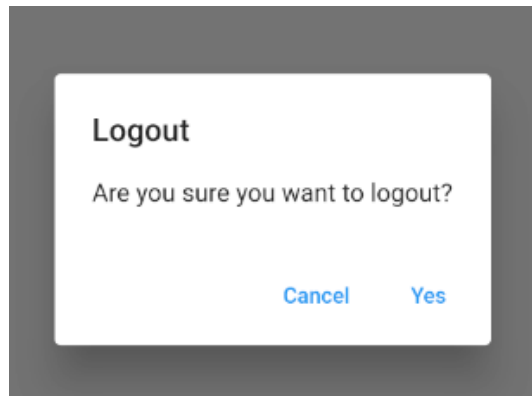


Screenshot of the email verification

Once in the app, users can switch between the different views and access the settings view and log out from there.



Screenshot of the Settings screen

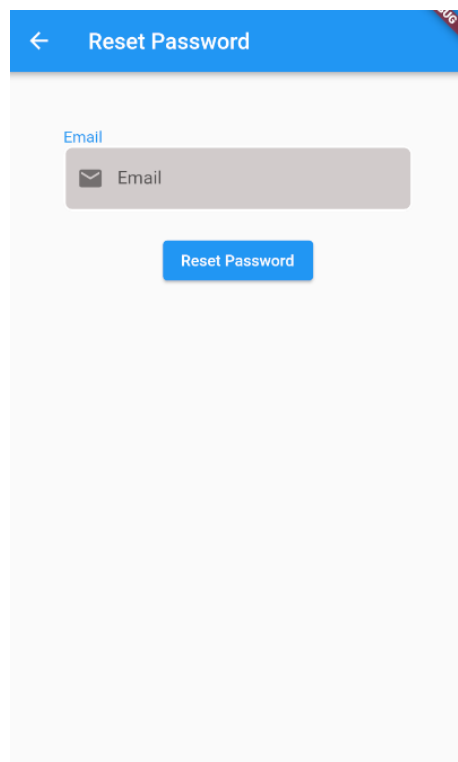


Screenshot of dialog before logout

Users will be asked whether they would like to logout when they click on the logout button.

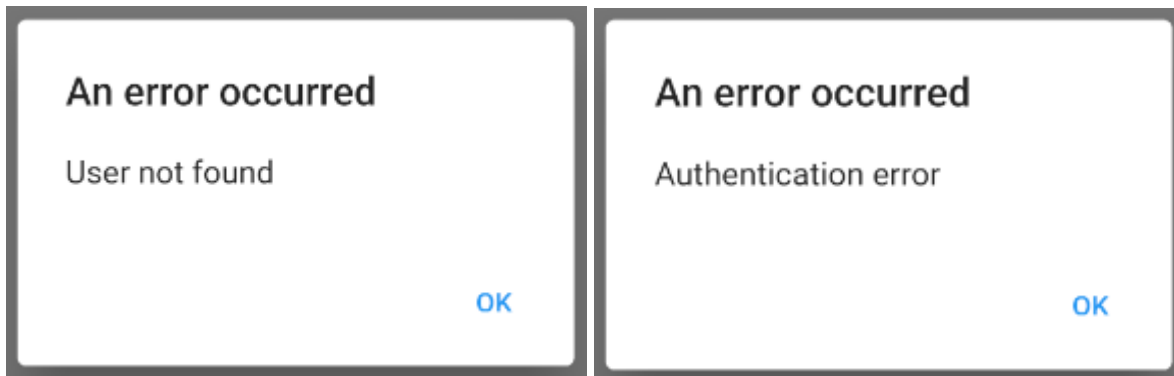
Resetting of Password

Users are also able to reset their passwords should they forget it. By clicking on the forget password text in the login page, they will then be brought to another page.



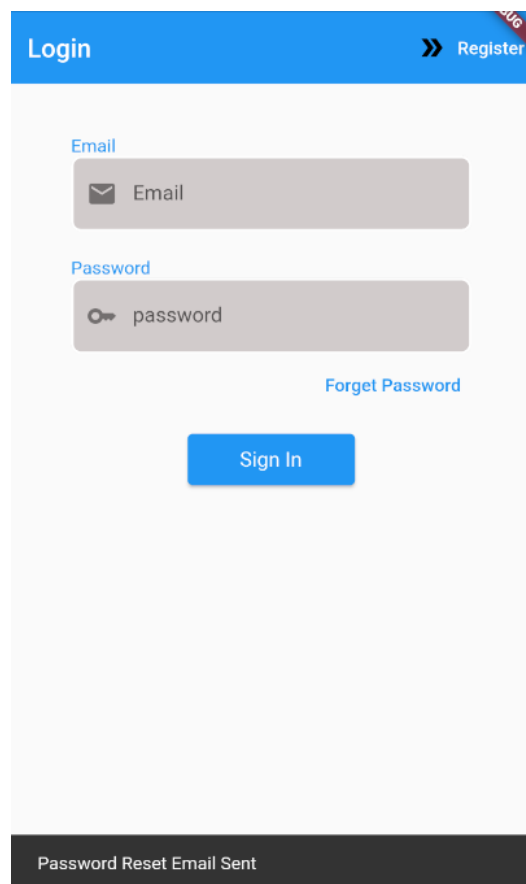
Screenshot of the reset password page

Users would then type their emails into the text field and click on the reset password button. Should they enter in an invalid email or a user that does not exist, the system will show them an error message.



Screenshot of the potential errors

They have entered a valid email, they will be brought back to the login page and a snack bar will appear at the bottom to let them know that a reset password email has been sent to them.



Screenshot with the snack bar

Users would then need to go to their emails and click on the link within the email sent to them. They can then follow the steps on screen to reset their password. Once done, users would see a screen like the one below.

Password changed

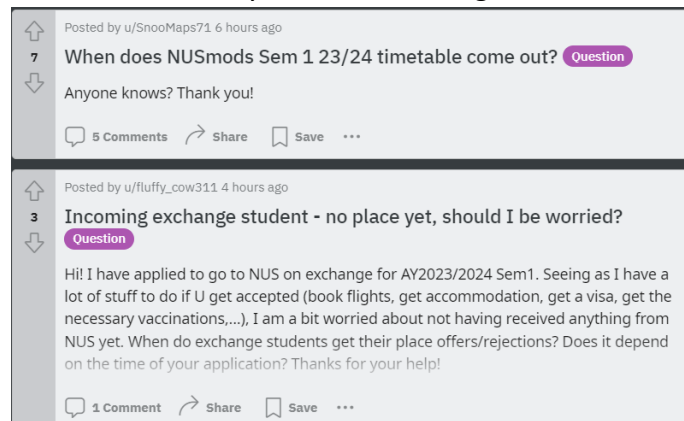
You can now sign in with your new password

Now, head back to the login page and login as per normal with your new password.

[Additional Features]: As now we only can log in with their email, we can still further add on the feature of them being able to login with an alias like using a username and unique ID/tag.

Posting of stock ideas

[Proposed]: Our application would have a system where users can freely post and share their thoughts with other fellow users. These posts can be about what stock ideas they have, whether certain stock investments may be worth looking into and so on.



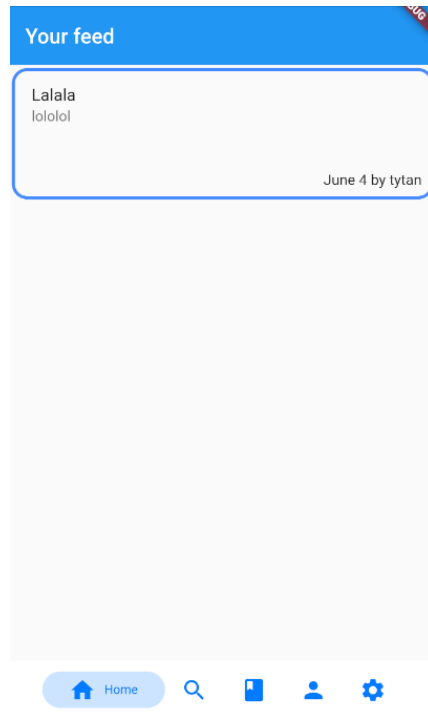
Screenshot of a section on Reddit, a popular discussion website and app.

Our proposed idea would be similar to that of the image above, where any user can come and view the comment or post that the user has made.

[Current Progress]:

Viewing of new blog posts //TODO Update the images

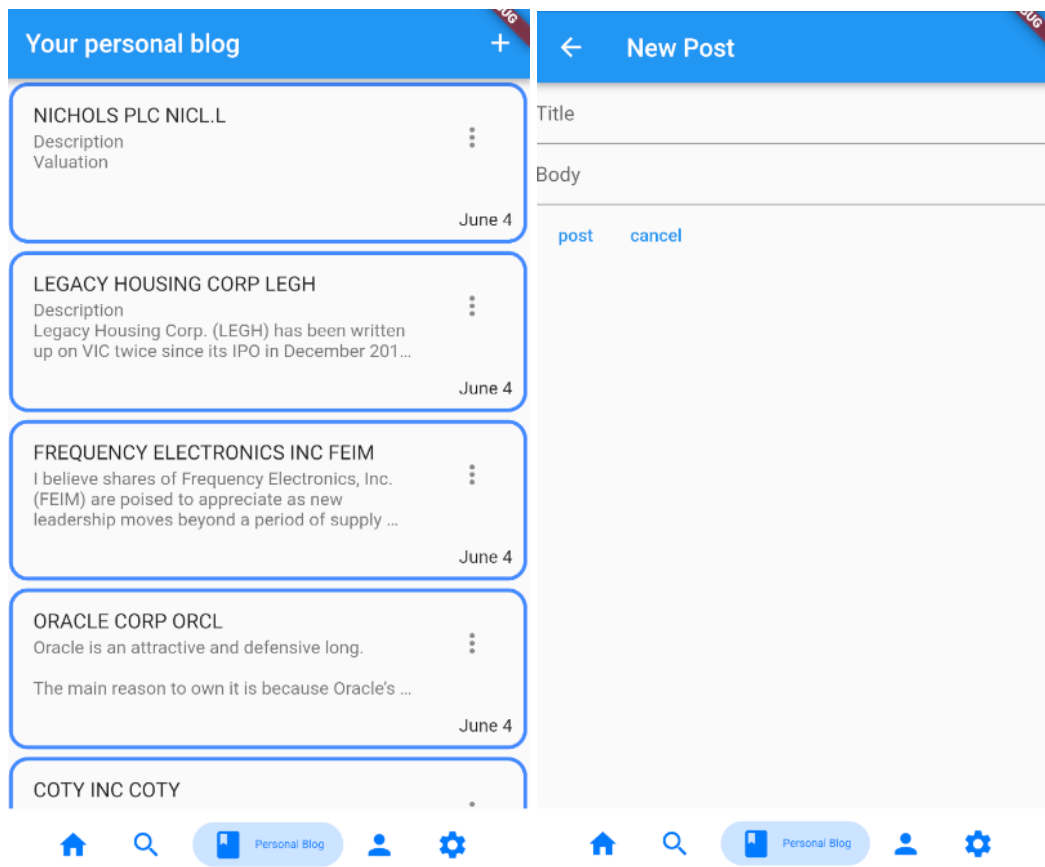
Our application's home page will display the latest few blog posts from your various followers, along with the date that it was posted. These posts will be updated in real-time.



Screenshot of the application's home page.

Posting of new blogs

The centre tab of our bottom navigation bar will be the screen for personal blog, which is also the screen that will allow users to post their new blog posts.



Screenshot on the left is the current personal blog screen while the right shows the new post screen

Users will also be able to view their own blogs here. The blogs are ordered based on the date they were posted, with the latest blog being at the top.

Editing of posts

Users are able to delete or edit their posts as they see fit as well.



Screenshot of the edit screen

Upvote of Posts

Once users have followed other users and are able to see their posts, they would be able to upvote the posts that they like.



By clicking on the arrow at the bottom left of each post, users would then be able to upvote the corresponding post. Once upvoted, the counter will update in real time and the arrow will turn red to signify that it has been upvoted.



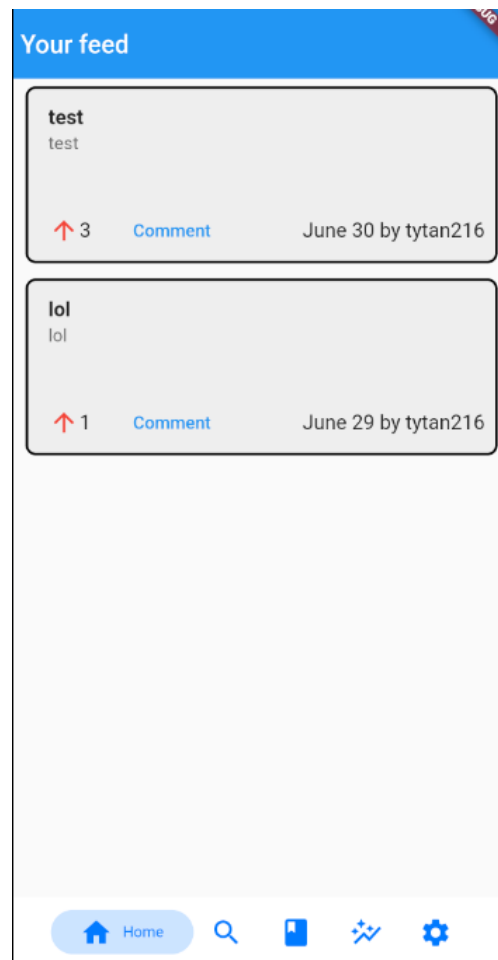
[Additional Features]: We could have a moderator or a form of validator to ensure that the posts are within limits and do not contain any form of misinformation or inappropriate language

Another possible additional feature is the ability to critique the posts itself. This would allow for users to give directly feedback to other users and raise their points or concerns

Comments of Post

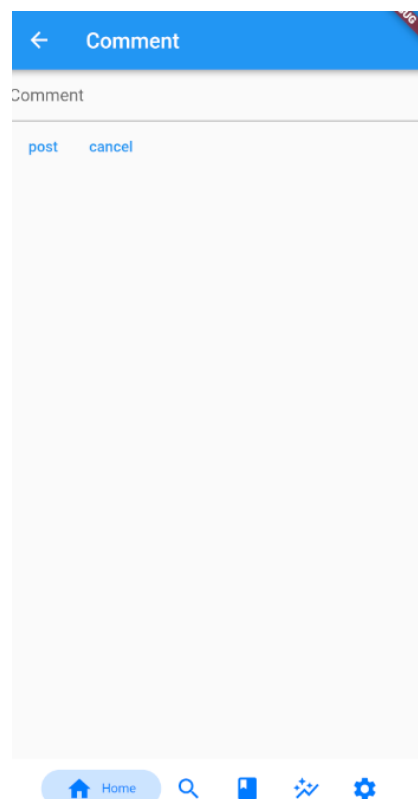
[Proposed]: As this is supposed to be similar to that of a blog, we believe that users should have a way to leave their thoughts and ideas on the various posts of other users. The way would be that when users view the post, they can then access and view the comments of that specific post.

[Current Progress]: Right now, users are able to comment on the posts of anyone through the comment button that is on the same row as the upvote and datetime.



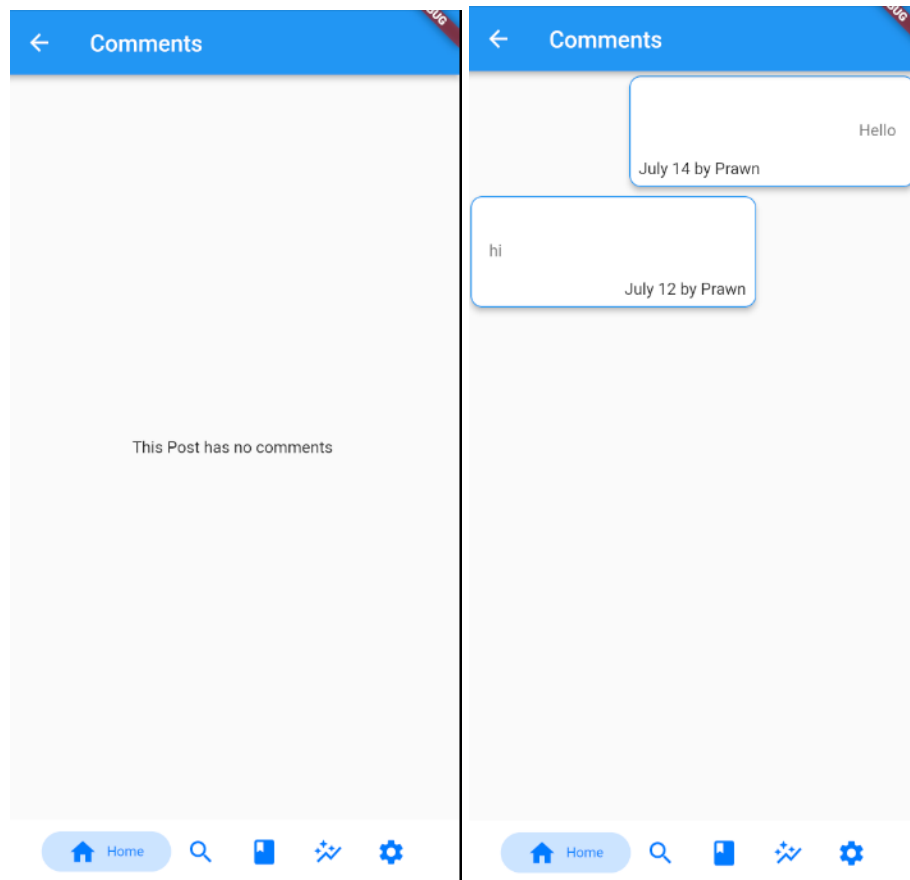
Screenshot of the main screen, with the comment button

When users click on the comment button, they will be brought to a page similar to the create post page. Currently, there is a character limit of 100 characters on the comments as we felt that it is supposed to be a comment and users need not type very long essays



Screenshot of the post comment page

To view all the comments of a post, users need to tap on the post, after which scroll down to the bottom till they see the comment section button and tap on it.



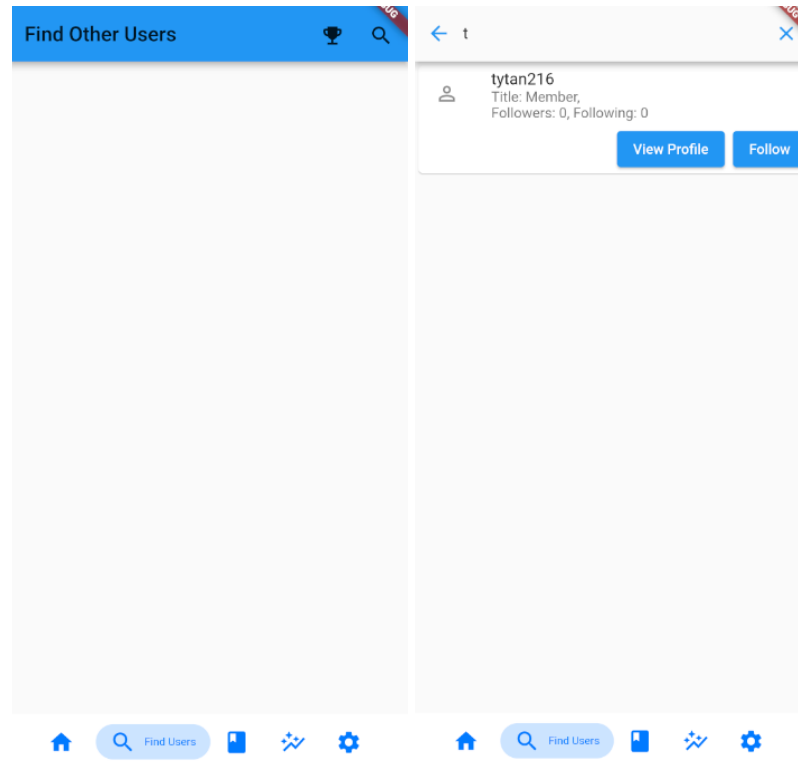
Screenshot of the comments section

[Additional Features]: --

Searching for fellow users

[Proposed]: In order for users to follow other users, they need to be able to find them. As such, we would like a feature where users are able to look for other users using their account names.

[Current Progress]: A screen has been made to house this feature.



Screenshots of the find users screen. The right shows one sample of the search in action

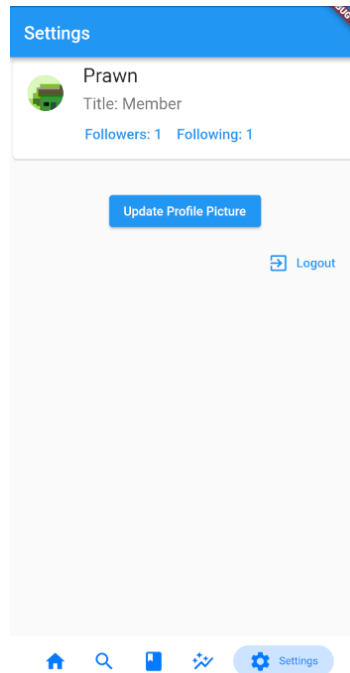
When searching for a user, current users need to search for their name or a part of it for it to appear in the body of the screen. All names that contain the value entered into the input field will be shown. The list updates in real time and if nothing matches, nothing will be shown.

Following Other Users

As users are searching for other users, they will also be given the option to follow other users and even view their profile (*Elaborated in the next feature*).

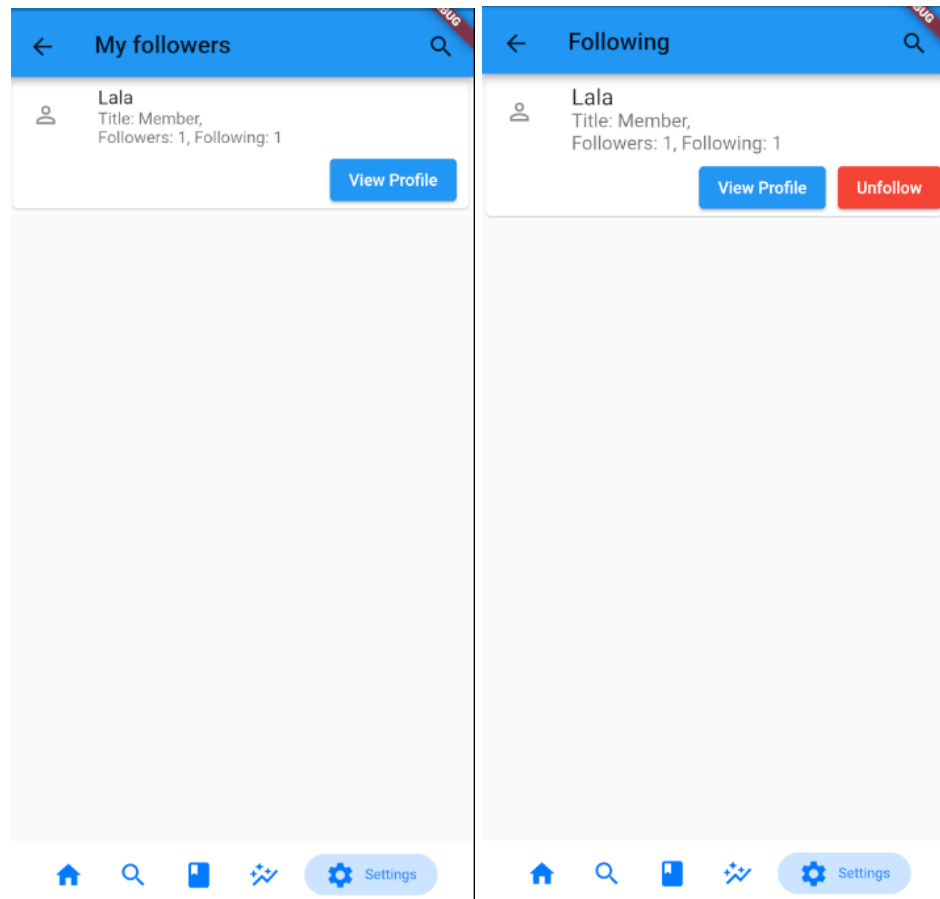
When users follow other users, their following count will increase by 1 and that user will no longer be displayed or shown within the find users screen. The person you followed will also have their followers increase by 1.

To access my followers and following page, head over to the settings tab to view it.



Screenshot of the settings screen

Tap on the followers word to display the my followers screen and the same thing for following.



Screenshot of the followers and following pages

Unfollowing Other Users

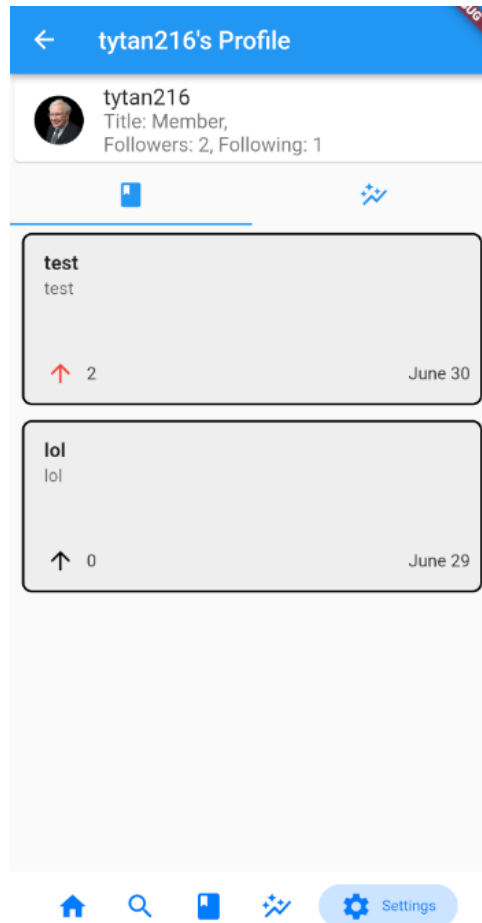
If users no longer want to follow someone, they have the option to unfollow them in the my followers page. To refollow them, users need to go back to the find users screen to do so.

[Additional Features]: This is linked to the other feature of ranking but the users could be shown based on the ranking system.

View Profile

[Proposed]: We would like for users to be able to see the blogs of the other users and possibly their stocks from a more centralised location, allowing them to make a more informed decision as to whether or not to follow.

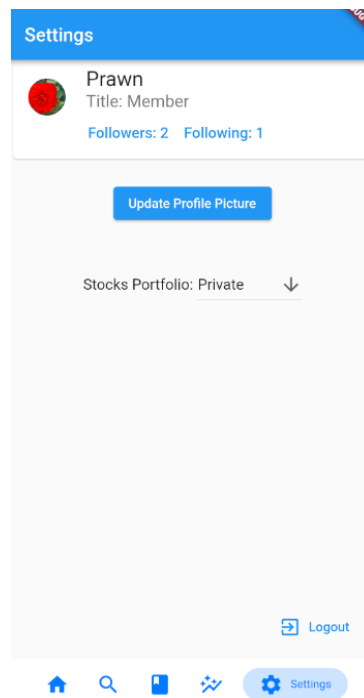
[Current Progress]: This feature can actually be accessed from multiple locations such as find users, followers and following. The view profile screen will look like the image below,



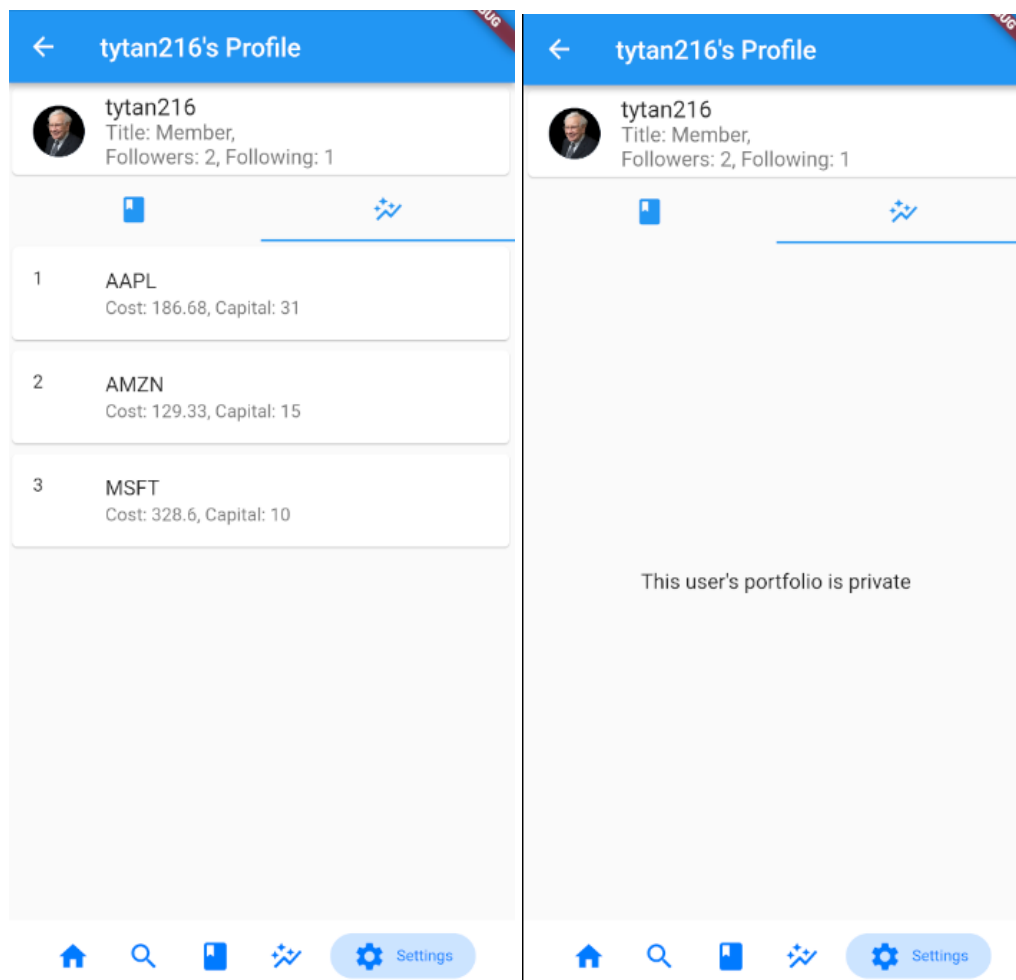
Users will be able to see the standard profile information at the top, followed by two tabs next, the left tab being the posts while the right is on stocks.

Privacy of stocks

For stocks, as there is some personal information involved, we have decided to allow users to have the option of setting their stock profiles to private so that others may not view them. To change this setting, users can head over to their profile page.



Here, users can choose between 3 options, which are private, friends only and public. When changed, this will update the privacy setting in real time. The following are how it would be when it is private and when it's public.



[Additional Features]: --

Ranking of users

[Proposed]: We would like for our app to also be able to rank the more popular users or the users whose posts have been viewed or liked the most. This would allow for new users to know who are the ones who may have slightly better insight and can give them better advice when it comes to stocks. This ranking system may be developed in a myriad of ways such as follower count, or even by the most total upvotes across their posts.

1	155	174
Posts	Followers	Following

Screenshot of Instagram, a popular social media platform

One possible implementation of a follower count system would be similar to that of Instagram, which would display the number of followers a user has in their profile.

I-a. Some functions

7 As these will be used in the continued fraction evaluations below, recall the *Riemann zeta function* $\zeta(s)$, and *Dirichlet beta function* $\beta(s)$,

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s}$$
$$\beta(s) = \sum_{n=1}^{\infty} \frac{(-1)^{n-1}}{(2n-1)^s}$$

and special cases of the *Clausen function* $\text{Cl}_s(x)$,

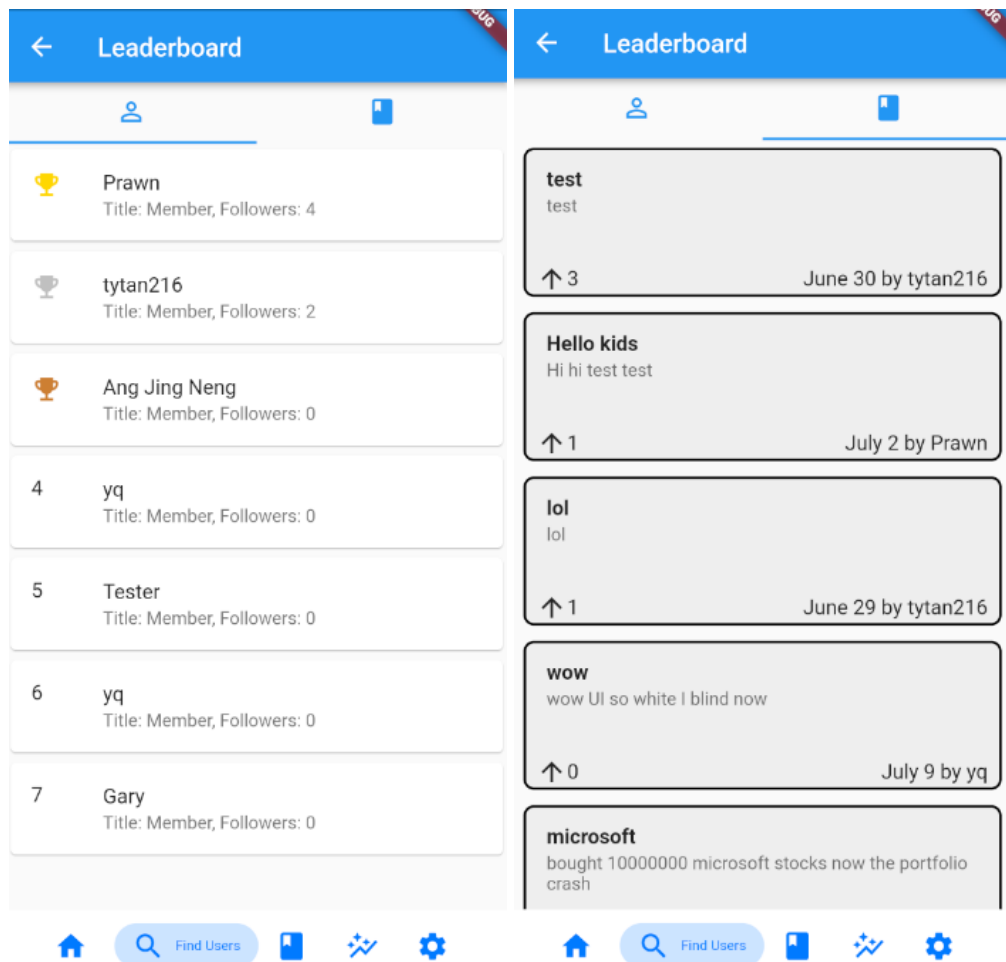
$$\text{Cl}_2(x) = \sum_{n=1}^{\infty} \frac{\sin(nx)}{n^2}$$
$$\text{Cl}_2\left(\frac{1}{2}\pi\right) = K = \beta(2)$$
$$\text{Cl}_2\left(\frac{1}{3}\pi\right) = \kappa$$

with *Catalan's constant* K and its cubic counterpart *Gieseking's constant* κ .

Screenshot of a post on Mathoverflow, a website where users can share their questions and answers about Math related topics.

Another possible implementation would be to use the total upvotes of all posts. Like the image above, the number of upvotes would be displayed for each post, summed together, and then ranked.

[Current Progress]: We are using the follower count as the basis for our ranking. The more followers that they have, the higher up they will be on the leaderboard. This leaderboard is accessible from within the find users screen, by clicking on the trophy icon at the top right of the screen. The leaderboard will have 2 tabs, the first will be the ranking of the actual users based on their follower count and the second is the ranking of posts, which is ranked by the number of upvotes of each post.



Screenshot of the leaderboard screen

[Additional Features]: With the ranking system in place, we could implement a filter feature that would display users with higher ranks first so that their posts are more easily accessible to other users.

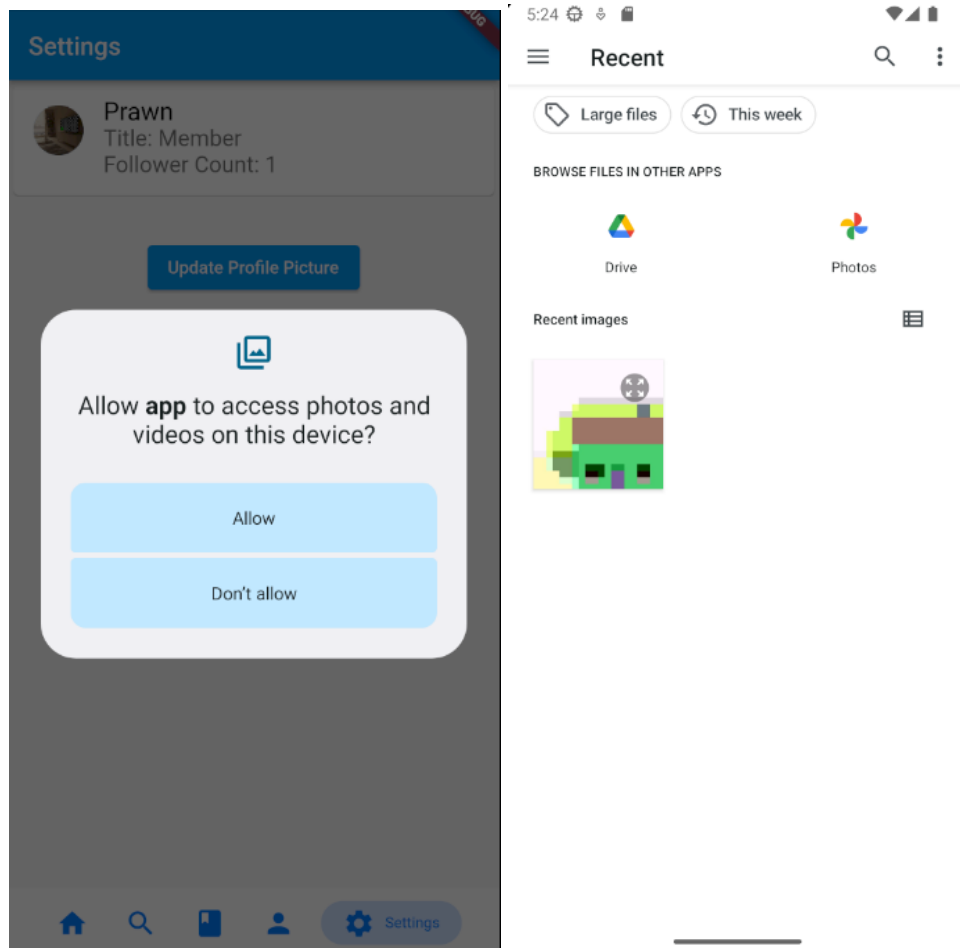
Currently, the posts will rank from highest to lowest. Maybe a possible filter is to change the way the posts or leaderboard is organised.

Another feature that could potentially be added is some form of incentive or privilege that comes with a certain rank or following.

Updating of Profile Picture

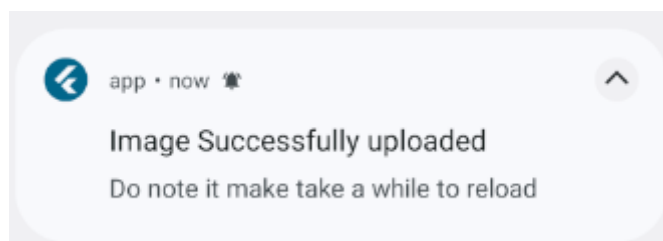
[Proposed]: Just like most other social media platforms, we would like for the users to be able to also upload and attach a profile picture of themselves to their overall profile. Users will start off with no image by default and can change it whenever they want to. This image will then be updated in real-time.

[Current Progress]: There is currently a button under the Settings screen to update the profile picture. When pressed for the first time, users will be prompted to allow the application to access their photo gallery.



Screenshot of permission dialog and choosing of image

After which, should notifications be enabled, users will receive a notification stating that their image has successfully been uploaded and may take a while for the new profile picture to appear



Screenshot of the local notification

You may also click on the various profile pictures in the various screens to view the full image of it.



Screenshot of the full image view

Just click on the spaces around the image to go back to the page you were just on.

[Additional Features]: Can possibly add gifs as profile pictures.

Realtime updates of market and stocks prices

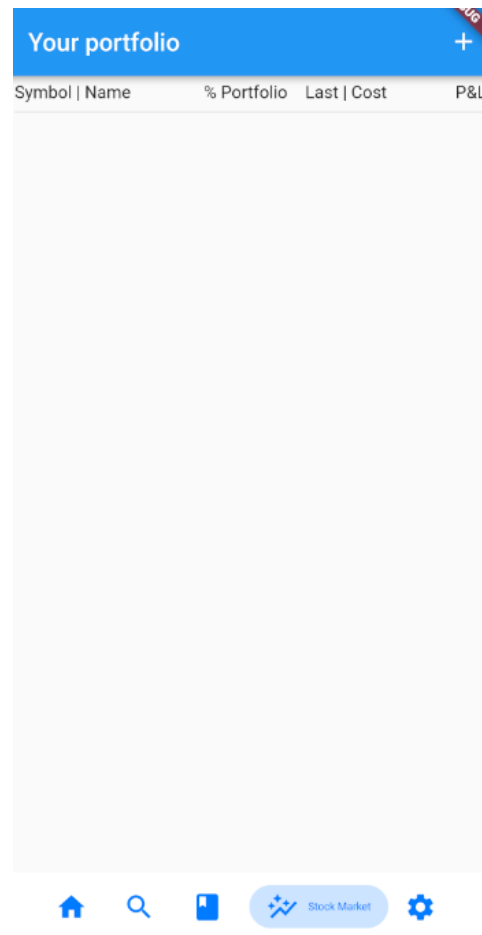
[Proposed]: We would like for the users to also be able to look up the current state of the market in real-time without having to use another app or website. The data will then be displayed on the screen.

iEdge-UOB APAC Yield Focus Green REIT Index (SGD)	▼	2,179.35	▼
FTSE ST Consumer Goods & Services Index	▼	253.80	▼
iEdge SG All Healthcare Index	▼	1,981.36	▼
iEdge S-REIT Leaders Index SGD	▲	1,164.16	▼
iEdge SG ESG Leaders Index	▲	987.74	▼
iEdge-CNBC China Growth Economy Index	▼	1,262.49	▼

Screenshot of a portion of the SGX website, detailing the current market rates and whether they are on the rise or falling

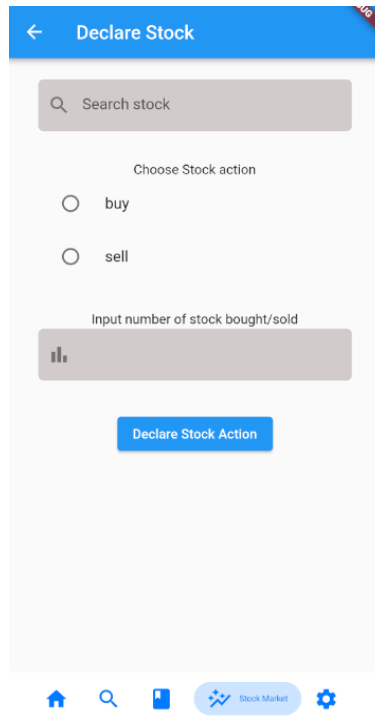
This feature would be implemented using a form of API such as restful API to obtain the data in real time and display it to the users.

[Current Progress]: We have decided to use the alpha vantage API to help us track real time data, after which we will store the data at that moment in firestore.



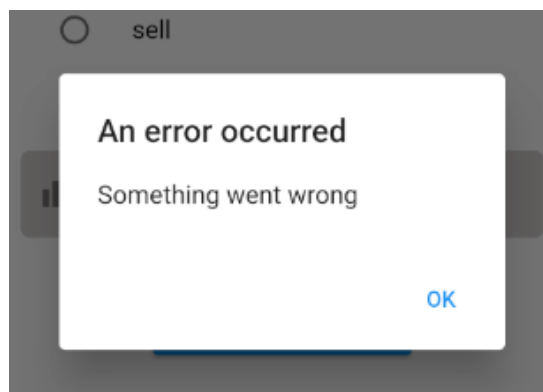
Screenshot of the portfolio page

Same as with the blog, clicking the plus button at the top will bring the user to a new screen where they can either declare or sell the various stocks and how many.

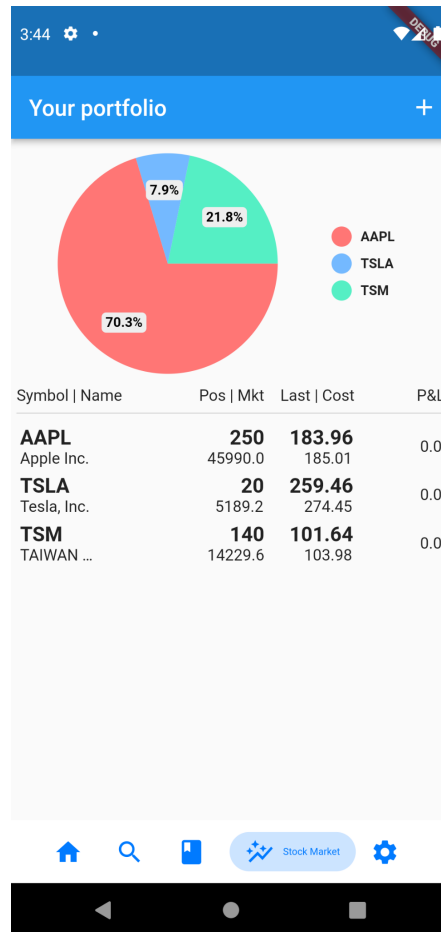


Screenshot of the declaration screen.

If users try to declare a stock that does not exist, an error dialog will be thrown.



If there are no errors and that the firestore database gets updated successfully, then the stock will be displayed back under portfolio



Screenshot of the addition of newly declared stock

Once stocks have been declared, a pie chart will also be displayed to show you the proportion of your various stocks.

Users can also see their transactional history of each stock by clicking on it. This tab will then display what the user has done, be it buy or sell, along with the price and date of transaction

The screenshot displays a mobile application interface for a stock transaction history. At the top, a blue header bar contains a back arrow and the text 'Stock history - MSFT'. Below the header is a table with the following columns: Direction, Quantity, Price, and Date. The table shows a single transaction: a 'BUY' (in green) for a quantity of 3 at a price of 340.54 on the date 7/2/2023. The bottom of the screen features a navigation bar with icons for home, search, a document, a 'Stock Market' button, and settings.

Direction	Quantity	Price	Date
BUY	3	340.54	7/2/2023

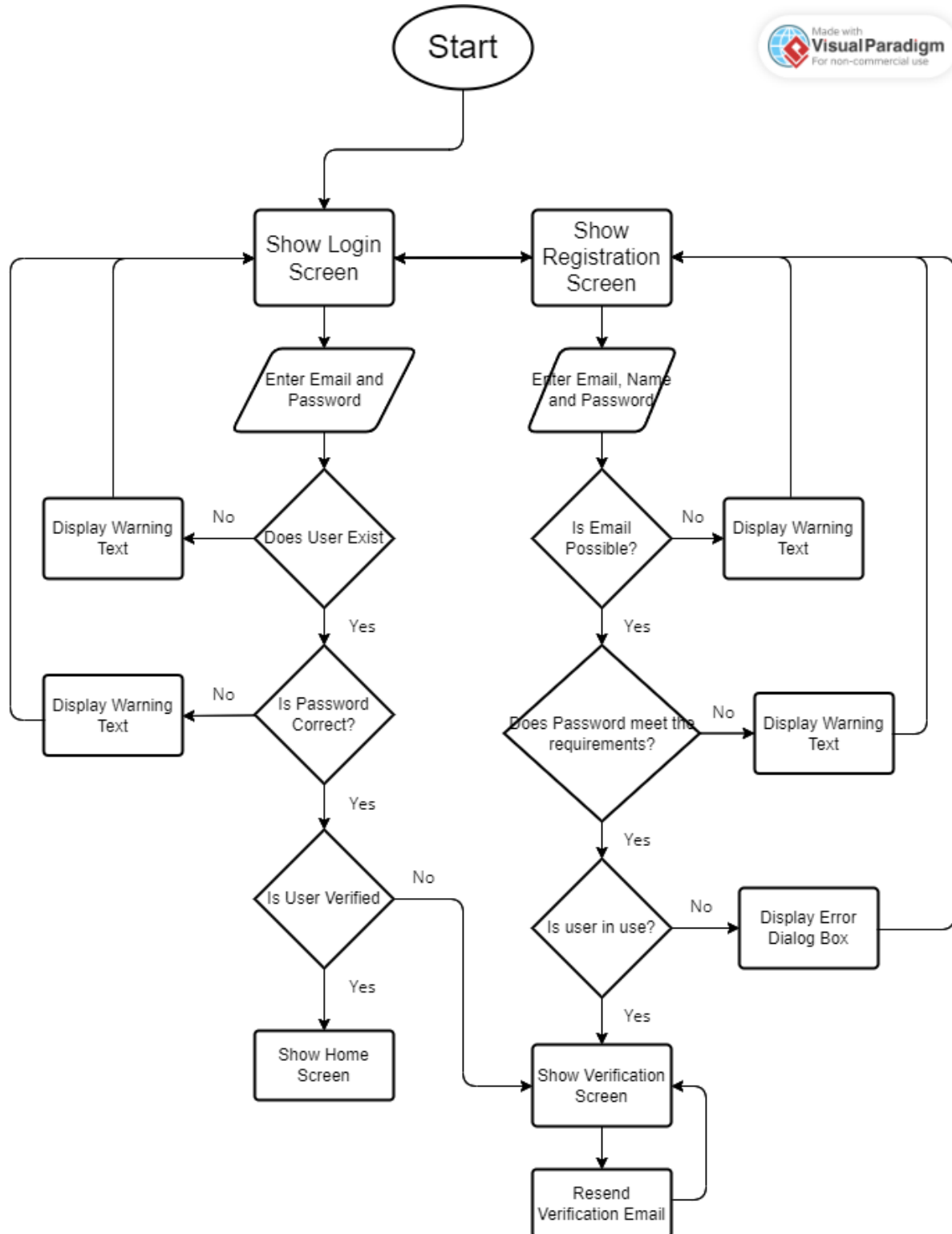
Screenshot of stock history page

[Additional Features]: We could have a setting that would allow users to set only the stocks or markets they want to view, so a feature such as a filter could be implemented.

Data and Logic Flow

Authentication

For authentication, users will be able to toggle between login and registration screens. They will then input the data into the various fields such as email and password. Then, the system will check their input and return the corresponding output



Home View

Here is a diagram of the various interaction of the screens once the user has logged in

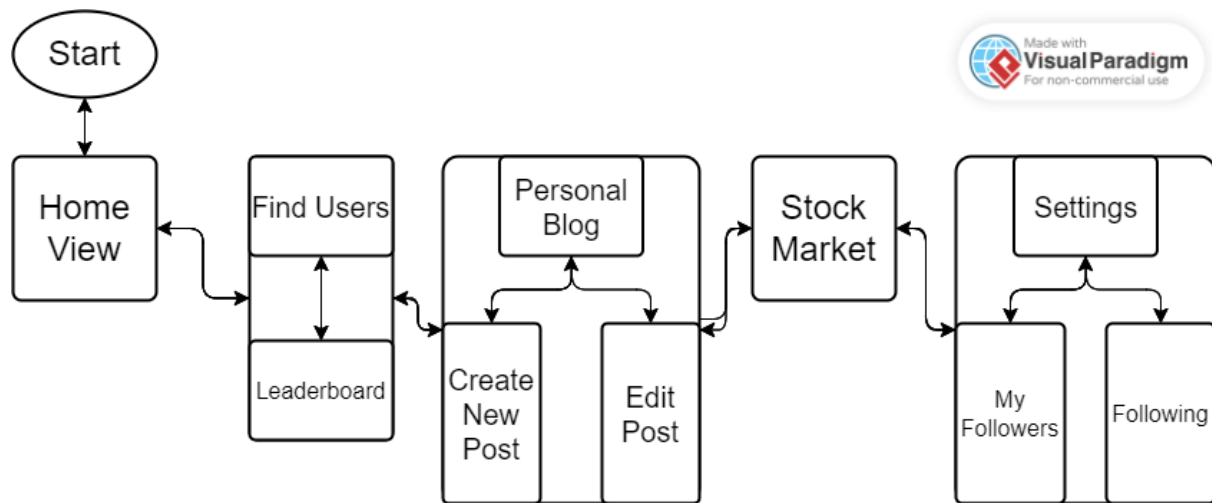


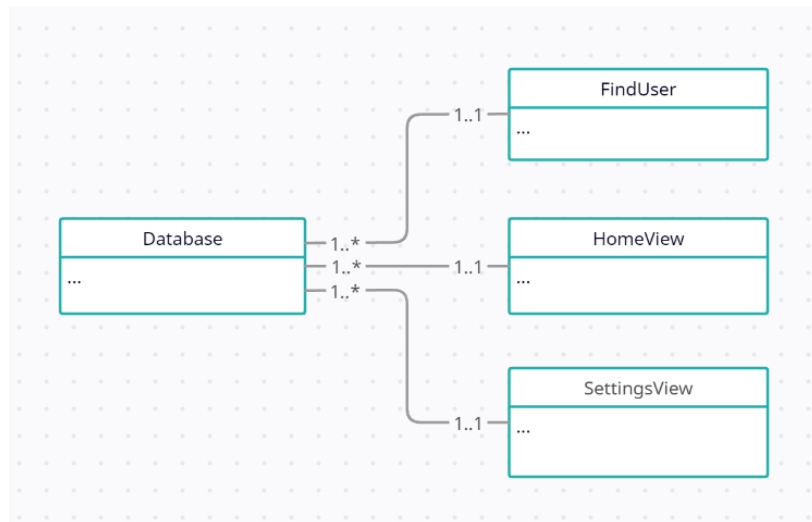
Diagram of the screen interaction

As can be inferred from the diagram above, users can easily manoeuvre from one screen to another quite easily by clicking on the corresponding icons on the bottom navigation bar. There are also other screens within one main screen and users can also go between them, all while the option to go to another main screen still exists.

OOP Principles

Information Hiding

One way we achieved this was to move all the functions that related to firebase as well as firestore into a different class. By doing so, we will ensure that the functions cannot be altered outside of its abstraction and further implementations just need to know its input and output.



Partial Diagram of the Association of some classes

This diagram shows that although there are quite a few classes that rely on the Database class, they do not need to know the functionalities of it. All they need are the functions that will return their desired output given an input.

Code Reusability

There are quite a few instances where we abstracted out certain information to cut down on the amount of code that has to actually be written.

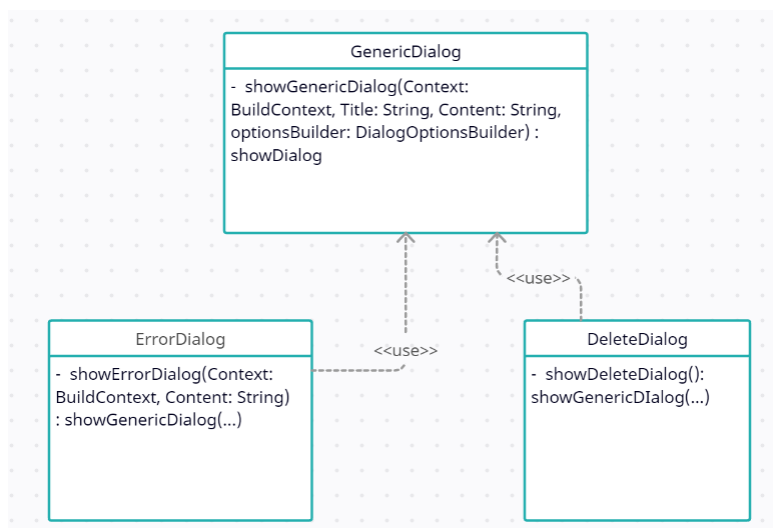


Diagram representing the dependency of various dialog classes

In the diagram above, we can see that the `ErrorDialog` and `DeleteDialog` both make use of the `GenericDialog` class. This ensures code reusability and further extensibility in the future should there be more Dialogs to be made.

Garbage Collection

We also used dispose functions to remove objects such as text controllers and other objects in stateful widgets when they are not in use. This allows for better memory and space allocation.

```
@override
void dispose() {
  _textControllertitle.dispose();
  _textControllerbody.dispose();
  super.dispose();
}
```

Screenshot of the dispose function

The above snippet shows the dispose function, where it will dispose of all text controllers once not in use.

Error Handling

Using of customised exceptions allows us to have a better grasp of what exactly the error is rather than just doing a `pokemon(Catch them all)` exception handling

For example, within our authentication service code, there will be points where the given functions could throw various exceptions.

```
} on FirebaseAuthException catch (e) {
  if (e.code == 'wrong-password') {
    throw WrongPasswordAuthException();
  } else if (e.code == 'user-not-found') {
    throw UserNotFoundAuthException();
  } else {
    throw GenericAuthException();
  }
} catch (_) {
  throw GenericAuthException();
}
```

Screenshot of the throw custom exceptions

After the exception gets thrown, it will then lead to different error dialogs to be shown on screen.

```
} on UserNotFoundAuthException catch (e) {  
  await showErrorDialog(  
    context,  
    'User not found',  
  );  
  setState(() {  
    loading = false;  
  });  
} on WrongPasswordAuthException catch (e) {  
  await showErrorDialog(  
    context,  
    'Wrong credentials',  
  );  
  setState(() {  
    loading = false;  
  });  
}
```

Screenshot of showing error dialog

This would then allow users to have a better idea of what error they have made and can make the correction from there.

Software Architecture

Client-Server architecture

Our application implements some form of Client-Server architecture

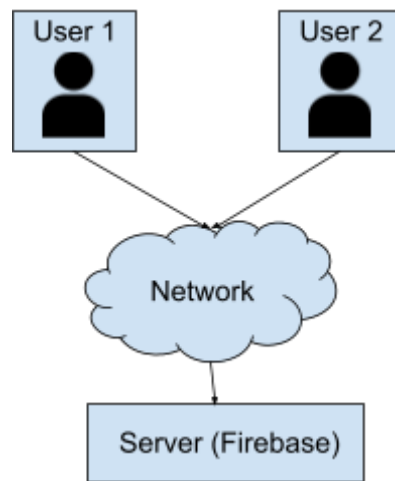


Diagram of the architecture

Whenever the user clicks on a button or uploads a post, they will go through the network and upload it to the firebase servers. Anytime the users need their data, they will also retrieve their data from the server rather than from each other (Not P2P).

Event Driven architecture

Our GUI also implements event driven architecture. Whenever a button is clicked, it will send the signal for other events to also be triggered

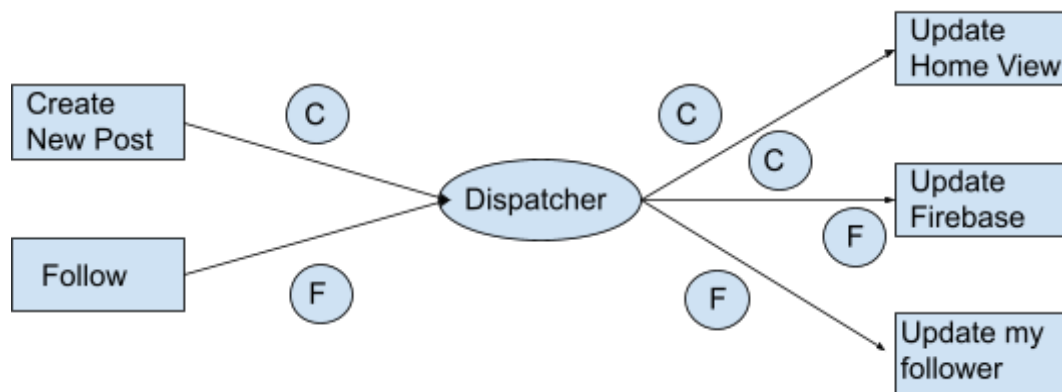


Diagram of events for create new post and follow

In accordance with the diagram above, when Create New Post is triggered, it will send an emitter over to the dispatcher and then over to the relevant events to continue. So an example is where Create New Post will trigger both update home view and update firebase while follow then triggers update firebase and update my follower.

N tier architecture

Our software also implements a N tier or multi-layer architecture.

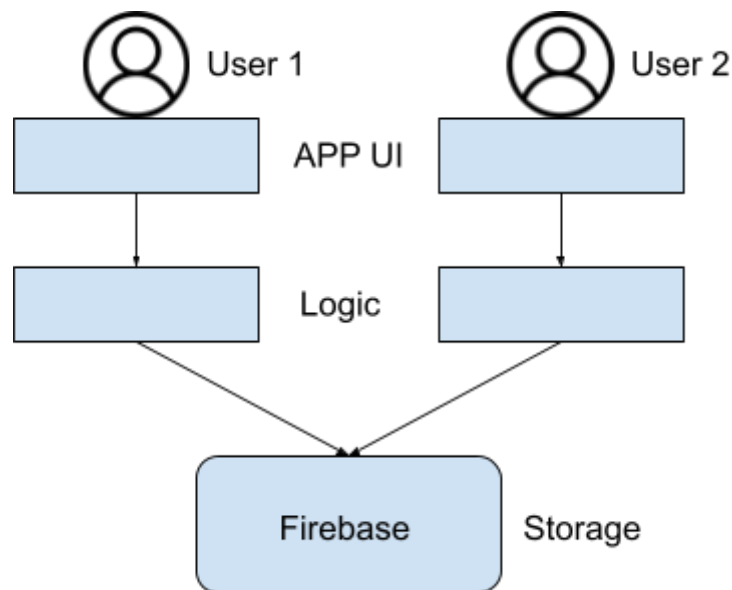


Diagram of N tier structure

As can be seen from the diagram above, within each device, there will be two layers, the UI and then the logic layer, which will then pass on to the server end which is the storage(Firebase)

Software Design Patterns

Singleton Pattern

In our design of the application, we have a few instances where we applied the singleton Pattern. This is to ensure that only one instance of the blogservice object can exist in order to not only save memory, but also allow for it to be easily applied in various areas of our code.

```
static final FirebaseBlogStorage _shared =  
    FirebaseBlogStorage._sharedInstance();  
FirebaseBlogStorage._sharedInstance();  
factory FirebaseBlogStorage() => _shared;
```

Screenshot of the blogService code

In the screenshot above, we can see that there will only be one instance of this class as they will all use this shared static instance.

Model View Controller Pattern (MVC)

In flutter, there are 2 forms of view patterns, namely Model View Controller (MVC) or Model View ViewModel(MVVM). For our application, we decided to make use of the traditional MVC pattern.

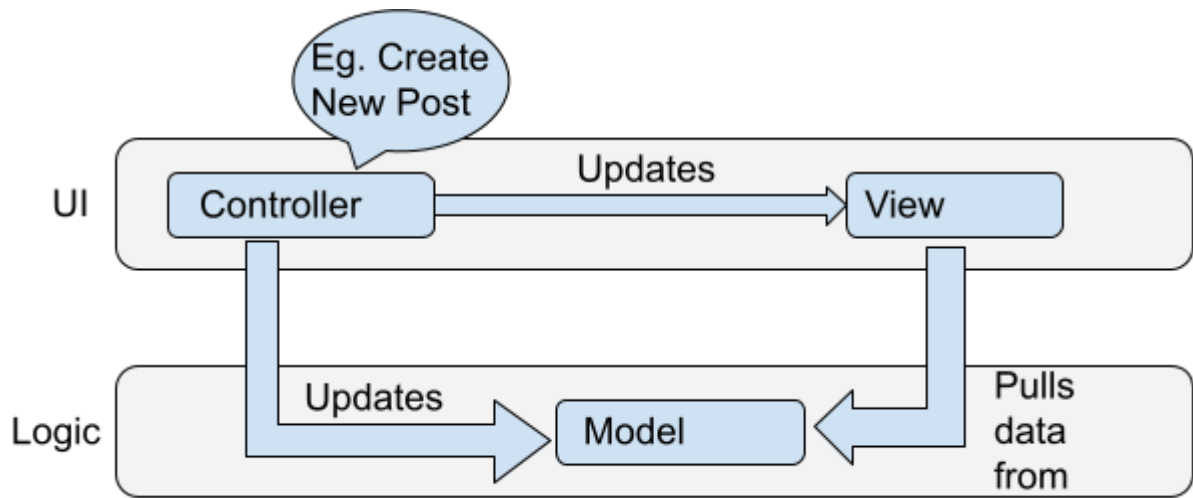


Diagram of MVC

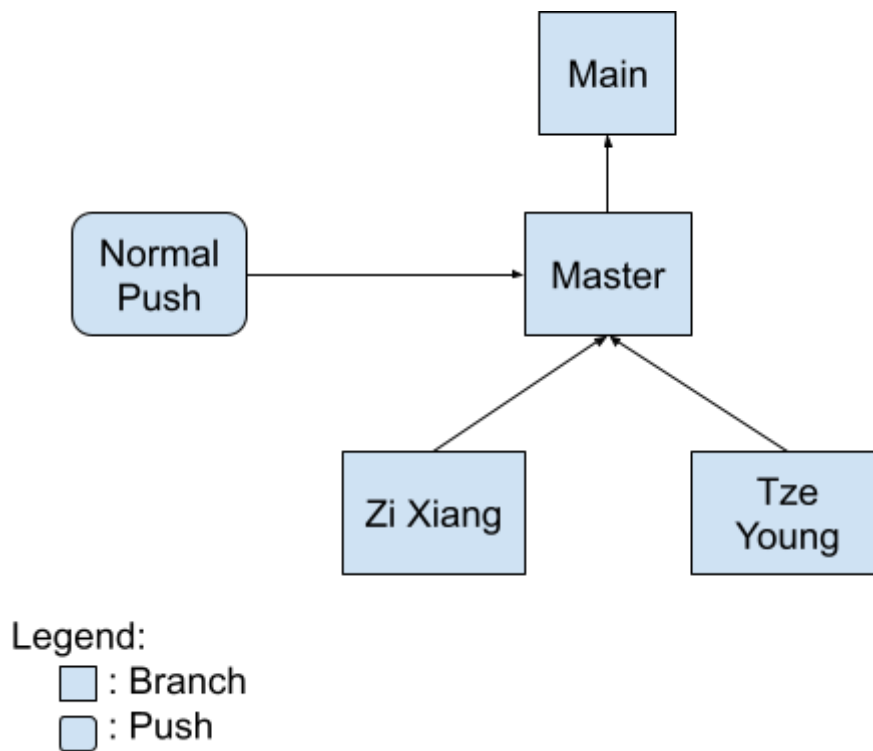
The UI layer will contain both the controller and view while our logic layer contains the model, which is our firestore and firebase storage.

When a button is clicked, such as create new post, it will update the view of the app to contain the new image as well as update the model, our storage, with the new post. The view will then also obtain the new post by pulling the data from the model.

Version Control

Branching

For our github version control, we have multiple branches that we actually use. We have a main branch that will contain the version with all working files and completed features. Next we have a master branch. This is normally the branch that we will push to on our respective ends. We will push to this branch when we have completed a feature or UI portion and would like the other party to test and make sure that it is working as intended. Sometimes we also use this branch to push incomplete features so that the other person can help to fix the issues. Should we have a huge problem with the code such as a compilation error or some major bug like flutter issues, we will push to our own branches and have the other person clone it and do testing. This is to ensure that major bugs never end up on any working branch.



Note that the names may differ slightly on the actual github repo

Pull requests and issues

Whenever we have a feature or a particular portion of the code that we would like to work on, we would first create an issue for it. We make sure to assign the correct person to the feature, along with the label and milestone. After we are done with that particular issue, we will then inform the other person before closing it.

We also make use of pull requests to close some of these issues. However, we only use pull requests when merging from master to main to inform one another that we are pushing the current complete version up.

Testings:

Unit Testing

As of now, we have a few Unit Testing files such as `auth_test` and `database_test`. Each of these tests is designed to test the functionality of each function. For the sake of testing, we used an additional plugin that flutter provides called `flutter_test`. All test files are then stored at the root of the project. We use the command `flutter test test/filename_test.dart` to run the test and see the outcome.

Auth_Test:

We used a mock version of the `auth_service` class in order to test the functionalities of the functions in this class.

```
Run | Debug
group('Mock Authentication', () {
  final provider = MockAuthProvider();
  Run | Debug
  test('Should not be initialized to begin with',
    () => expect(provider.isInitialized, false));

  Run | Debug
  test(
    'Cannot log out if not initialized',
    () {
      expect(
        provider.logout(),
        throwsA(const TypeMatcher<NotInitializedException>()),
      );
    },
  );
});
```

Screenshot of part of the test

Test No.	Description	Functions Used	Pass / Fail
1	Should not be initialised to begin with	<ul style="list-style-type: none">isInitialised	Pass
2	Cannot log out if not initialised	<ul style="list-style-type: none">logout	Pass
3	Should be able to be initialised	<ul style="list-style-type: none">InitialiseisInitialise	Pass
4	User should be null after initialisation	<ul style="list-style-type: none">currentUser	Pass
5	Should be able to initialise in less than 2 hours	<ul style="list-style-type: none">InitialiseisInitialise	Pass
6	Create user should delegate to login function	<ul style="list-style-type: none">createUser	Pass

7	Logged in user should be able to get verified	<ul style="list-style-type: none"> • sendEmailVerification • currentUser 	Pass
8	Should be able to log out and log in	<ul style="list-style-type: none"> • logOut • logIn • currentUser 	Pass

The reasons for creating the tests as such are as follow:

1. Ensuring that the functions work correct
2. Ensuring that Objects are initialised only when told to
3. Ensuring that login In and Out works

Blog_service:

For Blog_service testing, we had to make use of a mock firebase firestore as well as a mock auth_service to ensure that the testing would go smoothly. The idea behind using these mock versions is to ensure that we get as close to proper testing without affecting the current database.

```
Run | Debug
test('Cannot create other user's posts', () async {
  await _authService.logIn(email: tester1email, password: password);
  expect(
    () async => await _blogService.createNewPost(
      authService: _authService,
      firestoreUser: tester1,
      title: 'title',
      body: 'body',
    ),
    isA<void>());
  expect(
    () async => await _blogService.createNewPost(
      authService: _authService,
      firestoreUser: tester2,
      title: 'title',
      body: 'body',
    ),
    throwsA(const TypeMatcher<InvalidPermissionsException>()));
  _authService.logOut();
});
```

Screenshot of part of the test

Test No.	Description	Functions Used	Pass / Fail
1	Cannot create other user's post	<ul style="list-style-type: none"> • createNewPost 	Pass
2	Cannot update other user's post	<ul style="list-style-type: none"> • updatePost 	Pass
3	Cannot delete other user's post	<ul style="list-style-type: none"> • deletePost 	Pass
4			

5			
---	--	--	--

The reason for designing the tests in this manner is as such:

1. Check that users can only edit their own posts
2. Check that users can affect their own posts without errors

database_test:

For database_test, some slight adjustments had to be made to the code for it to work properly. The main change is using a fake firebase firestore for testing purposes using the fake_cloud_firestore plugin. As such, a mock version of the database class resides at the bottom of this test file.

```
Run | Debug
group("Mock Database", () {
  final database = MockDatabase(uid: uid);
  Run | Debug
  test("Document should not exist", () async {
    var doc = await database.checkExist(uid);
    expect(doc.exists, false);
  });

  Run | Debug
  test("Document should be initialised", () async {
    await database.updateUserData("Unit Test", "Member", 0);
    var doc = await database.checkExist(uid);
    expect(doc.exists, true);
  });
});
```

Screenshot of part of the tests

Test No.	Description	Functions Used	Pass / Fail
1	Document should not exist	<ul style="list-style-type: none"> • checkExist 	Pass
2	Document should not be initialised	<ul style="list-style-type: none"> • updateUserData • checkExist 	Pass
3	Document should contain correct data	<ul style="list-style-type: none"> • currentUserStream 	Pass
4	Document should update correctly	<ul style="list-style-type: none"> • updateUserData • currentUserStream 	Pass
5	Should return a list of all users	<ul style="list-style-type: none"> • updateUserData • stonk 	Pass
6	Document should not contain	<ul style="list-style-type: none"> • checkFollowerExist 	Pass

	follower		
7	Document should have added follower	<ul style="list-style-type: none"> • addFollower • checkFollowerExist 	Pass
8	Follower Count should increase by 1	<ul style="list-style-type: none"> • currentUserStream 	Pass
9	Follower should be removed	<ul style="list-style-type: none"> • removeFollower 	Pass
10	Privacy should be Public by default	<ul style="list-style-type: none"> • currentUserStream 	Pass
11	Privacy should be changed to Private	<ul style="list-style-type: none"> • updatePrivacy • currentUserStream 	Pass

These are the functionalities and functions currently and tested. The reasons for the designing the tests in this manner is as such:

1. Able to check the existence of documents
2. Able to read and write to firestore with no errors
3. Ensure that the removal of documents works

Widget Testing

For widget testing, our goal is to be able to check whether the various child widgets are loading correctly and not so much the data.

Authentication Screens

For the main authentication screens, we use widget testing to check if the widgets are loading correctly.

```
testWidgets("Check Registration Page", (widgetTester) async {
  await widgetTester.pumpWidget(MockRegister());

  /// Checking of top App bar
  final checkSideIcon = find.text("Log In");
  final checkAppBar = find.text("Registration");
  expect(checkAppBar, findsOneWidget);
  expect(checkSideIcon, findsOneWidget);
});
```

Screenshot of part of the widget testing

For the registration and login page, we make sure that the top app bar has all of the text fields and buttons loaded correctly. We also make sure that the correct number of text form fields have been loaded in. Specifically for the login page, we check if the forget password is loading in correctly.

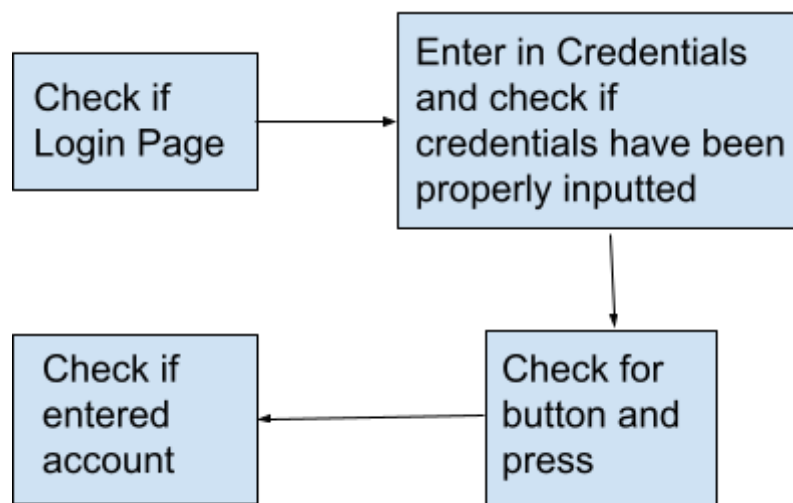
Integration Testing

Now that we have done widget as well as unit testing, we will start with the integration testing to make sure that the various systems such as authentication work correctly.

Authentication Tests

Login_test:

For this test, we tested to see if given the right inputs, would the app be able to login in properly without any errors. Here is the sequence of checks:



Register_test:

Here, we did a similar procedure as compared to login. The only difference is that we check to see if the email was sent out correctly and that after the email verification screen, users would then be brought back over to the login screen.

Reset_password_test:

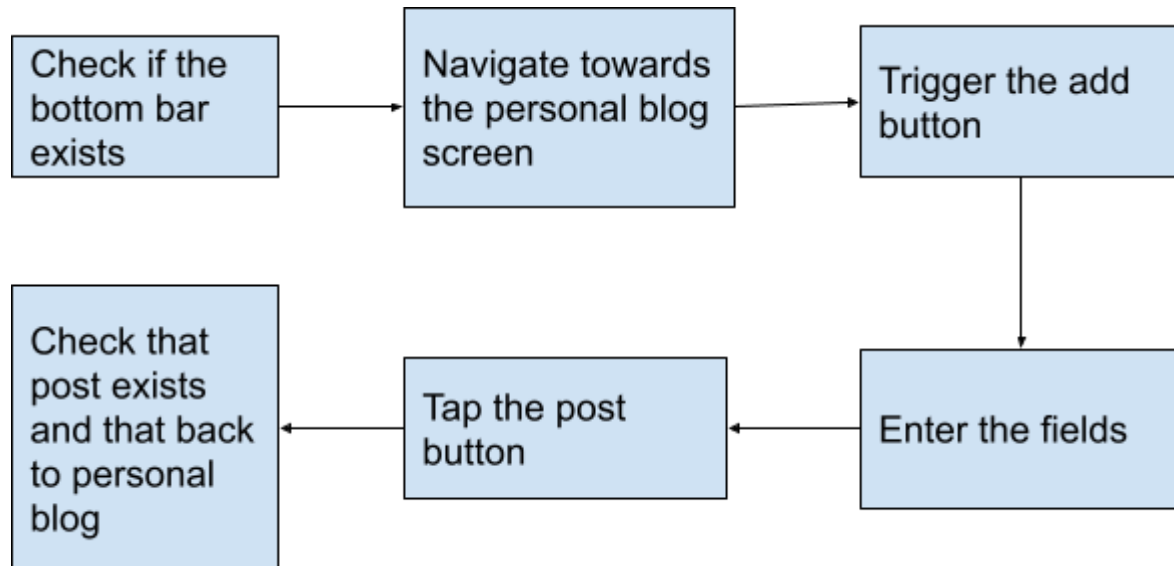
For reset_password, all that was tested was to check if the action snack bar at the bottom of the screen would be displayed correctly should the users actually did request for a reset password email.

Features Testing

For the features side, we have decided to test the more prominent features to ensure that it was working correctly and as intended.

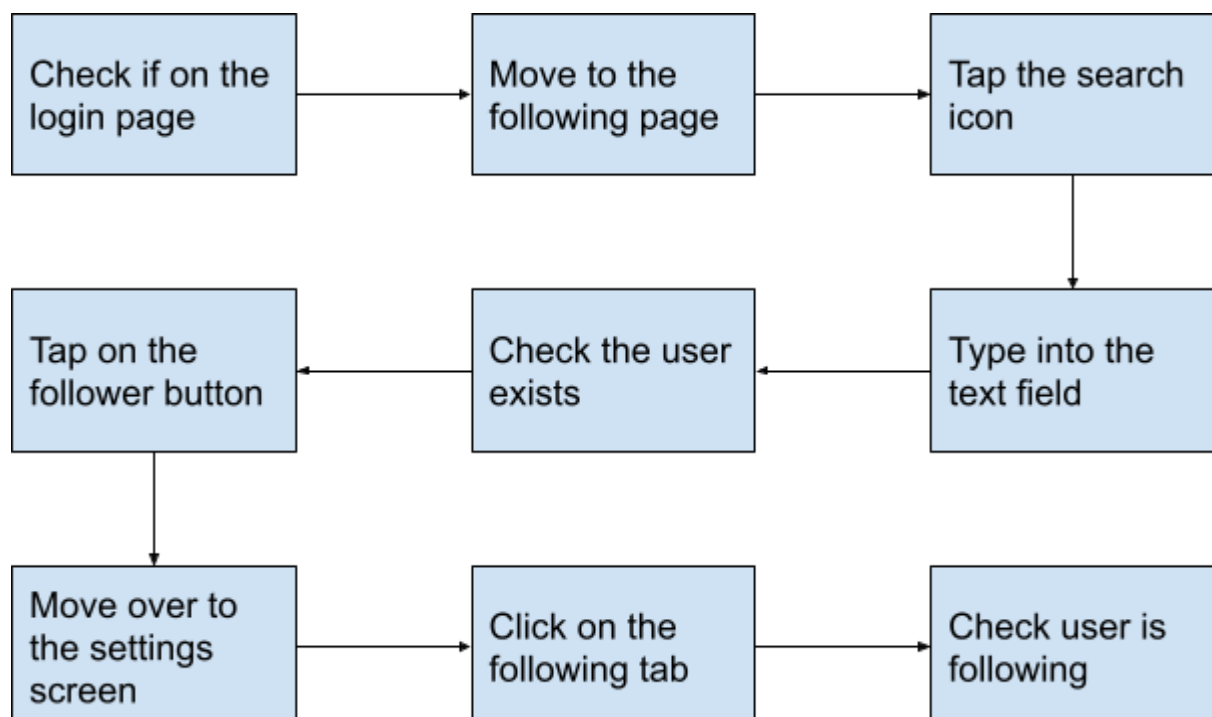
Posts_test:

For this test, we started in the feed screen, from which the test will navigate towards the personal blog screen, then trigger the add button, write the post and then post it.



Follower_test:

For this test, we once again did a similar form of check as the post test. However, this test actually goes between more screens to check if photos also load correctly and check if the other screens work.

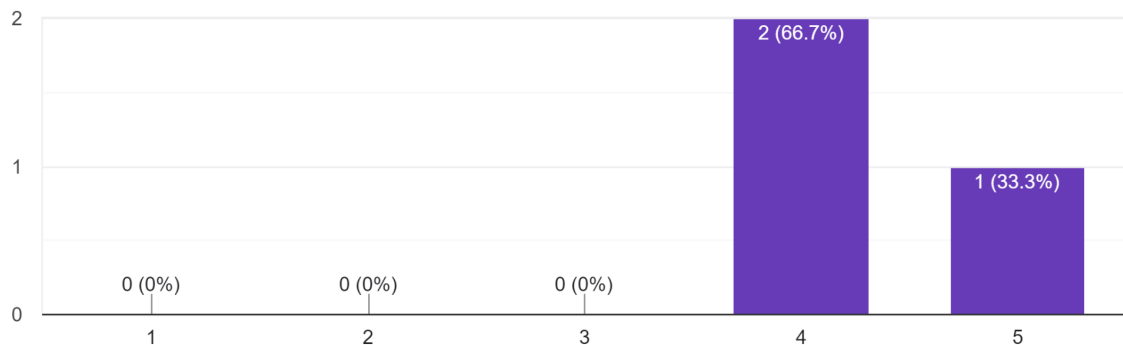


User Testing

Another form of testing that we decided to make use of is user testing. For user testing we decided to make a survey using google forms to obtain data from our testers. Questions in the survey include things like whether the app was easy to use and whether there were any prominent bugs.

Are the features relevant to the app?

3 responses






Comments from the user testing:

- Overall the features are relevant
- However, there seems to be a bug when using the stock profile portion

When I bought stocks, the whole portfolio page crashed. Had to restart the whole app. 5/7 experience.
Maybe can add more profile stuff.

However, this is a limitation of the application and not something that can currently be fixed. One way to circumvent the issue is the use the paid version of the alpha vantage API which would allow more calls and thus prevent the call limit and crashing.

Tech Stack

1. Flutter: For front-end and UI design 
 - a. Used dcdg for UML class diagram creation
2. Firebase: For back-end and database access with authentication 
 - a. Used Firestore for real-time database storage
3. Git: For version control and backup purposes 

Milestone 1 Overall Current Progress

1. Login and Registration System in place
 - a. Email Verification
 - b. Basic Input validation
 - c. Error Dialog box popup
2. Link to Firestore established
 - a. Successful update and retrieval of data
 - b. Document with User ID as key
3. Rough UI of Home Page with logout feature
 - a. Side screen that allows for the create post and find other users option
 - b. Find users option searches for name that contain the given input

Milestone 2 Overall Current Progress



1. Added the blogging service
 - a. Able to create new post
 - b. Able to edit them
 - c. Able to also delete them
2. Added bottom navigation bar to aid the moving between screens
 - a. Bottom bar contains the essential tabs
 - b. Some screens also have links to other screens
3. Added following and my followers feature
 - a. Access following and followers from settings
 - b. Can follow and also unfollow

4. Add profile picture and profile picture upload
 - a. Has a notification system if notifications are enabled for the app
5. Added stock market screen
 - a. Able to declare buying and selling various stocks
 - b. Able to search for existing stock
 - c. Error will be thrown when declare a non existence stock

Milestone 3 Overall Current Progress

1. Added stock history to allow users to view past transactions
2. Added privacy settings so that users are able to decide whether or not to display their stock profile to various groups
3. Updated the view profile page
 - a. Includes profile, blogs and possibly stocks
4. Added the ranking of posts based on the number of upvotes
 - a. Allow users to see which is the most popular post chosen by the community

Timeline and Development Plan

MS	Task	Description	In-Charge	Timestamp
1	Basic Research on flutter and firebase 	Look up how to use the widgets and classes from flutter.	Tze Young Zi Xiang	1 - 8 May
		Create a firebase account to test and see how to make use of the authentication features and firestore.		
	Creation of Login and Registration System functionality 	Basic features for a login and registration A way to swap between the two screens	Tze Young Zi Xiang	5 - 10 May
	Email	Ensure that the user has	Tze Young	8 - 15 May

	Verification (Firebase) ✓	the actual email account and has verified it		
	Error and Exception Catching ✓	Should there be an error message thrown by firebase, return it has a dialog box for the user to view the issue when logging in	Tze Young	
	Login and Registration System UI ✓	Design the UI for the authentication pages	Zi Xiang	18 - 29 May
	Establish Link to Firestore ✓	Turn on the firestore link via firebase and also ensure that the app can receive and update the data in firestore	Zi Xiang	
	UI of home view ✓	Add relevant widgets for the home view page. These widgets would later be updated for their various purposes	Zi Xiang	
Milestone 1 Progress: <ul style="list-style-type: none"> Ideation Proof of concept: <ul style="list-style-type: none"> Login and Registration System Backend services roughly operation Home Page UI 				29 May
2	Finishing Backend Firestore user services ✓	Finalise creation, reading, updating, and deletion of firestore data (CRUD operations)	Zi Xiang	30 May - 7 June
	Posting and blogging database services ✓	Create a page to allow users to post	Tze Young	4 - 20 June
	Handling of user data ✓	Feature to follow and view other user's posts	Zi Xiang	11 - 20 June
		Rank in terms of follower count		
		Search for other users by their account name		

	Remember Me ✗ (Decided to remove due to lack of use)	Stay logged in into the app without having to retype the email and password each time	Tze Young	20 - 26 June
	Create new Post page UI ✓	Display new posts page and also display the posts on home page	Tze Young	20 -26 June
	Resetting of password ✓	Add a feature where users are able to reset their password if they forget it	Zi Xiang	20 - 26 June
	Testing with unit and widget testing ✓	Test the various widgets and services	Tze Young Zi Xiang	20 -26 June
Milestone2 : Trial Run <ul style="list-style-type: none"> • Blogging System • Following and searching of fellow users 				26 June
3	Real-time data from markets ✓	Get data from SGX and update their investment portfolio in real-time	Tze Young	26 June - 10 July
	Commenting other user posts ✓	Allow for users to comment and critique the other user's posts	Zi Xiang	26 June - 10 July
	Refinement of UI ✓	Further beautify the app	Tze Young Zi Xiang	Throughout MS 3
	Testing using integration testing ✓	Test the app as a whole rather than just the individual components	Tze Young Zi Xiang	10 - 24 July
	Testing of MVP ✓	Make sure that all required features are up and running. Errors are properly handled	Tze Young Zi Xiang	10 -24 July
Milestone 3: MVP <ul style="list-style-type: none"> • All above features to be operational • App to be tested 				24 July
4	Custom valuation Metrics	Feature to help users calculate their own valuation metrics for their stocks	Tze Young Zi Xiang	TBC

	Further testing of features	Use other forms of testing to ensure features are working Testing of the new additional features	Tze Young Zi Xiang	
	Logging in with alias	Allow for users to login with a specific ID or name instead of full email	Zi Xiang	
	Further refining of UI	Make sure all additional features fit with the current theme	Tze Young Zi Xiang	
Splashdown: MVP with additional features <ul style="list-style-type: none"> • MVP with some new features like signing in with alias 				23 August

Current Testing Limitations

1. There is no IOS version for now. Should you not have an android device or emulator, let us know so we can do a live demonstration.
2. There are no test accounts due to the current limitations. Testers would need to either register and create their own account or let us know so that we can do a live demonstration.
3. Do note that there is a chance that when you start the app for the first time, you may be automatically logged in. If you are logged in and some screens are greyed out, please close the app fully and restart it. If you happen to be logged in to another account, log out and start testing from there.
4. Do note that you can only add 5 stocks per minute and that declaration of stocks may take a while to update. If no errors appear on screen, it is currently in the process of updating and once done, it will return the user to the portfolio page.
5. This is a continuation from point 4. Due to API limitations, this 5 calls is across all accounts and as such, if you are unable to add to test, let us know and we can either do a live demo or delete a stock from one of the existing accounts.

*** This is here as a side note, but the API we are using is able to do more than 5 calls if we pay ***

Milestone 3 Links

Class Diagram

Class Diagram:  [Orbital We Believe.svg](#)

- Note to download the svg file and open on a browser to view the full diagram

Work Log

Refer to attached spreadsheet:  [Project Log](#)

Poster

Refer to poster:  Orbital milestone 3 poster.png

Video Demonstration

Refer to the video demonstration: [MS3 vid.mp4](#)

APK

Refer to the APK: [We Believe.apk](#)

Milestone 2 Link

Google Drive: [Milestone 2](#)

Milestone 1 Link

Google Drive: [Milestone 1](#)

Acknowledgement

External Packages Used

1. Firebase_auth
2. Firebase_core
3. Cloud_firestore
4. Dcdg (Class UML Diagram)
5. Easy_search_bar
6. Persistent_bottom_nav_bar
7. Intl
8. Http
9. Permission_handler
10. Image_picker
11. Device_info_plus
12. Firebase_storage
13. Fake_cloud_firestore
14. Firebase_storage_mock
15. Flutter_local_notifications
16. Pie_chart
17. Shared_preferences
18. is_first_run

Sources referred

1. The Net Ninja (Youtube) @@author {iamshaunjp}
- 2.