FIND MY PET – CROSS PLATFORM MOBILE APPLICATION

TECHNICAL MANUAL

Supervisor:	Paul Barry
Author:	Martin Walsh
ID:	C00170339

Contents

Table of Figures

Abstract

The purpose of this project is to create a cross platform mobile application (iOS & Android) in which the user can post details of lost or found animals in their area. This mobile application will be written using Flutter Technology, an open-source UI software development kit created by Google. The application will present the user with the option to post or view lost and found animals in a selected area. There will be a range of features, such as push notifications if an animal is reported lost in a user's area, searching for posts based on location and communicating with other users via comments.

1. Introduction

This is a technical manual for the fourth-year student project named *FindMyPet*. *FindMyPet* is a mobile application designed for both iOS and Android with the aim of helping its users to find their lost pets quickly and safely.

It can be suggested that the likelihood of a lost pet being returned home is increased if an image and description of the animal is made available to the public in that area. It is common to see dogs wandering the streets without an owner in Ireland and is difficult to gauge if they are lost. Ideally the app will spread awareness of lost animals in the area. In turn, this will help users make a connection between the lost pet listed on the app and the animal they spot on the street.

The project was built using Flutter, Googles UI toolkit for building cross platform mobile application using a single codebase. It uses an object-oriented programming language called Dart. Dart is similar to C in terms of syntax and it compiles to native code.

2. Firebase

Firebase is a Backend-as-a-Service (Baas), used for mobile and web app development. It was selected during the research stage of this project as it offers a multitude of tools and services that are ideal for this project. The following are key tools Firebase offers that were used in this project. This section provides useful information on what each does and how it works. These tools will be referenced in the screens and functionality section.

Firebase is directly connected with the *FindMyPet* mobile application. This was done by registering the application with Firebase. This involved multiple steps such as sharing the apps package name with Firebase and the Debug signing certificate SHA-1. Firebase then generates a configuration file to be added to the app. From there it required certain changes and code snippets be added to build files within the app. The last step included adding the Firebase SDKs to the app.

This allows for higher security as Firebase can work directly with the app by creating an instance of the Firebase tools needed within the code. As a result, no API keys or other potentially exploitable items need to be written into the application.

2.1 Authentication

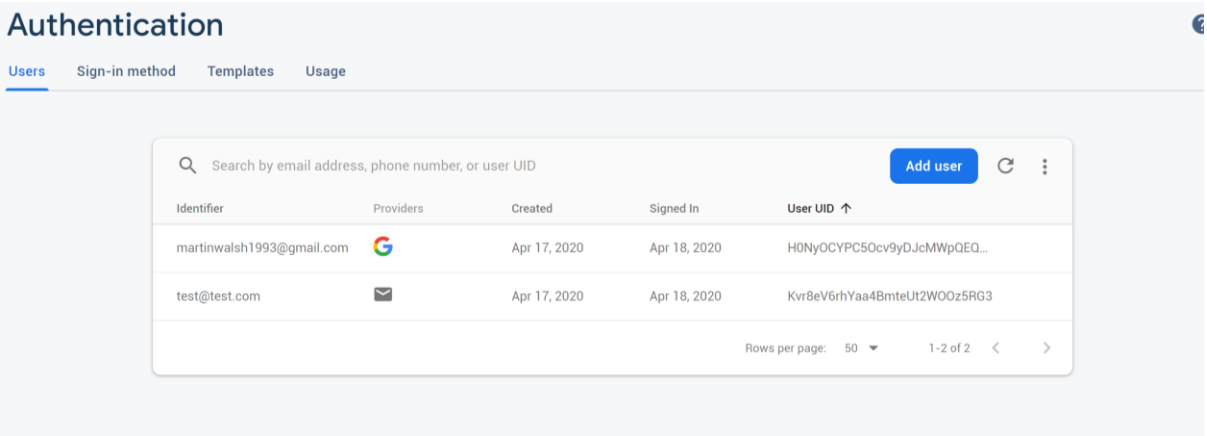


Figure 1: Firebase Authentication

Firebase authentication is used in this project to both obtain and create authenticated credentials for a user. In Figure 1 there are two users authenticated with the application, one via Google sign-in and the other through an email and password combination.

The user can choose to create an account with either of these methods. Once a user selects their preferred route their credentials are passed to the Firebase Authentication SDK. Credentials are generated if the user is signing up for the first time. When they log in, the backend service will verify those credentials and return a response to the application.

2.2 Cloud Firestore

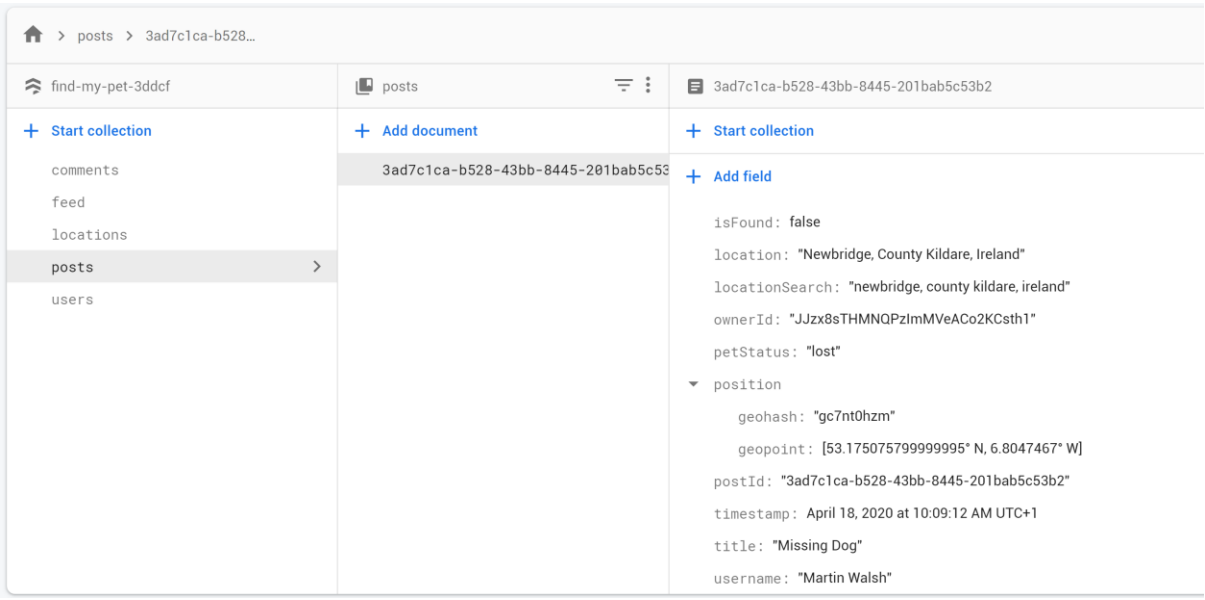


Figure 2: Cloud Firestore

Cloud Firestore is a NoSQL database that stores the data used in this project. It is a scalable database and offers the ability to set up real-time listeners within the code to keep app data in sync.

The database stores information in collections, these collections contain documents which contain data. In the above example (Figure 2) “posts” is a collection which contains a document, that document contains data for that post.

2.3 Storage

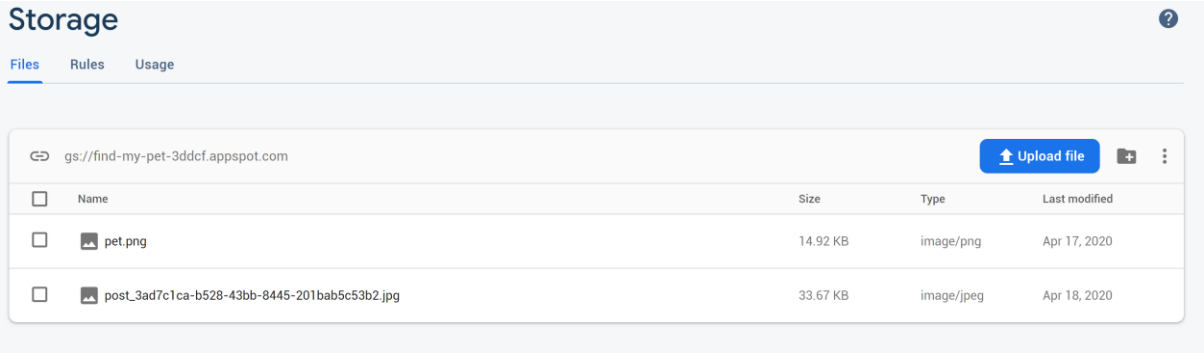


Figure 3: Firebase Storage

Firebase storage gives the ability to upload media data. For the purposes of this project it was used for images. In order to reference and use the images again the filename must be known. Therefore, each posts image is stored under the post ID that it relates to (Figure 3).

2.4 Functions

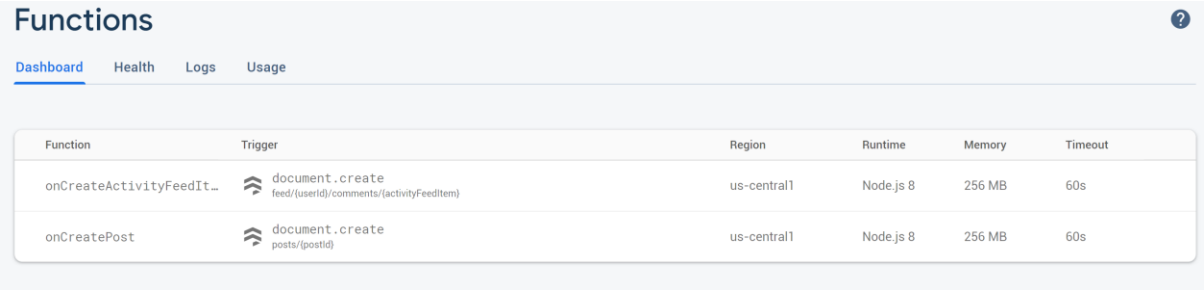


Figure 4: Firebase Functions

Functions are a serverless framework that automatically run backend code in response to an event being triggered. For this project the functions were written in JavaScript and are used in conjunction with Firebase Cloud Messaging, to send notifications if a new post is created within 10km of a user. It is also utilised when a user comments on a post.

The application code does not need to be changed to add or modify functions, they can be deployed separately, making it a very powerful tool. The only prerequisite is that the application contains code that allows it to handle and display these notifications.

3. Screens and Functionality

3.1 Authentication

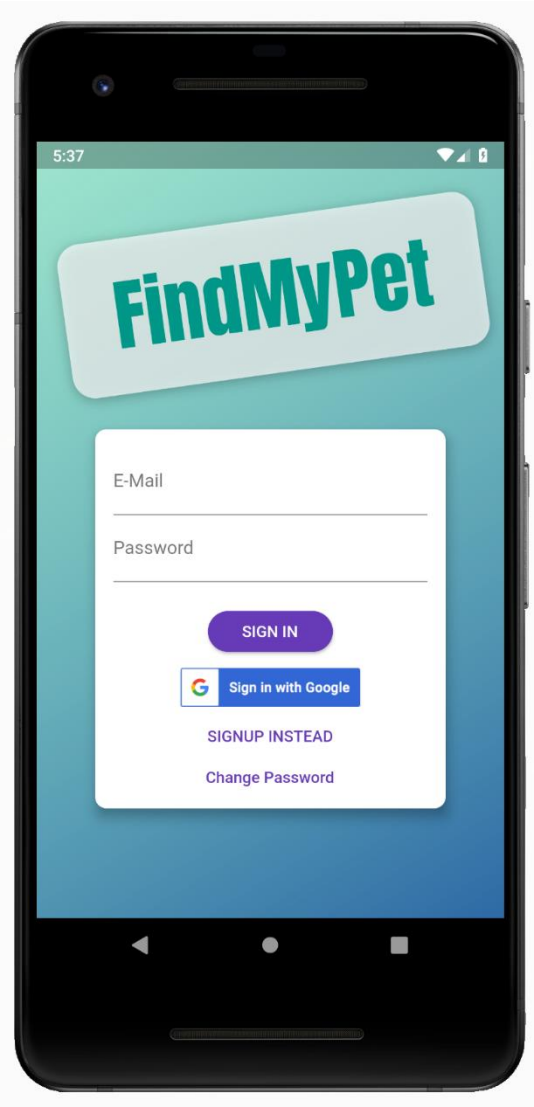


Figure 5: Login Screen

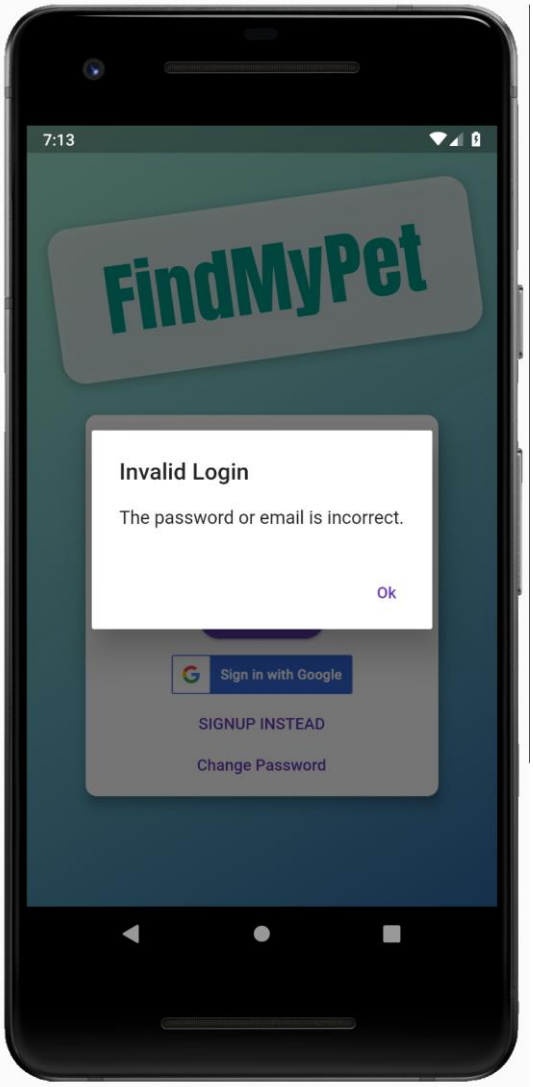


Figure 6: Login Error

This is the screen the user is presented with when the app is opened (Figure 5). There are four possible actions the user can carry out.

1. The user can login via the email and password combination. When this is done the application connects with firebase authentication to check whether the user's credentials are correct. If successful, it will sign the user in and updates the Auth State stream, which will automatically navigate the user to the home page.
   If the authentication fails, the user will be show an alert box, stating that the password or email is incorrect (Figure 6). The error message being general was a conscious choice as to not give away too much information to a potentially malicious user.

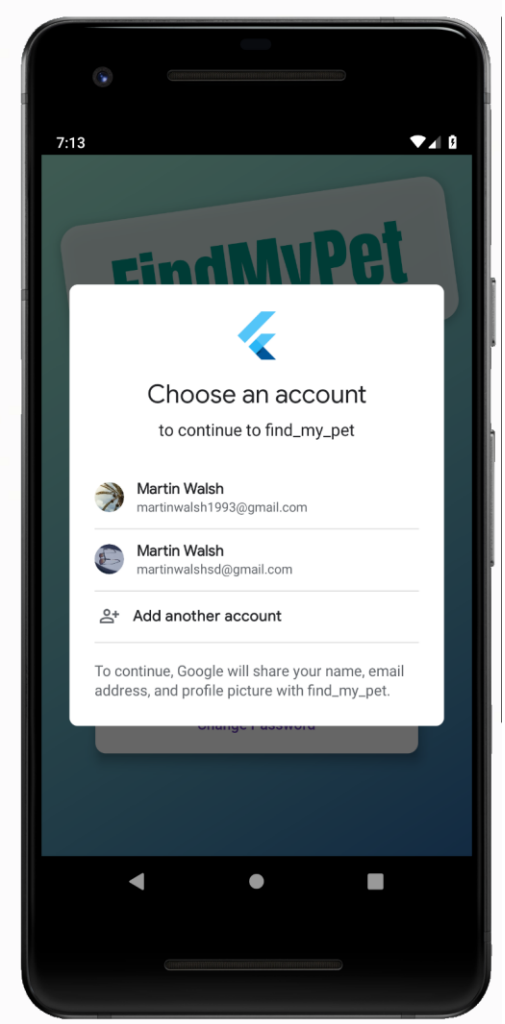


Figure 7: Google Login Screen

2. The user also has the option to login using 'Sign in with Google'. If the user chooses this option, the application again must authenticate with Firebase, but the backend is different as a Google sign-in package must be imported and different functions carried out. It is an easier process for the user as no data needs to be entered, they choose their existing account (Figure 7) and if authenticated are moved to the home screen.

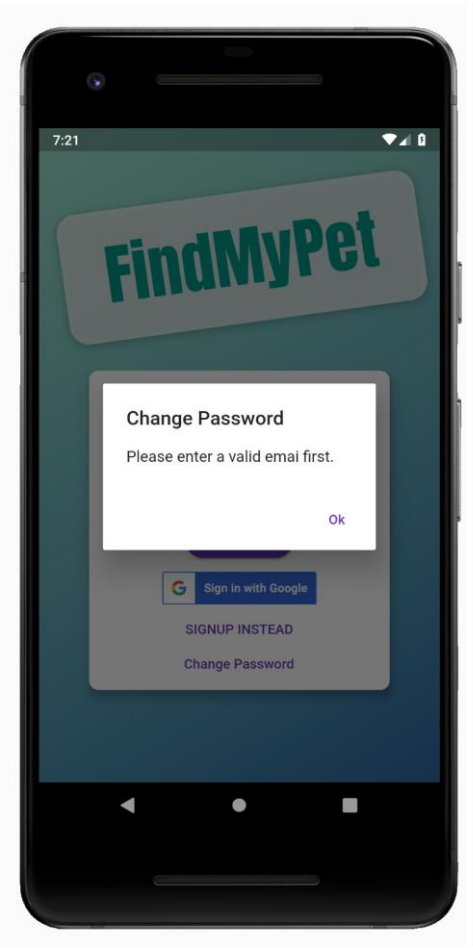


Figure 8: Change Password Error

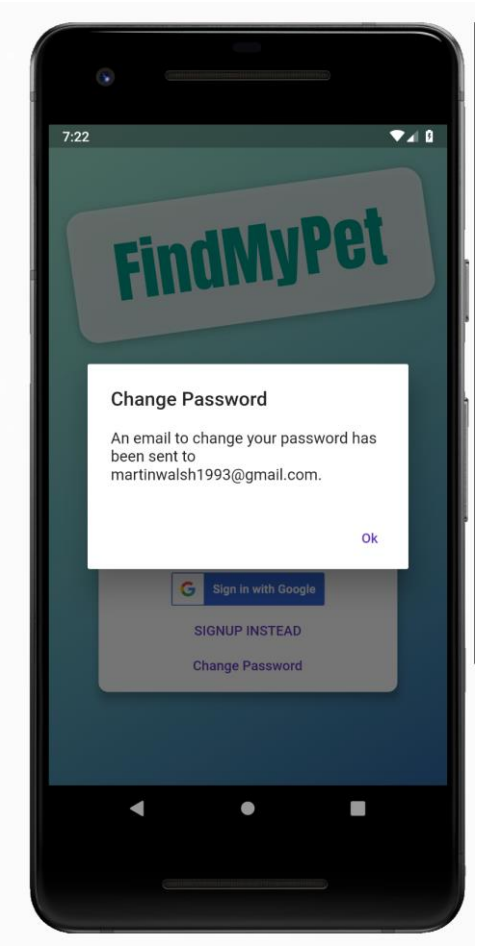


Figure 9: Change Password Success

3. There is an option for a user to change their password if forgotten. To keep the UI simple, the user simply must enter their email and tap the change password button. If the user taps the button without entering a valid email first there will be a pop up instructing them on what to do (Figure 8). If the process is carried out correctly, they will receive a different pop up letting them know the email to change their password has been sent (Figure 9). They can then follow the link to enter and save their new password (Figures 10 and 11).

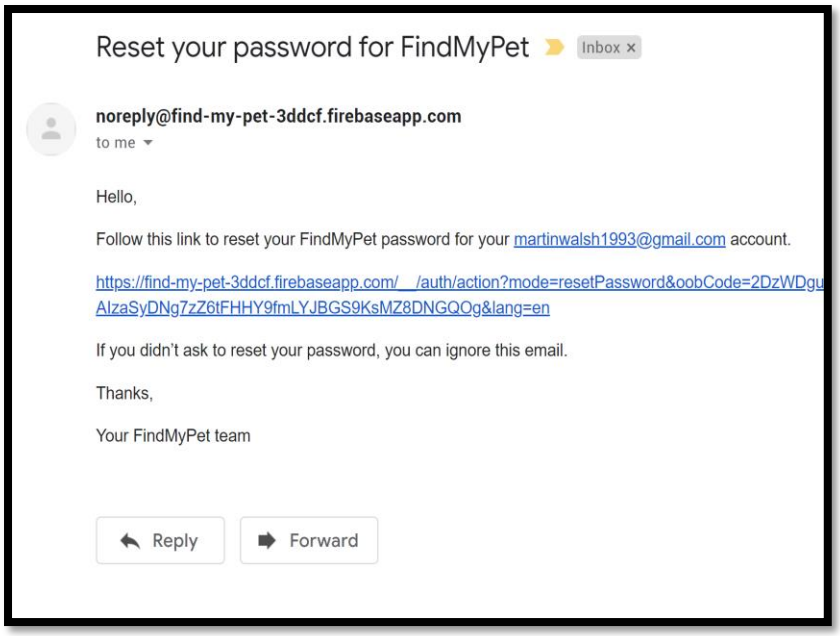


Figure 10: Change Password Email

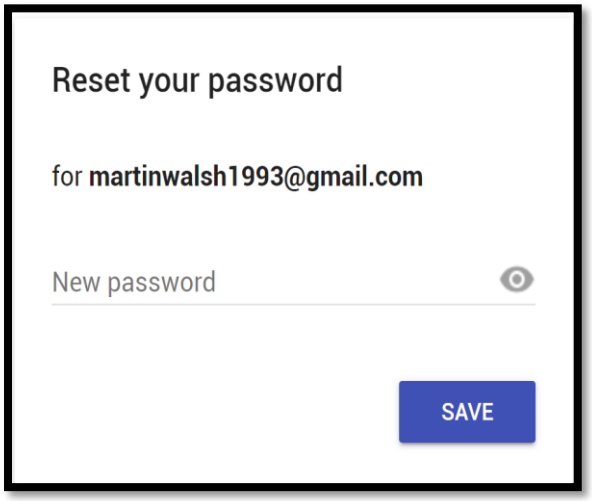


Figure 11: Change Password Screen

4. The user also has the option to sign-up if they have not yet created an account. In keeping with trying to keep the UI simple, this will not navigate to a new page but instead only the contents in the white box will change. This was carried out by wrapping certain display elements in conditional statements. When the user taps sign-up, certain elements will be replaced with new ones (Figure 12).

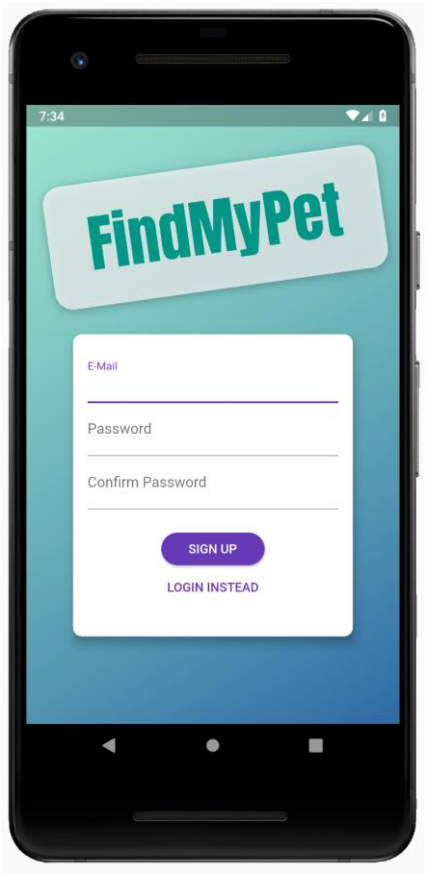


Figure 12: Sign Up Screen

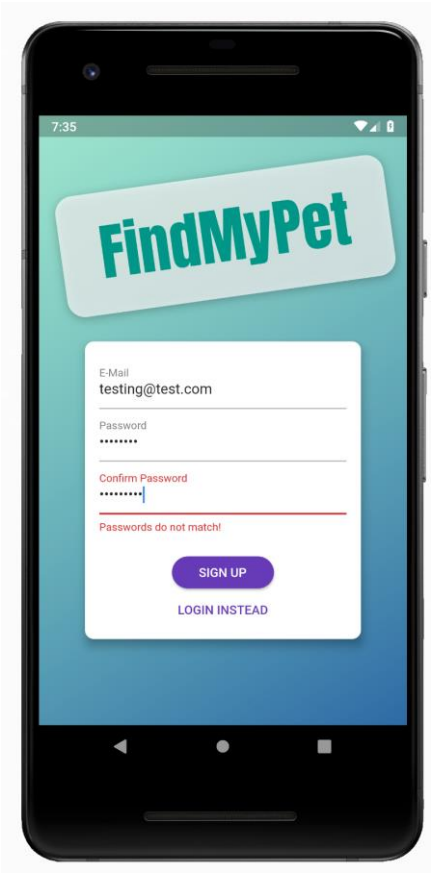


Figure 13: Incorrect Password

The sign-up screen consists of entering an email, password and confirmation password. There is error checking in place for things such as insufficient passwords (Figure 13) and checks against existing email addresses to ensure the same email is not used twice (Figure 14). When the user does enter sufficient details, the application connects with Firebase and creates credentials for that user. An entry is added to the Firestore database in order to store information about that user (Figure 16). Once completed the user is prompted to let them know they can sign-in with their submitted credentials (Figure 15).

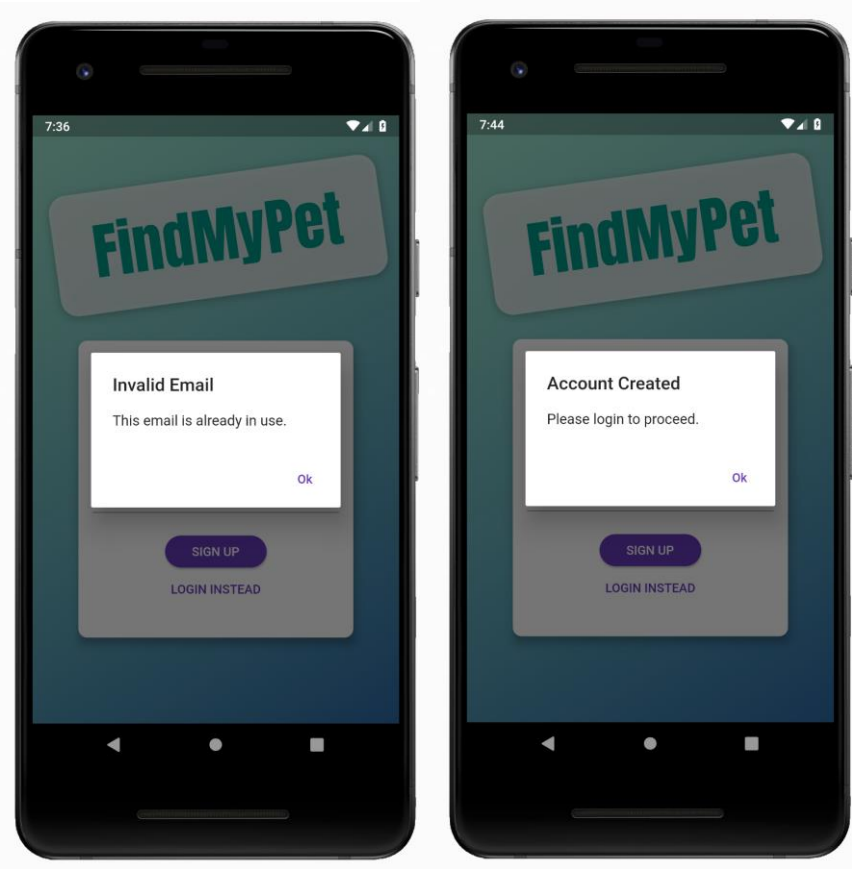


Figure 14: Invalid Email *Figure 15: Account Created*

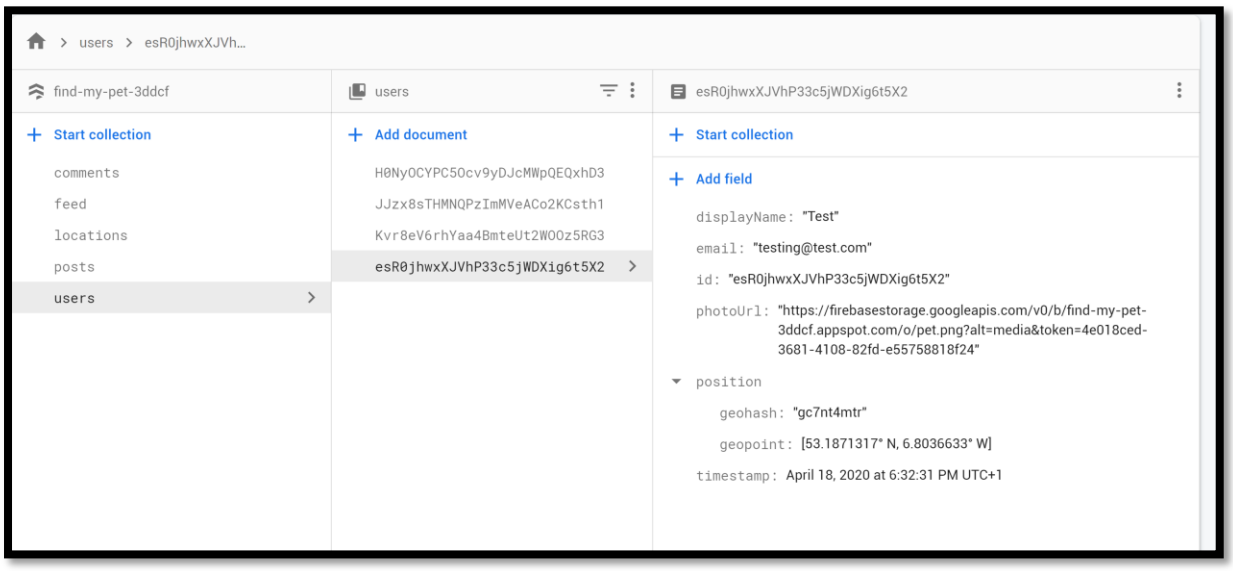


Figure 16: Database Entry

When the user logs in by either means there is a notification token generated for that user, which is required for sending push notifications. The user's location is also stored, which is needed for determining which notifications they should receive based on their given location.

For both authentication types when the user leaves the app and reopens it at a later stage, the application checks if the previous authentication has expired. If it has not expired the user will be automatically logged in and directed to the home page.

Additionally, if the user logs in via Google, their display image is stored and used within the application alongside forms, profile screen and comments they make. If they do not create an account via Google a display image is generated for them.

3.2 Posts Overview

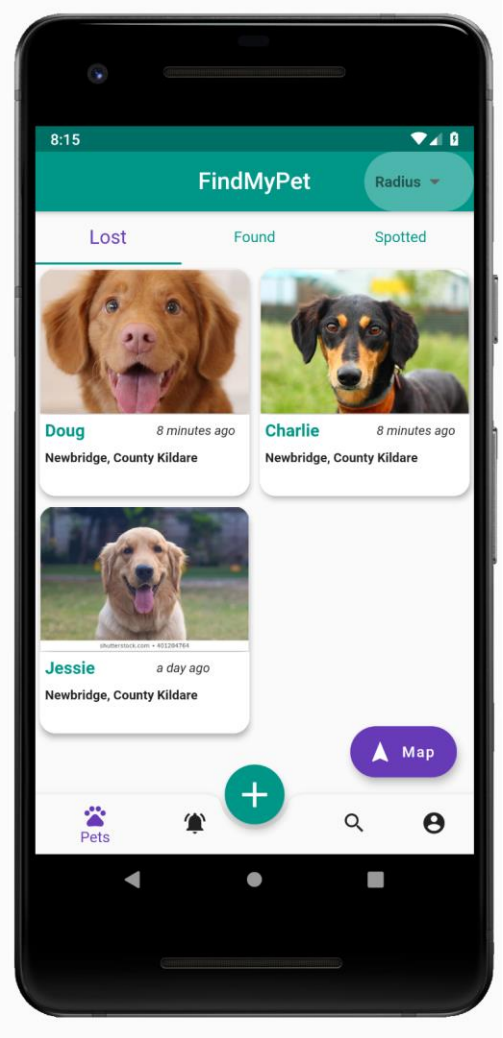


Figure 17: Posts Overview Screen

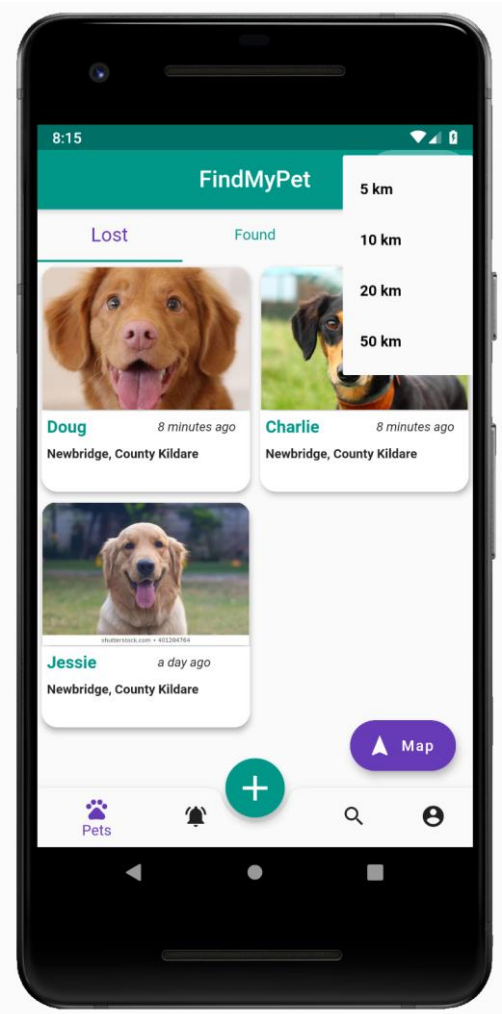


Figure 18: Radius Dropdown Menu

The screen that dispays an overview of the posts (Figure 17) is the first screen to be displayed when the user logs in. This, along with each screen in the application, is displayed dynamically. This means it is responsive to changes in size. The height and width of the screen are calculated and divided amonst the widgets on the screen.

The screen itself has three tabs along the top, for the different types of posts. This went through multiple stages of development as initially the tab only reloaded the relevant posts if tapped on but not if the user was to swipe to another tab. Therefore a controller was introduced, which keeps track of each tab and associates it with a number. The controller is actively listening and will reload the screen with the relevant posts by either tapping on the tab or swiping.

Each tab represents a different type of post. The “Lost” tab is for owners to create a post if their pet has gone missing. “Found” is for individuals who have found a pet by itself and are keeping it until they can find the owner. While “Spotted” is for individuals who have seen a pet without an owner but are unable to take it in. Instead, the user takes a picture and shares it in the “Spotted” section along with a description of where it was last seen.

The summarised time is built using a package that was imported. It alows you to manipulate stored date/time objects to be displayed as such.

There is a radius dropdown menu option in the top bar which when tapped gives the user four distance options (Figure 18). Once the user selects an option from the drop down menu, that pages state will be reinitialised and posts from within the selected radius will be shown.

The user can also tap on any of the posts to bring up a full screen version which contains more detail.

3.3 Individual Post

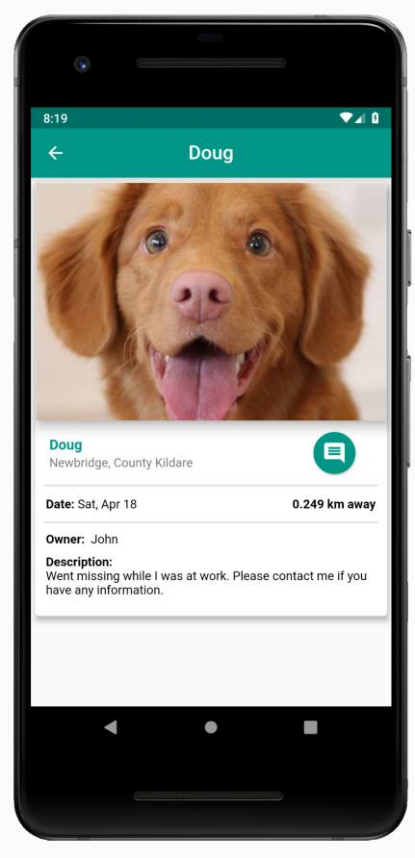


Figure 19: Individual Post Screen

When the user taps on a post that is in a tile view it opens the post in full screen (Figure 19). From here all the information entered about that pet can be viewed.

The information is pulled from the Firestore database and extracted using a predefined data model that stores all the data gathered into chosen variables. This allows the instant creation of a “Post” object by calling on this method.

This page is built in sections, with the image being sourced from Firebase storage. To prevent longer loading times the image is built using a cached image package. This allows for the image to be shown in low quality while waiting on the full image to be ready. It can be suggested this helps make for a better user experience. Additionally, if the image is slow to appear a loading icon will act as an intermediary until the full image is ready. This will allow for the rest of the user experience to go uninterrupted.

The next section contains the pets name, location and floating action button, which if pressed will navigate to the comments page of the relevant post. This is followed by the date and distance. The date is formatted to be displayed in a more friendly and readable way than it is stored. As it is stored as the exact hour, minute, second and includes the date and time zone. The distance is generated dynamically based on the user's current location when opening the post. It is built with a future builder, which is a widget that builds itself based on the data it is getting from the database. The page will load while the distance is being retrieved. If the distance takes time to retrieve a loading spinner will be displayed in its place.

3.4 Comments

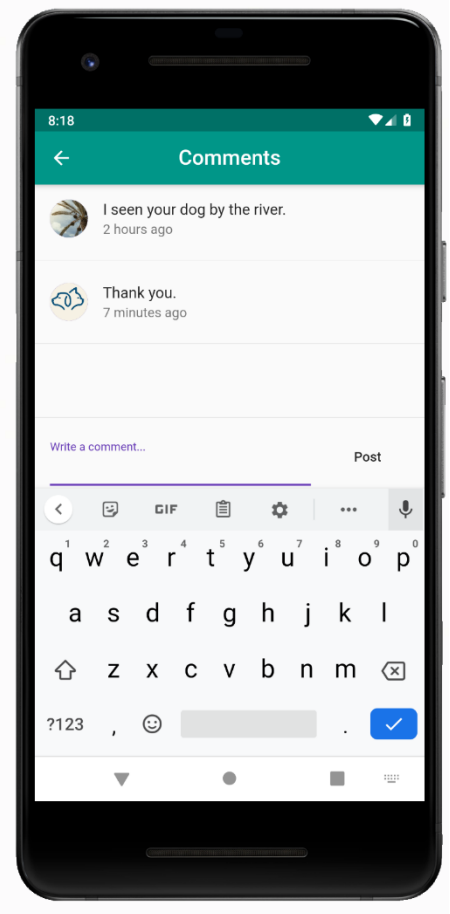


Figure 20: Comments Screen

This screen shows users a list of all comments on a post (Figure 20). It provides the comment data, alongside information about when the comment was posted and the user's avatar. The comments are shown in real time as they are built using a stream. This is a way of setting a function to listen to a particular collection within the database. Therefore, when a user comments it should reflect in the app almost instantly, depending on network strength.

This screen is built using two classes, one being the overall view and layout of the comments. The other being the individual comment objects.

3.5 Map

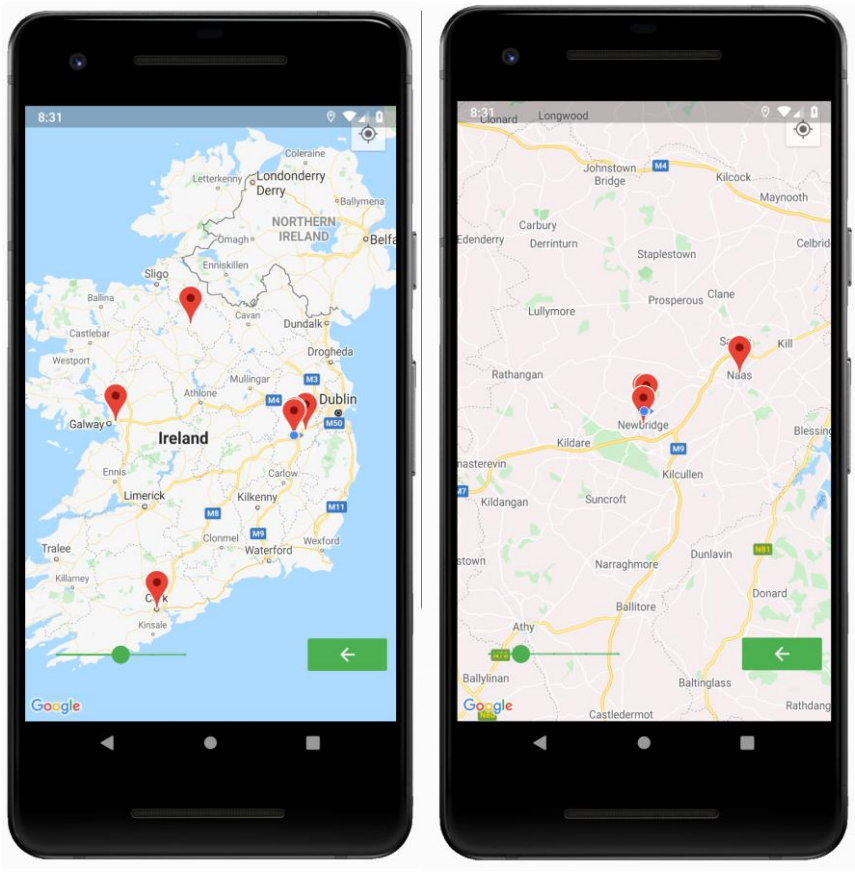


Figure 21: Map zoomed Out *Figure 22: Map Zoomed In*

This screen is used to display all posts in Ireland via markers (Figure 21). It is also built using a stream, so any posts added will be displayed as a marker in real time.

The map has three possible UI interactions. The slider on the bottom left are different levels of zoom (Figure 22). This was achieved by making a small map variable where each key value pair corresponds to different zoom levels. The second button on the bottom right can be used to navigate back to the previous screen. Lastly, the button on the top right will direct the camera to the user's current location. The map also responds to touch to navigate and zoom.

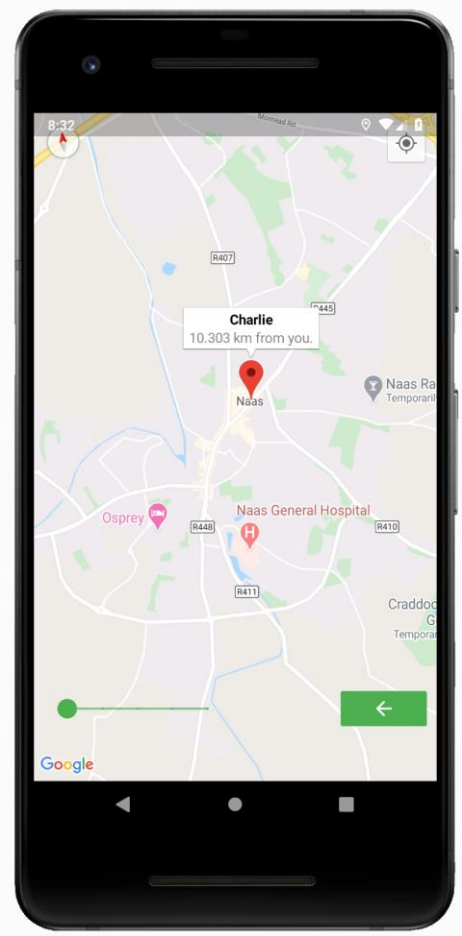


Figure 23: Map Marker Information

By tapping on any of the markers, the name of the post and the distance from the user will appear in a pop-up information window (Figure 23). Tapping on this information window will open the post page.

3.6 Activity Feed

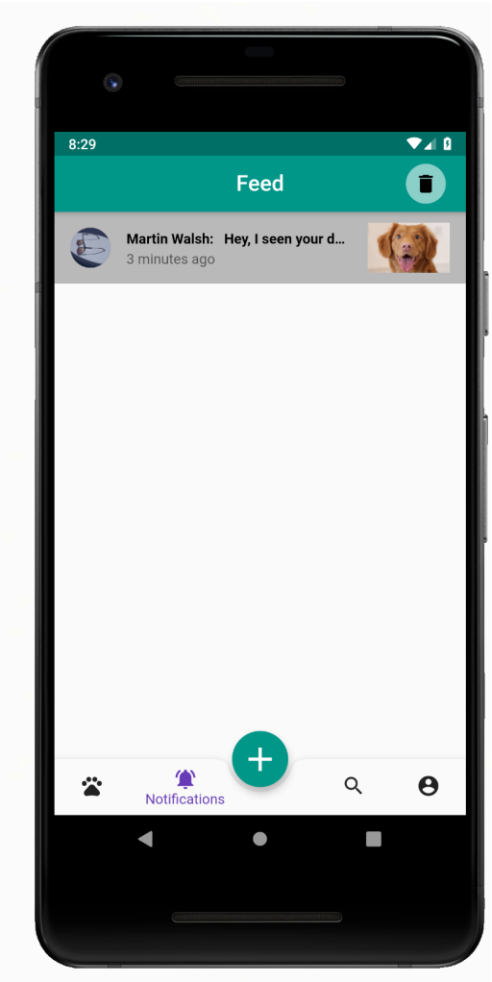


Figure 24: Activity Feed Screen

The activity feed displays any comments that have been made on the user's post. It does this by searching the database for all feed items under the current user's ID. It then adds them to a list which is displayed.

The screen is broken into two different classes. The overall page is built using a future builder which calls on the previously mentioned data before trying to populate the feed items.

The second class is to construct each activity feed item, as seen in (Figure 24). It displays the user's avatar and name, the comment data, time elapsed since it was posted and the posts picture. Each activity feed item can also be tapped on to navigate directly to the post.

3.7 Add Post

Adding a post comprises of multiple screens. The first screen the user sees when hitting the add button is as seen in Figure 25. Which shows the user three different types of posts they can create. Each of which contain a description underneath. When the user picks a post type, that information is stored. In order to correctly store it in the database for later use.

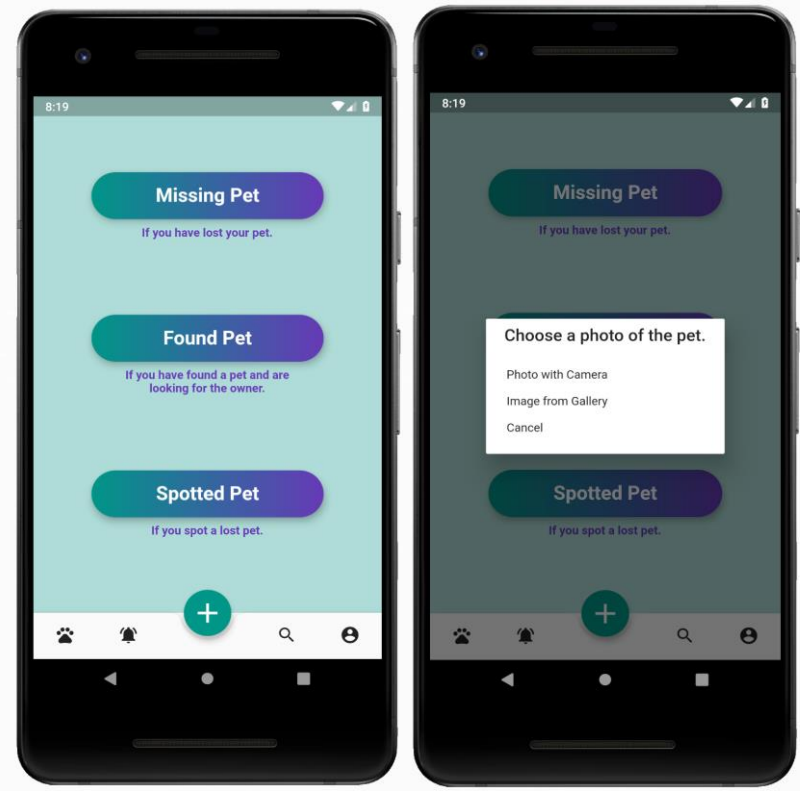


Figure 25: Add Post Screen *Figure 26: Image Choice*

After choosing an option the user is presented with a pop-up screen with two options for choosing an image of their pet (Figure 26). They can choose to take a photo and access their device camera or choose from their gallery which will give them access to all of their stored images. After either choice the image is stored temporarily in the app and displayed back to the user as part of the form (Figure 27).

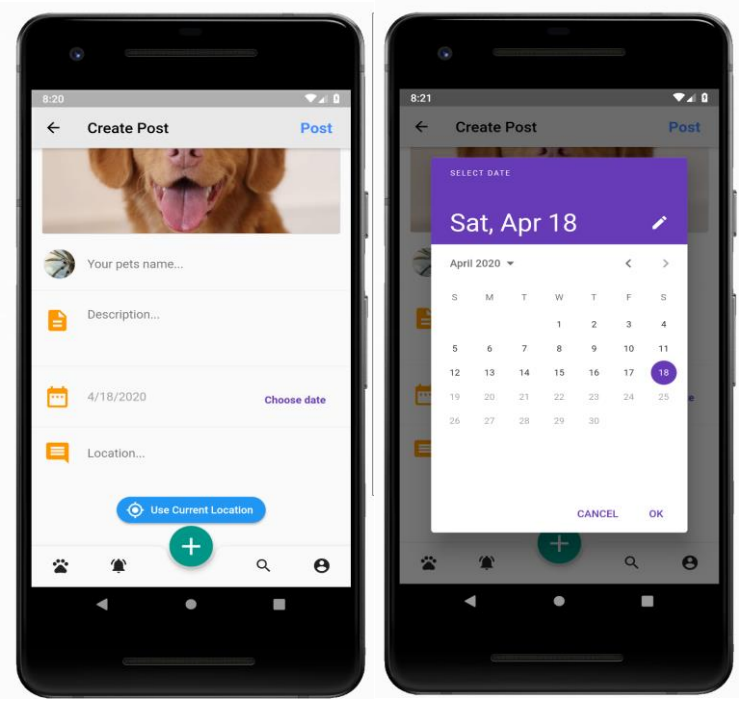


Figure 27: Create Post Screen *Figure 28: Calendar View*

The post form consists of four fields, the first two being normal text fields for the user to fill in. The next is the date field, which defaults to the current date as it is likely to be the date chosen the most. If the date does need to be changed the user can tap the choose date button which produces a fully functioning calendar (Figure 28). A date past the current date cannot be picked.

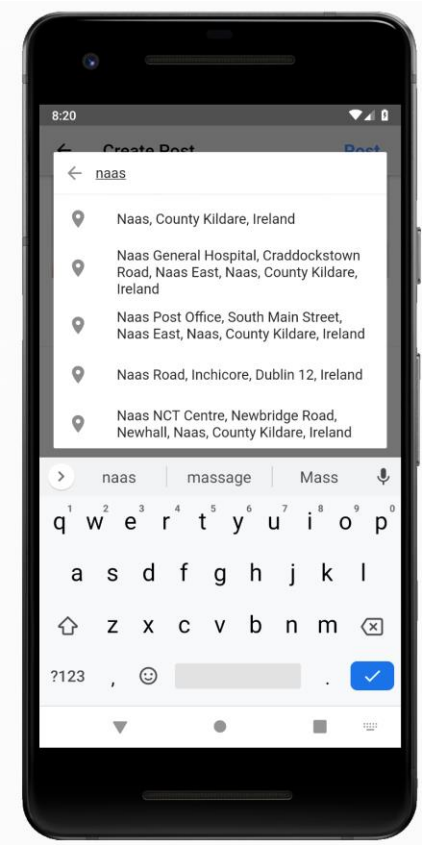


Figure 29: Google Auto Fill Locations

The final field to be filled is the location of the animal when it went missing. The user has two options to fill in this section. They can hit the blue button to get the current location using the devices GPS (Figure 27). The second option is to enter a location manually, which if tapped brings up Googles auto-generated location search. This allows the user to quickly and easily choose their location (Figure 29).

The auto-generated search is coded differently to retrieve the current location as it is using a package provided by Google and receives the data in a different way. The data payload that is returned by this search must have the latitude and longitude extracted so that it can be stored in the database. This is needed to allow other location functionality throughout the app to work.

When the user taps post in the top right of the screen (Figure 27), the post button is then locked to prevent the user trying to upload the same post more than once. Then, multiple actions take place. Firstly, if the keyboard is open it closes and the page view moves to the top of the screen and a loading bar is shown. Then within the application an auto-generated unique ID is created for the post using an imported package. The post is stored under this ID in the posts collection in Firestore. The image is then compressed before being stored in

Firestore Cloud Storage in order to save space and loading times. The post is then added to Firestore in the posts collection and the locations collection.

3.8 Search

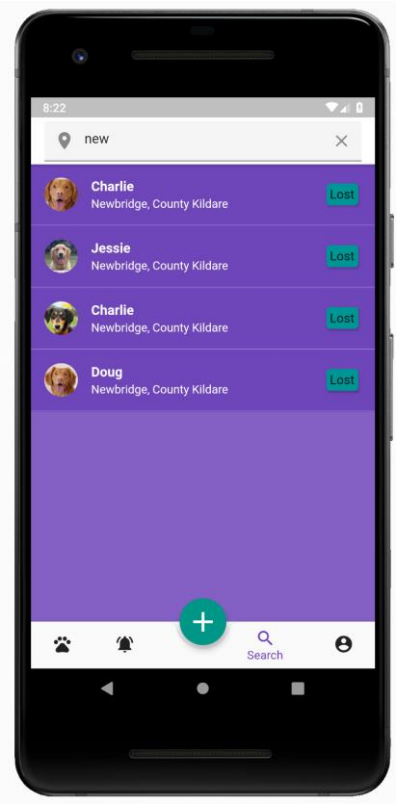


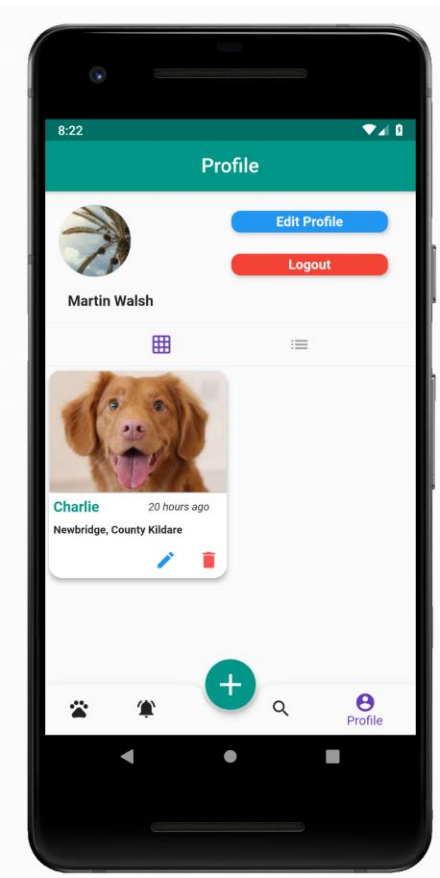
Figure 30: Search Screen

The search screen allows the user to search for posts based on location. It works by comparing the users search query against the location field in the posts collection contained in the database. The search returns results that are equal or greater to the search query, so the user only needs to partially type the location (Figure 30).

Within the search bar there is also a location icon, which if tapped will fill in the search bar with the user's current location. The location is retrieved in real time using the user's GPS. There is also an icon that displays an X, if pressed it will clear the user's search bar.

There is a separate class to build the individual search results and their structure. Each post is displayed with a circular image of the posts actual photo. Alongside the pet's name and location, and the status of that pet. If the user taps on the post, they will navigate to the post itself.

3.9 Profile



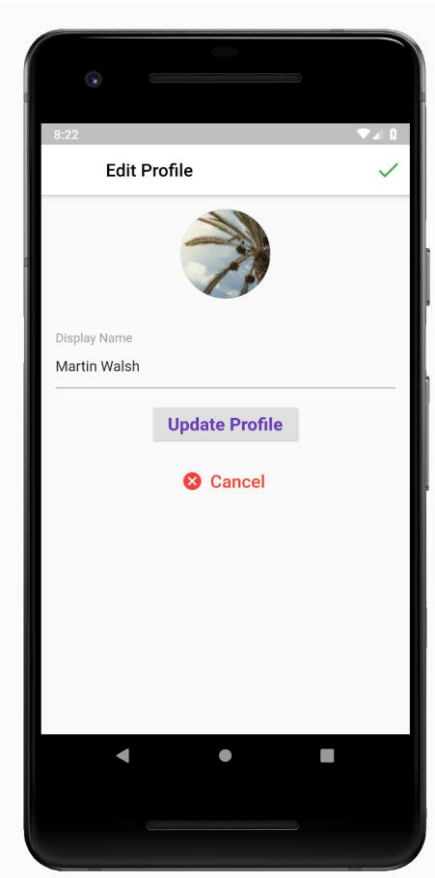


Figure 31: Profile Screen

Figure 32: Edit Profile Screen

The profile screen contains the user's posts and details (Figure 31). It is only viewable by the user and will only display their own posts. The profile screen is broken into three sections, the header, the post view options and the user's posts.

The header contains the user's avatar, which if the account was created through Google will be their Google avatar. If the account was created through email and password the image will be a standard *FindMyPet* avatar. Also in this section is the user's name alongside an edit profile and logout button.

The edit profile button will allow the user to update their display name (Figure 32) which will appear alongside any comments or posts they make. While the logout button will sign the user out of their Firebase authentication instance and redirect them to the authentication page. This also stops them from being automtacially logged back in when they next use the app as credentails must be reentered. Both buttons use one code function but accept different parameters to change the look, action and text in order to remove code redundancy.

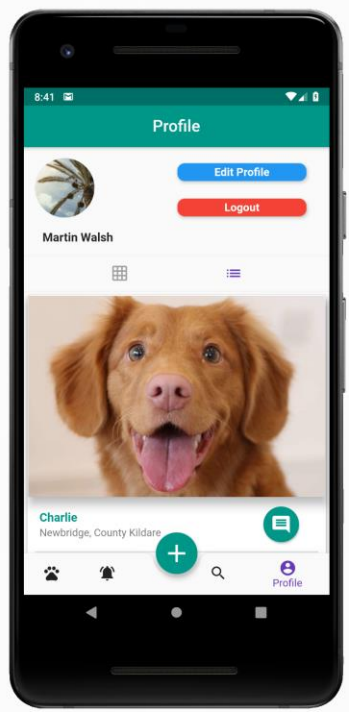


Figure 33: Profile List View

The next section of the screen is the viewing orientation of the posts. The user has the option to view their post or posts in a grid view (Figure 31) or a list view (Figure 33). List view allows the user to see what the post will look like in full screen mode. It was added to give the user more control over their experience.

The third section is where the users posts are displayed. When they are viewed on their own profile there is an option to edit or delete the post. When the user taps on the edit button the application navigates to a new page. In this page the form fields are already filled with what the user had previously entered (Figure 34). When editing is finished and the update button is tapped, there will be a pop-up on the bottom letting the user know the posts has been updated. The page will then automatically return to the profile screen with the updated information.

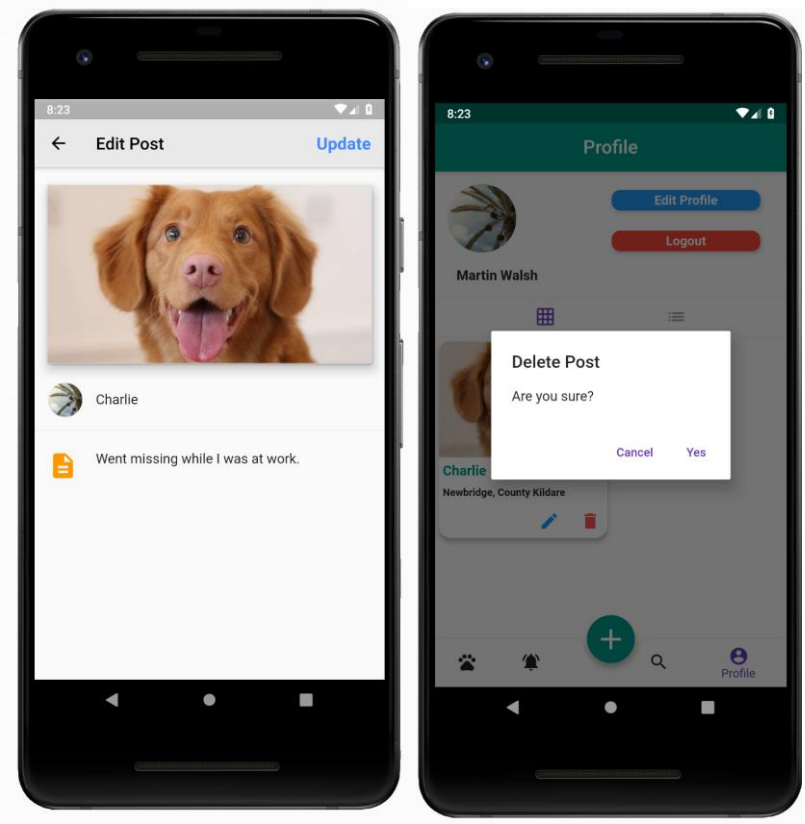


Figure 34: Edit Post *Figure 35: Delete Post*

The other option available on the post is the delete button. When pressed an alert box will appear to check if the user is sure they want to delete the post (Figure 35). If confirmed the post will be deleted from the database and disapear from the user's profile. When the post is deleted the image in storage, any activity feed items relating to the post, location information and comments relating to that post will be removed.

4. Database Layout

The Firestore database for this project contains five collections:

4.1 Comments

1. The comments collection contains a list of documents that are each identified by a post ID (Figure 36). Done in order to show the correct comments for each post.
2. Each of the documents identified by a post ID contain their own comment collection that subsequently store all the comments on that post, identified by their own unique ID (Figure 37).
3. Each one of these comments contain; comment data, display name, userID, user photo URL and a timestamp of when the comment was made.

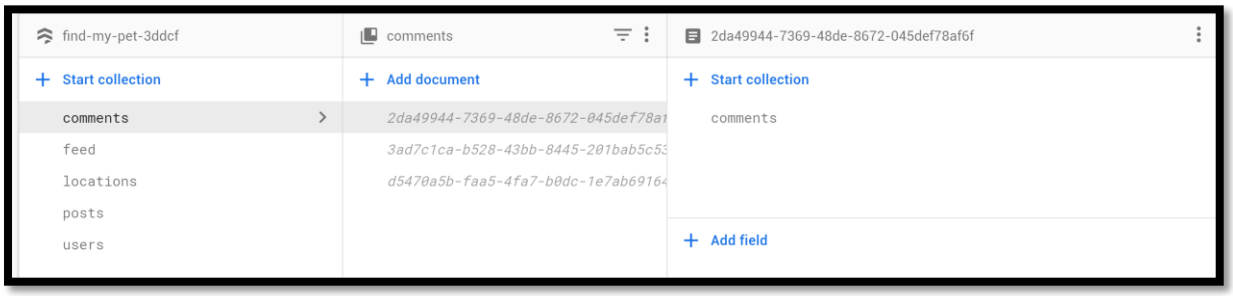


Figure 36: Database Comments Collection

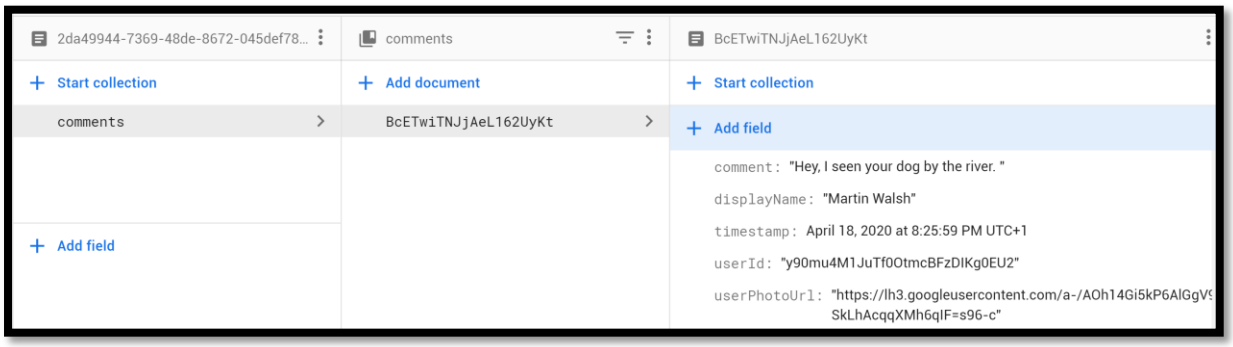


Figure 37: Database Comments Data

4.2 Activity Feed

1. The feed collection contains a list of documents that are each identified by a user's ID (Figure 38). Done in order to show the correct activity feed items for each user.
2. Each of the documents identified by the user's ID contain their own comment collection. This sub collection contains each time a user commented on one of their posts.
3. The data stored in each of these is; comment data, display name, userID, user photo URL, the posts image URL, the posts ID and a timestamp of when the feed item was created (Figure 39).

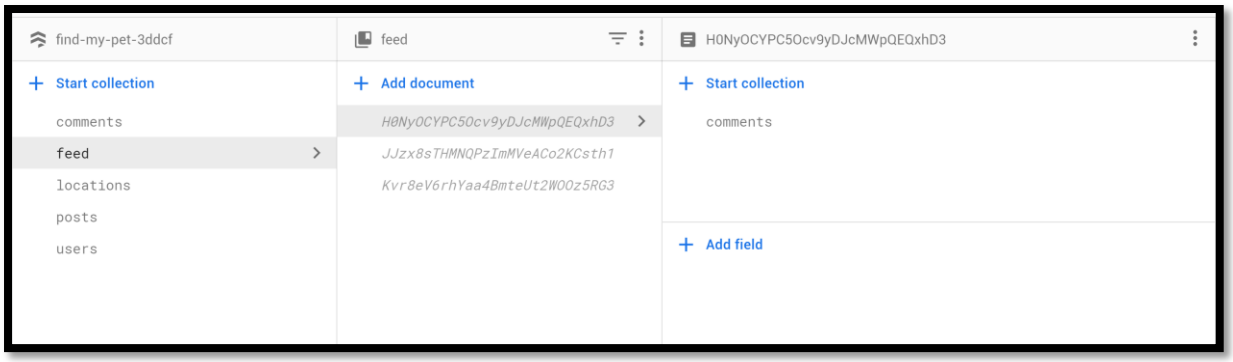


Figure 38: Database Feed Collection

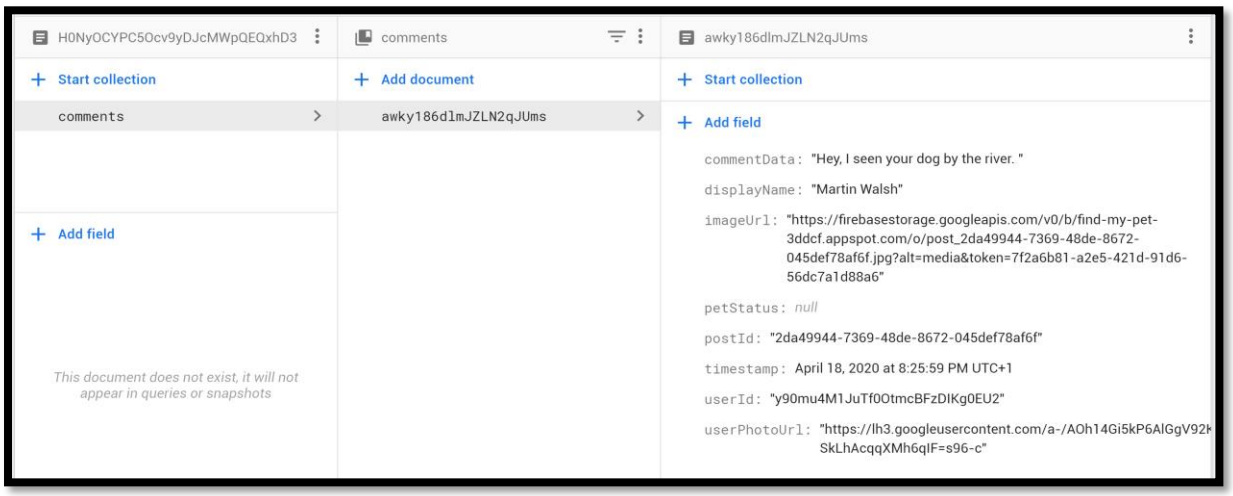


Figure 39: Database Feed Data

4.3 Locations

1. The locations collection contains a list of documents that are each identified by a Post ID. Done in order to display the correct post information in the map view.
2. Each one of these location documents contain; location name, location name in lower case, owner ID, post ID, posts title, a map containing the locations geohash and its latitude and longitude, and a timestamp of when the item was created (Figure 40).

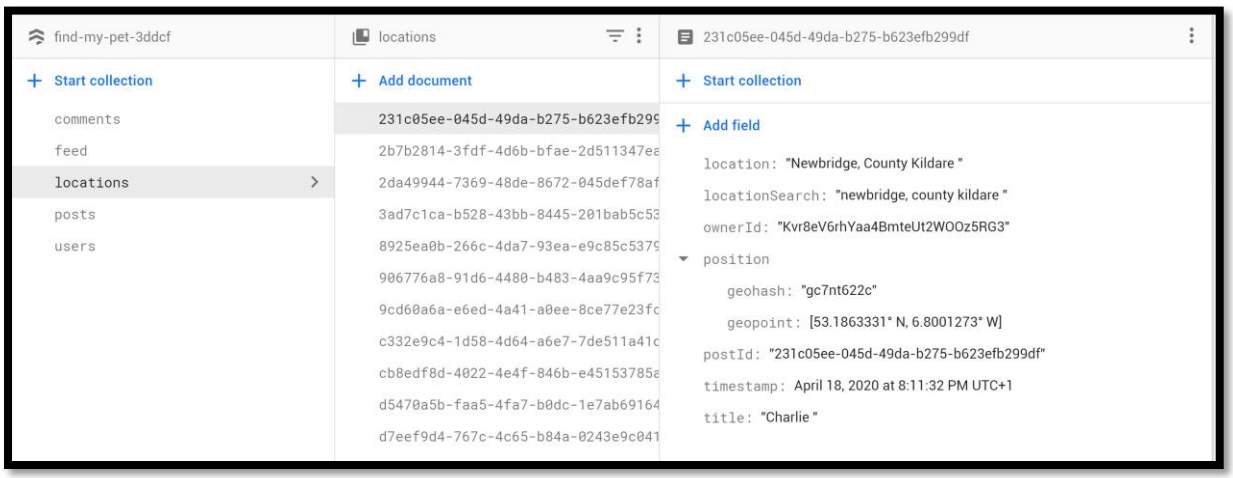


Figure 40: Database Location Collection

4.4 Posts

1. The posts collection contains a list of documents that are each identified by a Post ID.
2. Each one of these post documents contain; the posts date, description, image URL, location name, location name in lower case, owner ID, pet status, post ID, post title, a map containing the locations geohash and its latitude and longitude, and a timestamp of when the item was created (Figure 41).

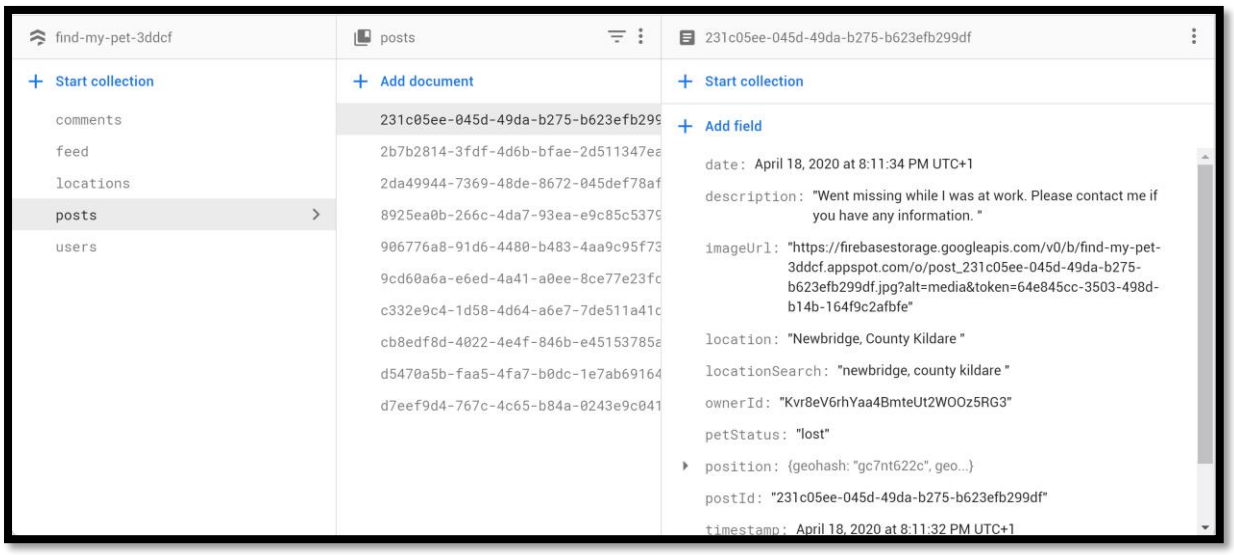


Figure 41: Posts Collection

4.5 Users

1. The user's collection contains a list of documents that are each identified by the user's ID.
2. Each one of these user documents contain; the notification token, display name, email, ID, photo URL, a map containing the locations geohash and its latitude and longitude, and a timestamp of when the item was created (Figure 42).

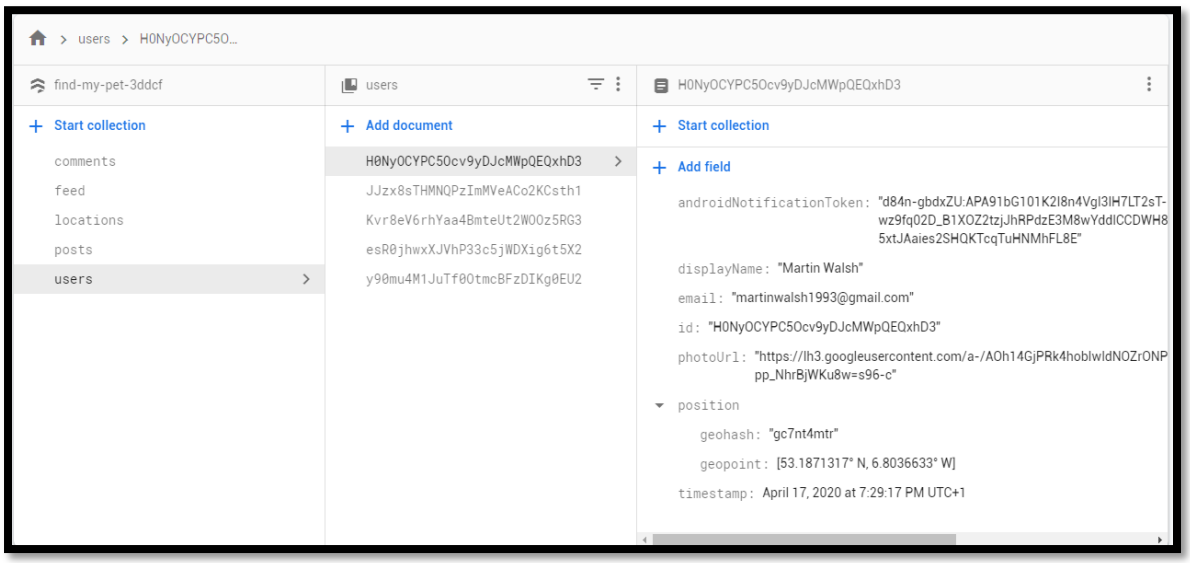


Figure 42: Users Collection

5. Location Approach

The overview screen displays posts within a radius chosen by the user (Figure 18). This screen has three tabs to display different types of posts, “Lost”, “Found” and “Spotted”. This meant the post type as well as the relevant location radius needed to be considered. The steps to return the correct data are as follows:

1. When starting the function to get the requested posts a Boolean value is set to true, which lets the application know to display a loading indicator. This is then switched to false when the posts are returned to stop the loading indicator and display the posts.
2. A list is initialised to store the values that will be returned from the database.
3. Conditional statements are carried out to determine which tab the user selected and therefore, which type of posts to return. It is then stored in a variable (Figure 43).

```
//Function to get posts based on users selected tab and radius
getPosts(int index) async {
  setState(() {
    isLoading = true;
  });

  List<Post> tempPosts = [];
  //Depending on users tab it will gather the required post types form the database
  QuerySnapshot snapshot;
  if (index == 0) {
    snapshot = await postsRef
        .orderBy("timestamp", descending: true)
        .where("petStatus", isEqualTo: 'lost')
        .getDocuments();
  } else if (index == 1) {
    snapshot = await postsRef
        .orderBy("timestamp", descending: true)
        .where("petStatus", isEqualTo: 'found')
        .getDocuments();
  } else if (index == 2) {
    snapshot = await postsRef
        .orderBy("timestamp", descending: true)
        .where("petStatus", isEqualTo: 'spotted')
        .getDocuments();
  }
```

Figure 43: Location Code Snippet 1

4. Then for each of the documents (posts) returned, the location of that post is retrieved from the data and compared against the user’s location by creating a geographical point using the latitude and longitude of each.
5. If the distance between the two location is less than the user’s chosen radius, then the ID of that document is stored in a list (Figure 44).

```
getIds(QuerySnapshot checkDistance) async {
  var pos2 = await location.getLocation();
  var point;
  checkDistance.documents.forEach((element) {
    GeoPoint pos1 = element['position']['geopoint'];
    point = geo.point(latitude: pos1.latitude, longitude: pos1.longitude);
    distance = point.distance(lat: pos2.latitude, lng: pos2.longitude);
    if (distance < radius) {
      setState(() {
        ids.add(element.documentID);
      });
    }
  });
}
```

Figure 44: Location Code Snippet 2

6. Once this is finished there is a list that contains the document IDs for each post within the user's radius.
7. These ID's are then compared against the documents returned in step 2. Wherever the ID's match, a Post object is created for that item and added to the list (Figure 45).
8. Once finished the list contains all the relevant posts to display.

```
ids.forEach((id) {
  snapshot.documents.forEach((element) {
    if (element.documentID == id) {
      tempPosts.add(Post.fromDocument(element));
    }
  });
});
```

Figure 45: Location Code Snippet 3

6. Push Notifications

Push notifications are a very important aspect of this project as it will have the biggest impact on helping to return pets home quickly and safely. Timing is of the utmost importance if a pet goes missing, in the time it takes to try make people aware that it is missing it could be too late.

With push notifications, as soon as a user creates a post about their missing a pet, all users within a 10-kilometre radius will receive a push notification. This notification will let them know that a pet has gone missing near them and display an image of that pet. This instantly makes users aware of what the pet looks like and that it is missing.

These push notifications were created using Firebase Cloud Functions, which as mentioned previously is a framework that allows you to automatically run backend code in response to events triggered by other Firebase features. In this case the functions would be triggered when a post was added to the database. The functions are not written in Dart like the rest of the application but instead written in JavaScript.

As JavaScript is the only language compatible with writing these functions, the solution for working out if the post is within a user's location radius was different. The same packages

could not be used. Also, unlike the previous location algorithm, that compares all the live posts against the user of the device's location. This algorithm starts with having the location of a particular post and then must check it against all users' locations, to see if they are within the radius.

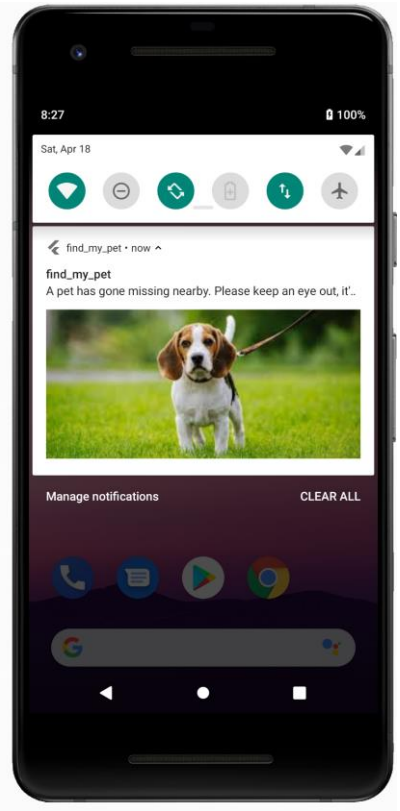


Figure 46: Lost Pet Notification

The function was created by creating a listener on the posts section of the database (Figure 41). Once a post is created the function is triggered and multiple steps take place:

1. Extracts the location data from the post, the latitude and longitude, and stores it.
2. A reference to the user's collection in the database is created, which points at where each users document can be found.
3. This list of user documents is then pulled down and an instance of each is looped over. i.e. Each loop represents a user's data.
4. With each loop, that user's location is retrieved and compared to the location of the post to determine the distance (Figure 47).
5. If conditionals are met, such as the user is within the required radius and the user is not the one who created the post. Then, before creating the notification, it is checked that the user has a notification token.
6. If all the above is correct, the notification body, token, post image, and recipient ID are stored in the notification message (Figure 48).
7. Then using Firebase admin, the notification is sent and displayed to the user (Figure 46).
8. The last three steps are repeated for each user.

```
//Function to display notifications to users within a 10km radius of the posts location
exports.onCreatePost = functions.firestore
    .document('/posts/{postId}')
    .onCreate(async (postSnap, context) => {
        //When a post is added run this function
        console.log('New Post Created', postSnap.data());

        var postLoc = postSnap.data().position.geohash;
        var userRef = admin.firestore().collection(`users`);
        userRef.get().then(snapshot => {
            //For each user, get their location and check if they are within the set radius of the post
            snapshot.forEach(doc => {
                var userLoc = doc.data().position.geohash;
                var distance = GeohashDistance.inKm(userLoc, postLoc);
                //If conditionals are met, send notification
                if (distance < 10 && postSnap.data().petStatus != "found" && postSnap.data().petStatus != "spotted" && postSnap.data().own
                    if (doc.data().androidNotificationToken) {
                        //send notification
                        sendPostNotification(doc.data().androidNotificationToken, doc.data().id, postSnap.data());
                    } else {
                        console.log("No token for user, notification could not be sent");
                    }
                }
                else {
                    console.log("Further than 15km");
                }
```

Figure 47: Lost Pet Notification Code 1

```
    function sendPostNotification(androidNotificationToken, id, postInformation) {

        const body = `A pet has gone missing nearby. Please keep an eye out, it's name is ${postInformation.title}.`;

        //Create message for push notification
        const message = {
            notification: { body: body, image: postInformation.imageUrl },
            token: androidNotificationToken,
            data: { recipient: id }
        };
        //send message with firebase admin
        admin.messaging().send(message).then(response => {
            console.log("Sent Message for post", response);
        }).catch(error => { console.log("Error", error); })
    }
});
)
```

Figure 48: Lost Pet Notification Code 2

There are three different types of notification that can be sent, each of which must have code in place for the application to handle them. Firstly, a notification for when the user is in the app, which is presented as a pop-up dialog at the bottom of the screen. The other two are displayed the same way, as seen in (Figure 46). One is when the app is running in the background of the mobile device, while the other is when the app is fully closed.

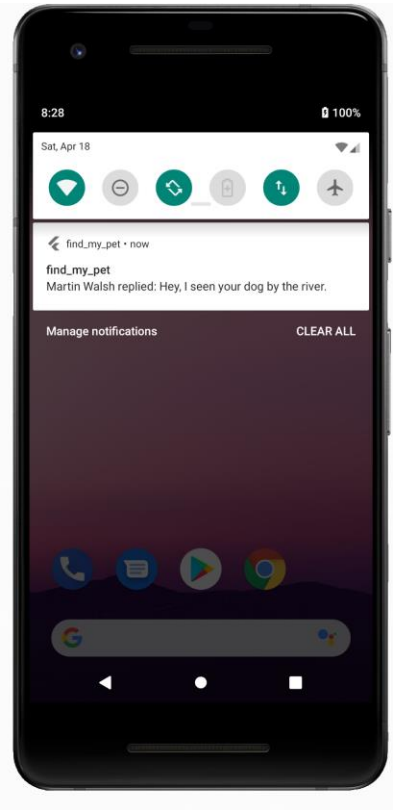


Figure 49: Comment Push Notification

There is also another type of push notification implemented into the application that is sent to a user when someone comments on a post they have made (Figure 49).This is necessary in the event of another user having information about the whereabouts of the pet. It ensures the owner of the post will be made aware even if not using the application at that time. It displays the commenter's name and the contents of the comment.

Plagiarism Declaration

Declaration

- I declare that all material in this submission e.g. thesis/essay/project/assignment is entirely my/our own work except where duly acknowledged.
- I have cited the sources of all quotations, paraphrases, summaries of information, tables, diagrams or other material; including software and other electronic media in which intellectual property rights may reside.
- I have provided a complete bibliography of all works and sources used in the preparation of this submission.
- I understand that failure to comply with the Institute's regulations governing plagiarism constitutes a serious offense.

Student Name: Martin Walsh

Student Number: C00170339

Signature: 

Date: 20/04/2020

Project Code

Screens

activity_feed.dart

```
import 'package:cached_network_image/cached_network_image.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';
import 'package:timeago/timeago.dart' as timeago;

import '../screens/home.dart';
import '../screens/post_screen.dart';
import '../widgets/progress.dart';

class ActivityFeed extends StatefulWidget {
  @override
  _ActivityFeedState createState() => _ActivityFeedState();
}

class _ActivityFeedState extends State<ActivityFeed> {
  //Function to get all feed items for the current user from firebase
  //Then add them to a list and return the list
  getActivityFeed() async {
    QuerySnapshot snapshot = await feedRef
        .document(currentUser.id)
        .collection('comments')
        .orderBy("timestamp", descending: true)
        .limit(10)
        .getDocuments();
    List<ActivityFeedItem> feedItems = [];
    snapshot.documents.forEach((doc) {
      feedItems.add(ActivityFeedItem.fromDocument(doc));
    });
    return feedItems;
  }

  //Function to clear the users notification feed
  deleteItems() async {
    QuerySnapshot snapshot = await feedRef
        .document(currentUser.id)
        .collection('comments')
        .getDocuments();
    snapshot.documents.forEach((doc) {
      doc.reference.delete();
    });
    setState(() {});
  }

  //Builds the overall activity feed screen
  @override
```

```
 Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
     backgroundColor: Theme.of(context).accentColor,
     title: Text(
      'Feed',
      style: TextStyle(
       color: Colors.white,
       fontSize: 22.0,
      ),
      overflow: TextOverflow.ellipsis,
     ),
     centerTitle: true,
     //Actions contains the clear feed button, that calls deleteItems function above
     actions: <Widget>[
      Padding(
       padding: const EdgeInsets.only(right: 20.0),
       child: CircleAvatar(
          backgroundColor: Colors.white54,
          child: IconButton(
           onPressed: () => deleteItems(),
           icon: Icon(Icons.delete),
           color: Colors.black,
          )),
      )
     ],
    ),
    //Body of the page is made up of the activity feed items
    //Created using a future builder, which will call on the data
    //in the database before trying to populate the feed items.
    body: Container(
     child: FutureBuilder(
      future: getActivityFeed(),
      builder: (context, snapshot) {
       if (!snapshot.hasData) {
        return circularProgress();
       }
       return ListView(
        children: snapshot.data,
       );
      },
     ),
    ),
   );
  }
}

//Seperate class for individual feed items that defines the structure of each item
```

```
class ActivityFeedItem extends StatelessWidget {
  final String commentData;
  final String imageUrl;
  final String postId;
  final Timestamp timestamp;
  final String type;
  final String userId;
  final String userPhotoUrl;
  final String username;
  final String petStatus;

  ActivityFeedItem({
    this.commentData,
    this.imageUrl,
    this.postId,
    this.timestamp,
    this.type,
    this.userId,
    this.userPhotoUrl,
    this.username,
    this.petStatus,
  });

  //factory describes creating an instance from a firebase document
  factory ActivityFeedItem.fromDocument(DocumentSnapshot doc) {
    return ActivityFeedItem(
      commentData: doc['commentData'],
      imageUrl: doc['imageUrl'],
      postId: doc['postId'],
      timestamp: doc['timestamp'],
      type: doc['type'],
      userId: doc['userId'],
      userPhotoUrl: doc['userPhotoUrl'],
      username: doc['displayName'],
      petStatus: doc['petStatus'],
    );
  }

  //Function to navigate to post if tapped on
  showPost(context) {
    Navigator.push(
        context,
        MaterialPageRoute(
            builder: (context) => PostScreen(
                postId: postId,
                userId: userId,
              )));
  }
```

```
//Builds the image preview for the feed item
Widget imagePreview(context) {
  return GestureDetector(
    onTap: () => showPost(context),
    child: Container(
      height: 50,
      width: MediaQuery.of(context).size.width * .2,
      child: AspectRatio(
          aspectRatio: 16 / 9,
          child: Container(
            decoration: BoxDecoration(
              image: DecorationImage(
                fit: BoxFit.cover,
                image: CachedNetworkImageProvider(imageUrl),
              ),
            ),
          )),
    ),
  );
}

//Builds and displays each activity item
//Each contains the users image, comment data, time since it happened
//the users name and an image of the post
@override
Widget build(BuildContext context) {
  return Padding(
      padding: EdgeInsets.only(bottom: 3),
      child: Container(
          color: Colors.black26,
          child: ListTile(
            title: GestureDetector(
              onTap: () => showPost(context),
              //Using RichText to highlight certain areas in notification
              child: RichText(
                overflow: TextOverflow.ellipsis,
                text: TextSpan(
                    style: TextStyle(
                      fontSize: 14,
                      color: Colors.black,
                    ),
                    children: [
                      TextSpan(
                        text: username,
                        style: TextStyle(fontWeight: FontWeight.bold),
                      ),
                      TextSpan(
```

```
                    text: ":\t\t\t$commentData",
                    style: TextStyle(fontWeight: FontWeight.bold),
                  )
                ]),
          ),
        ),
        leading: CircleAvatar(
          backgroundImage: CachedNetworkImageProvider(userPhotoUrl),
        ),
        subtitle: Text(
          timeago.format(timestamp.toDate()),
          overflow: TextOverflow.ellipsis,
        ),
        trailing: imagePreview(context),
      )));
  }
}
```

comments.dart

```
import 'package:cached_network_image/cached_network_image.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';
import 'package:timeago/timeago.dart' as timeago;

import 'package:find_my_pet/screens/home.dart';
import 'package:find_my_pet/widgets/header.dart';
import 'package:find_my_pet/widgets/progress.dart';

//Class that build the overall comment screen

class Comments extends StatefulWidget {
  final String postId;
  final String postOwnerId;
  final String postImageUrl;
  final String petStatus;
  final String displayName;

  Comments(
      {this.postId,
      this.postOwnerId,
      this.postImageUrl,
      this.petStatus,
      this.displayName});

  @override
  CommentsState createState() => CommentsState(
        postId: this.postId,
```

```
      postOwnerId: this.postOwnerId,
      postImageUrl: this.postImageUrl,
      petStatus: this.petStatus,
      displayName: this.displayName,
    );
}

class CommentsState extends State<Comments> {
  TextEditingController commentController = TextEditingController();
  final String postId;
  final String postOwnerId;
  final String postImageUrl;
  final String petStatus;
  final String displayName;

  CommentsState(
    {this.postId,
    this.postOwnerId,
    this.postImageUrl,
    this.petStatus,
    this.displayName});

  //Function to build the comments return from the database
  buildComments() {
    //Getting comments in realtime using stream
    //It actively listens to the database section you connect it to
    return StreamBuilder(
      stream: commentsRef
          .document(postId)
          .collection('comments')
          .orderBy("timestamp", descending: false)
          .snapshots(),
      builder: (context, AsyncSnapshot<QuerySnapshot> snapshot) {
        if (!snapshot.hasData) {
          return circularProgress();
        }
        List<Comment> comments = [];
        snapshot.data.documents.forEach((doc) {
          comments.add(Comment.fromDocument(doc));
        });
        return ListView(
          children: comments,
        );
      });
  }

  //Functions to add comments to to feed and comments collection in database
  addComments() {
```

```
  commentsRef.document(postId).collection("comments").add({
    "username": currentUser.username,
    "comment": commentController.text,
    "timestamp": timestamp,
    "userPhotoUrl": currentUser.photoUrl,
    "userId": currentUser.id,
    "displayName": currentUser.displayName,
  });
  if (postOwnerId != currentUser.id) {
    feedRef.document(postOwnerId).collection('comments').add({
      "type": "comment",
      "username": currentUser.username,
      "commentData": commentController.text,
      "timestamp": timestamp,
      "postId": postId,
      "userPhotoUrl": currentUser.photoUrl,
      "userId": currentUser.id,
      "imageUrl": postImageUrl,
      "petStatus": petStatus,
      "displayName": currentUser.displayName,
    });
  }
  commentController.clear();
}

//Dispay the comments screen
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: header(context, titleText: "Comments"),
    body: Column(
      children: <Widget>[
        Expanded(child: buildComments()),
        Divider(),
        ListTile(
          title: TextFormField(
            controller: commentController,
            decoration: InputDecoration(labelText: "Write a comment..."),
          ),
          trailing: OutlineButton(
            onPressed: addComments,
            borderSide: BorderSide.none,
            child: Text('Post'),
          ),
        ),
      ],
    ),
  );
```

```
  }
}

//Class to build the individual comment look and feel
//Comment data, user  image and time since posted
class Comment extends StatelessWidget {
  final String username;
  final String userId;
  final String userPhotoUrl;
  final String comment;
  final Timestamp timestamp;
  final displayName;

  const Comment({
    this.username,
    this.userId,
    this.userPhotoUrl,
    this.comment,
    this.timestamp,
    this.displayName,
  });

  factory Comment.fromDocument(DocumentSnapshot doc) {
    return Comment(
      username: doc['username'],
      userId: doc['userId'],
      userPhotoUrl: doc['userPhotoUrl'],
      comment: doc['comment'],
      timestamp: doc['timestamp'],
      displayName: doc['displayname'],
    );
  }

  @override
  Widget build(BuildContext context) {
    return Column(
      children: <Widget>[
        ListTile(
          title: Text(comment),
          leading: CircleAvatar(
            backgroundImage: CachedNetworkImageProvider(userPhotoUrl),
          ),
          subtitle: Text(timeago.format(timestamp.toDate())),
        ),
        Divider(),
      ],
    );
  }
```

```
}
```

edit_post.dart

```
import 'package:flutter/material.dart';
import 'dart:async';
import 'package:cached_network_image/cached_network_image.dart';

import 'package:find_my_pet/screens/home.dart';
import 'package:find_my_pet/widgets/custom_image.dart';
import 'package:find_my_pet/widgets/progress.dart';

//Class the build and display edit screen
class EditPost extends StatefulWidget {
  final String imageUrl;
  final String postId;
  final String description;
  final String title;

  EditPost({this.imageUrl, this.postId, this.description, this.title});

  @override
  _EditPostState createState() => _EditPostState();
}

class _EditPostState extends State<EditPost> {
  TextEditingController titleController = TextEditingController();
  TextEditingController descController = TextEditingController();
  bool isUploading = false;

  final _scaffoldKey = GlobalKey<ScaffoldState>();

  //Initialise page with title and description already filled with users previous data
  @override
  void initState() {
    super.initState();
    titleController.text = widget.title;
    descController.text = widget.description;
  }

  //Function to handle when the data is submitted
  handleSubmit(String postId) async {
    //Loading set to true while uploading to display loading spinner
    setState(() {
      isUploading = true;
    });
    //Update existing data with new data
    postsRef.document(postId).updateData(
```

```
      {"title": titleController.text, 'description': descController.text});

  setState(() {
    isUploading = false;
  });

  //Close keyboard on screen
  FocusScope.of(context).unfocus();

  //Display notification on the bottom of the screen
  SnackBar snackbar = SnackBar(
    content: Text('Post Updated!'),
  );
  _scaffoldKey.currentState.showSnackBar(snackbar);
  Timer(Duration(seconds: 1), () {
    //Then will return to previous screen
    Navigator.pop(context);
    titleController.clear();
    descController.clear();
  });
}

//Display the screen, form and buttons
@override
Widget build(BuildContext context) {
  final inputWidth = MediaQuery.of(context).size.width;
  return Scaffold(
    key: _scaffoldKey,
    appBar: AppBar(
      backgroundColor: Colors.white70,
      leading: IconButton(
          icon: Icon(Icons.arrow_back, color: Colors.black),
          onPressed: () => Navigator.of(context).pop()),
      title: Text(
        "Edit Post",
        style: TextStyle(color: Colors.black),
      ),
      actions: [
        FlatButton(
          //If uploading user cant press post again
          onPressed: isUploading ? null : () => {handleSubmit(widget.postId)},
          child: Text(
            "Update",
            style: TextStyle(
              color: Colors.blueAccent,
              fontWeight: FontWeight.bold,
              fontSize: 20.0,
            ),
```

```
        ),
      ),
    ],
  ),
  body: Container(
    width: inputWidth,
    child: ListView(
      children: <Widget>[
        isUploading ? linearProgress() : Text(''),
        Container(
          height: 220.0,
          width: inputWidth,
          child: Center(
            child: AspectRatio(
              aspectRatio: 16 / 9,
              child: Card(
                elevation: 1,
                child: Container(
                  height: MediaQuery.of(context).size.hashCode * .2,
                  width: double.infinity,
                  child: cachedNetworkImage(context, widget.imageUrl),
                ),
              ),
            ),
          ),
        ),
        Padding(
          padding: EdgeInsets.only(top: 10.0),
        ),
        //Show the users photo
        ListTile(
          leading: CircleAvatar(
            backgroundImage:
                CachedNetworkImageProvider(currentUser.photoUrl),
          ),
          title: Container(
            width: inputWidth,
            child: TextField(
              controller: titleController,
              decoration: InputDecoration(
                hintText: "Your pets name...",
                border: InputBorder.none,
              ),
            ),
          ),
        ),
        Divider(),
        ListTile(
```

```
            leading: Icon(
              Icons.description,
              color: Colors.orange,
              size: 35.0,
            ),
            title: Container(
              width: inputWidth,
              child: TextField(
                controller: descController,
                keyboardType: TextInputType.multiline,
                maxLines: 3,
                decoration: InputDecoration(
                  hintText: "Description...",
                  border: InputBorder.none,
                ),
              ),
            ),
          ),
        ],
      ),
    ),
  );
 }
}
```

edit_profile.dart

```
import 'dart:async';
import 'package:cached_network_image/cached_network_image.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import "package:flutter/material.dart";

import 'package:find_my_pet/models/user.dart';
import 'package:find_my_pet/screens/home.dart';
import 'package:find_my_pet/widgets/progress.dart';

//Class to edit a users profile
class EditProfile extends StatefulWidget {
  final String currentUserId;

  EditProfile({this.currentUserId});

  @override
  _EditProfileState createState() => _EditProfileState();
}

class _EditProfileState extends State<EditProfile> {
  //Key created to show notification when profile updated
```

```
TextEditingController displayNameController = TextEditingController();
bool isLoading = false;
User user;
bool _validDisplayName = true;

final _scaffoldKey = GlobalKey<ScaffoldState>();

//Calling function to get users data when pages initialises
@override
void initState() {
  getUser();
  super.initState();
}

//Function to get and store needed infromation
getUser() async {
  setState(() {
    isLoading = true;
  });
  DocumentSnapshot doc = await usersRef.document(widget.currentUserId).get();
  user = User.fromDocument(doc);
  displayNameController.text = user.displayName;
  setState(() {
    isLoading = false;
  });
}

//Function to build form
Column buildDisplayNameField() {
  return Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: <Widget>[
      Padding(
          padding: EdgeInsets.only(top: 12.0),
          child: Text(
            "Display Name",
            style: TextStyle(color: Colors.grey),
          )),
      TextField(
        controller: displayNameController,
        decoration: InputDecoration(
            hintText: "Update Display Name",
            errorText: _validDisplayName ? null : "Display name too short."),
      )
    ],
  );
}
```

```
//Function to update the database with new data
//And notify user when it is done
updateProfileData() async {
  SnackBar snackbar = SnackBar(
    content: Text("Profile updated!"),
  );

  setState(() {
    displayNameController.text.trim().length < 3 ||
            displayNameController.text.isEmpty
        ? _validDisplayName = false
        : _validDisplayName = true;
  });
  if (_validDisplayName = true) {
    await usersRef
        .document(widget.currentUserId)
        .updateData({"displayName": displayNameController.text.trim()});
  }
  //Close keyboard when done
  FocusScope.of(context).unfocus();
  _scaffoldKey.currentState.showSnackBar(snackbar);

  //Navigate back to previous page
  Timer(Duration(seconds: 2), () {
    Navigator.pop(context);
  });
}

//Build overall screen look and layout
@override
Widget build(BuildContext context) {
  return Scaffold(
    key: _scaffoldKey,
    appBar: AppBar(
      backgroundColor: Colors.white,
      title: Text(
        "Edit Profile",
        style: TextStyle(
          color: Colors.black,
        ),
      ),
      actions: <Widget>[
        IconButton(
          //Go back to profile page
          onPressed: () => Navigator.pop(context),
          icon: Icon(
            Icons.done,
            size: 30.0,
```

```
          color: Colors.green,
        ),
      ),
    ],
  ),
  //If loading show spinner, if not show data
  body: isLoading
      ? circularProgress()
      : ListView(
          children: <Widget>[
            Container(
              child: Column(
                children: <Widget>[
                  Padding(
                    padding: EdgeInsets.only(
                      top: 16.0,
                      bottom: 8.0,
                    ),
                    child: CircleAvatar(
                      radius: 50.0,
                      backgroundImage:
                          CachedNetworkImageProvider(user.photoUrl),
                    ),
                  ),
                  Padding(
                    padding: EdgeInsets.all(16.0),
                    child: Column(
                      children: <Widget>[
                        buildDisplayNameField(),
                      ],
                    ),
                  ),
                  RaisedButton(
                    onPressed: updateProfileData,
                    child: Text(
                      "Update Profile",
                      style: TextStyle(
                        color: Theme.of(context).primaryColor,
                        fontSize: 20.0,
                        fontWeight: FontWeight.bold,
                      ),
                    ),
                  ),
                  Padding(
                    padding: EdgeInsets.all(16.0),
                    child: FlatButton.icon(
                      onPressed: () => Navigator.pop(context),
                      icon: Icon(Icons.cancel, color: Colors.red),
```

```
                    label: Text(
                      "Cancel",
                      style: TextStyle(color: Colors.red, fontSize: 20.0),
                    ),
                  ),
                ),
              ],
            ),
          ),
        ],
      ),
    );
  }
}
```

home.dart

```
import 'dart:io';
import 'dart:math';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:firebase_messaging/firebase_messaging.dart';
import 'package:firebase_storage/firebase_storage.dart';
import 'package:google_sign_in/google_sign_in.dart';
import 'package:geoflutterfire/geoflutterfire.dart';
import 'package:geolocator/geolocator.dart';
import 'package:cloud_firestore/cloud_firestore.dart';

import 'package:find_my_pet/UI/base_widget.dart';
import 'package:find_my_pet/models/user.dart';
import 'package:find_my_pet/screens/activity_feed.dart';
import 'package:find_my_pet/screens/post_overview.dart';
import 'package:find_my_pet/screens/profile.dart';
import 'package:find_my_pet/screens/search.dart';
import 'package:find_my_pet/screens/post_form.dart';

//Authentication objects
final GoogleSignIn googleSignIn = GoogleSignIn();
final FirebaseAuth _firebaseAuth = FirebaseAuth.instance;
final StorageReference storageRef = FirebaseStorage.instance.ref();

//Firebase database references
final usersRef = Firestore.instance.collection('users');
final postsRef = Firestore.instance.collection('posts');
final commentsRef = Firestore.instance.collection('comments');
final feedRef = Firestore.instance.collection('feed');
final locationsRef = Firestore.instance.collection('locations');
```

```
//Location objects
Geoflutterfire geo = Geoflutterfire();
GeoFirePoint myLoc;

//Timestamp for database
final DateTime timestamp = DateTime.now();
//User object from data model
User currentUser;

enum AuthMode { Signup, Login }

class Home extends StatefulWidget {
  @override
  _HomeState createState() => _HomeState();
}

class _HomeState extends State<Home> {
  final FirebaseMessaging _firebaseMessaging = FirebaseMessaging();
  TextEditingController emailController = TextEditingController();
  TextEditingController passwordController = TextEditingController();
  PageController pageController;
  bool isAuth = false;
  int pageIndex = 0;

  //Global keys
  final _scaffoldKey = GlobalKey<ScaffoldState>();
  final GlobalKey<FormState> _unAuthKey = GlobalKey();
  final GlobalKey<FormState> _formKey = GlobalKey();

  //Auth status
  AuthMode _authMode = AuthMode.Login;
  //Map to store auth data
  Map<String, String> _authData = {
    'email': '',
    'password': '',
  };

  var _isLoading = false;

  //Carry out the below function when initialised
  @override
  void initState() {
    super.initState();
    pageController = PageController();
    // Detects when user signed in
    googleSignIn.onCurrentUserChanged.listen((account) {
      handleSignIn(account);
```

```
  }, onError: (err) {
    print('Error signing in: $err');
  });
  //Reauthenticate user when app is opened - Google login
  googleSignIn.signInSilently(suppressErrors: false).then((account) {
    handleSignIn(account);
  }).catchError((err) {
    print('Error signing in: $err');
  });
  //Reauthenticate user when app is opened - email login
  getUser().then((user) async {
    if (user != null) {
      DocumentSnapshot doc = await usersRef.document(user.uid).get();
      currentUser = User.fromDocument(doc);
      setState(() {
        isAuth = true;
      });
    }
  });
}

//Error function to be shown if issue loggin in
void _showErrorDialog(String message) {
  showDialog(
      context: context,
      builder: (ctx) => AlertDialog(
            title: Text('An error occured!'),
            content: Text(message),
            actions: <Widget>[
              FlatButton(
                child: Text('Okay'),
                onPressed: () {
                  Navigator.of(context).pop();
                },
              )
            ],
          ));
}

//Get user info if available
Future<FirebaseUser> getUser() async {
  return await _firebaseAuth.currentUser();
}

//Handles signin if done though email/password combo
Future<FirebaseUser> handleSignInEmail(String email, String password) async {
  //Authenticate user with firebase
  AuthResult result = await _firebaseAuth
```

```
      .signInWithEmailAndPassword(email: email, password: password)
      .catchError((signUpError) {
    //If authentication fails show error box
    showDialog<void>(
        context: context,
        builder: (BuildContext context) {
          return AlertDialog(
            title: Text('Invalid Login'),
            content: Text('The password or email is incorrect.'),
            actions: <Widget>[
              FlatButton(
                child: Text('Ok'),
                onPressed: () {
                  Navigator.of(context).pop();
                },
              ),
            ],
          );
        });
  });

  //If it doesnt fail store information
  final FirebaseUser user = result.user;

  //Ensure it contains necessary data
  assert(user != null);
  assert(await user.getIdToken() != null);

  final FirebaseUser current = await _firebaseAuth.currentUser();
  assert(user.uid == current.uid);
  print('signInEmail succeeded: $user');

  //If authenticated build user from data model
  DocumentSnapshot doc = await usersRef.document(user.uid).get();
  currentUser = User.fromDocument(doc);
  //Configure user for push notifications if not already done
  configurePushNotifications();
  setState(() {
    isAuth = true;
  });
  return user;
}

//Handle a user signing up
handleSignUp(email, password) async {
  //Create the user in firebase database
  AuthResult result = await _firebaseAuth
      .createUserWithEmailAndPassword(email: email, password: password)
```

```
      .catchError((signUpError) {
    //If email already in use display error box stating so
    if (signUpError is PlatformException) {
     if (signUpError.code == 'ERROR_EMAIL_ALREADY_IN_USE') {
      showDialog<void>(
          context: context,
          builder: (BuildContext context) {
           return AlertDialog(
            title: Text('Invalid Email'),
            content: Text('This email is already in use.'),
            actions: <Widget>[
             FlatButton(
              child: Text('Ok'),
              onPressed: () {
               Navigator.of(context).pop();
              },
             ),
            ],
           );
          });
     }
    }
   });

   //If it doesnt fail store information and check its contents
   final FirebaseUser user = result.user;
   assert(user != null);
   assert(await user.getIdToken() != null);
   await getUserLocation();

   //Store user info in database as well as authentication platform
   await usersRef.document(user.uid).setData({
    "id": user.uid,
    "photoUrl":
      "https://firebasestorage.googleapis.com/v0/b/find-my-pet-
3ddcf.appspot.com/o/pet.png?alt=media&token=4e018ced-3681-4108-82fd-e55758818f24",
    "email": user.email,
    "displayName": user.displayName,
    "timestamp": timestamp,
    "position": myLoc.data,
   });
   //Create current user using data obtained and data model
   var doc = await usersRef.document(user.uid).get();
   currentUser = User.fromDocument(doc);
   showDialog<void>(
     //Notify user when account created
     context: context,
     builder: (BuildContext context) {
```

```
        return AlertDialog(
          title: Text('Account Created'),
          content: Text('Please login to proceed.'),
          actions: <Widget>[
            FlatButton(
              child: Text('Ok'),
              onPressed: () {
                Navigator.of(context).pop();
              },
            ),
          ],
        );
      });
}

//Handles the form being submitted
//Validate, error checks, calls necessary functions and switches auth mode if authenticated
Future<void> _submit() async {
  if (!_formKey.currentState.validate()) {
    // Invalid
    return;
  }
  //Saves the inputs
  _formKey.currentState.save();
  setState(() {
    _isLoading = true;
  });
  try {
    if (_authMode == AuthMode.Login) {
      handleSignInEmail(emailController.text, passwordController.text);
    } else {
      handleSignUp(emailController.text, passwordController.text);
    }
    //On: checking for specific type of exception
    //This error is thrown if data validation fails
  } on HttpException catch (error) {
    var errorMessage = 'Authentication failed';
    if (error.toString().contains('EMAIL_EXISTS')) {
      errorMessage = 'This email address is already in use.';
    } else if (error.toString().contains('INVALIS_EMAIL')) {
      errorMessage = 'This is not a valid email address.';
    } else if (error.toString().contains('WEAK_PASSWORD')) {
      errorMessage = 'This password is too weak.';
    } else if (error.toString().contains('EMAIL_NOT_FOUND')) {
      errorMessage = 'Could not find a user with that email.';
    } else if (error.toString().contains('INVALID_PASSWORD')) {
      errorMessage = 'Invalid password.';
    }
```

```
    _showErrorDialog(errorMessage);
  } catch (error) {
    const errorMessage = 'Could not authenticate you. Please try again.';
    _showErrorDialog(errorMessage);
  }

  setState(() {
    _isLoading = false;
  });
}

void _switchAuthMode() {
  if (_authMode == AuthMode.Login) {
    setState(() {
      _authMode = AuthMode.Signup;
    });
  } else {
    setState(() {
      _authMode = AuthMode.Login;
    });
  }
}

//Hangles sign in if done through google
handleSignIn(GoogleSignInAccount account) async {
  if (account != null) {
    //Create user in database
    await createUserInFirestore(account);
    print('User signed in!: $account');
    setState(() {
      isAuth = true;
    });
    //Configure push notifications if not already done
    configurePushNotifications();
  } else {
    setState(() {
      isAuth = false;
    });
  }
}

//Get the users current location
getUserLocation() async {
  Position position = await Geolocator()
      .getCurrentPosition(desiredAccuracy: LocationAccuracy.high);
  setState(() {
    myLoc =
        geo.point(latitude: position.latitude, longitude: position.longitude);
```

```
  });
}

//Configure push notifications
configurePushNotifications() {
  var userId = currentUser.id;
  //Add notification token to users data in database
  _firebaseMessaging.getToken().then((token) {
    DocumentReference docRef = usersRef.document(userId);
    docRef.updateData({"androidNotificationToken": token});
  });

  //Configure notifications for the app
  _firebaseMessaging.configure(
    //display notifications when app is off
    onLaunch: (Map<String, dynamic> message) async {
      final String recipientId = message['data']['recipient'];
      final String body = message['notification']['body'];

      if (recipientId == userId) {
        SnackBar snackbar = SnackBar(
            content: Text(
          body,
          overflow: TextOverflow.ellipsis,
        ));
        _scaffoldKey.currentState.showSnackBar(snackbar);
      }
    },
    //display notifications when app is in background
    onResume: (Map<String, dynamic> message) async {
      final String recipientId = message['data']['recipient'];
      final String body = message['notification']['body'];

      if (recipientId == userId) {
        SnackBar snackbar = SnackBar(
            content: Text(
          body,
          overflow: TextOverflow.ellipsis,
        ));
        _scaffoldKey.currentState.showSnackBar(snackbar);
      }
    },

    //display notifications when in the app
    onMessage: (Map<String, dynamic> message) async {
      final String recipientId = message['data']['recipient'];
      final String body = message['notification']['body'];
```

```
      if (recipientId == userId) {
        SnackBar snackbar = SnackBar(
            content: Text(
          body,
          overflow: TextOverflow.ellipsis,
        ));
        _scaffoldKey.currentState.showSnackBar(snackbar);
      }
    },
  );
}

//Create user in the database
createUserInFirestore(GoogleSignInAccount account) async {
  //Get and auth details
  final GoogleSignInAccount googleUser = await googleSignIn.signIn();
  final GoogleSignInAuthentication googleAuth =
      await googleUser.authentication;
  final AuthCredential credentials = GoogleAuthProvider.getCredential(
      idToken: googleAuth.idToken, accessToken: googleAuth.accessToken);

  getUserLocation();
  FirebaseUser userDetails =
      (await _firebaseAuth.signInWithCredential(credentials)).user;

  //check if user exists in users collection in db (according to their id)
  DocumentSnapshot doc = await usersRef.document(userDetails.uid).get();
  if (!doc.exists) {
    //Make a new user doc in users collection if it doesnt exist
    usersRef.document(userDetails.uid).setData({
      "id": userDetails.uid,
      "photoUrl": userDetails.photoUrl,
      "email": userDetails.email,
      "displayName": userDetails.displayName,
      "timestamp": timestamp,
      "position": myLoc.data,
    });
    //Contains users data from database
    doc = await usersRef.document(userDetails.uid).get();
  }

  //If data does exist create current user using User data model
  currentUser = User.fromDocument(doc);
}

login() {
  googleSignIn.signIn();
}
```

```
logout() {
  googleSignIn.signOut();
}

//Keep track of users page, i.e bottom navigation
onPageChanged(int pageIndex) {
  setState(() {
    this.pageIndex = pageIndex;
  });
}

//Change users page when tapped
onTap(int pageIndex) {
  pageController.animateToPage(
    pageIndex,
    duration: Duration(
      milliseconds: 200,
    ),
    curve: Curves.easeInOut,
  );
}

//Dispose of controller to prevent memory leak
@override
void dispose() {
  pageController.dispose();
  super.dispose();
}

//Build the screen that is shown when user is authenticated
Scaffold buildAuthScreen() {
  return Scaffold(
    key: _scaffoldKey, //for notifications
    resizeToAvoidBottomPadding: false,
    body: PageView(
      //The different page options
      children: <Widget>[
        PostsOverview(),
        ActivityFeed(),
        Upload(
          currentUser: currentUser,
        ),
        Search(),
        //? means only get ID if not null
        Profile(profileId: currentUser?.id),
      ],
      controller: pageController,
```

```
      onPageChanged: onPageChanged,
      physics: NeverScrollableScrollPhysics(),
    ),

    //Floating action button for adding a post
    floatingActionButton: FloatingActionButton(
      heroTag: 'btn1',
      onPressed: () => onTap(2),
      child: Icon(
        Icons.add,
        size: 40,
      ),
    ),
    //Bottom navigation for the different pages
    bottomNavigationBar: BottomAppBar(
      notchMargin: 5,
      shape: CircularNotchedRectangle(),
      clipBehavior: Clip.antiAlias,
      child: BottomNavigationBar(
        currentIndex: pageIndex,
        onTap: onTap,
        selectedItemColor: Theme.of(context).primaryColor,
        unselectedItemColor: Colors.black87,
        //Each page name and icon, keeps an index
        items: const <BottomNavigationBarItem>[
          BottomNavigationBarItem(
              icon: Icon(Icons.pets), title: Text('Pets')),
          BottomNavigationBarItem(
              icon: Icon(Icons.notifications_active),
              title: Text('Notifications')),
          BottomNavigationBarItem(
              icon: Icon(
                Icons.search,
                color: Colors.transparent,
              ),
              title: Text('')),
          BottomNavigationBarItem(
              icon: Icon(Icons.search), title: Text('Search')),
          BottomNavigationBarItem(
              icon: Icon(Icons.account_circle), title: Text('Profile')),
        ],
      ),
    ),
    floatingActionButtonLocation: FloatingActionButtonLocation.centerDocked,
  );
}

//Build the screen that is shown when user is unauthenticated
```

```
Scaffold buildUnAuthScreen() {
  final deviceSize = MediaQuery.of(context).size;
  return Scaffold(
    key: _unAuthKey,
    //Stack to place widgets on top of each other
    body: Stack(
      children: <Widget>[
        Container(
          decoration: BoxDecoration(
            //Gradient of colours for container
            gradient: LinearGradient(
              colors: [
                Color.fromRGBO(67, 206, 162, 1).withOpacity(0.5),
                Color.fromRGBO(24, 90, 157, 1).withOpacity(0.9),
              ],
              begin: Alignment.topLeft,
              end: Alignment.bottomRight,
              stops: [0, 1],
            ),
          ),
        ),
        SingleChildScrollView(
          child: Container(
            height: deviceSize.height,
            width: deviceSize.width,
            child: Column(
              mainAxisAlignment: MainAxisAlignment.center,
              crossAxisAlignment: CrossAxisAlignment.center,
              children: <Widget>[
                Flexible(
                  child: Container(
                    width: deviceSize.width * .9,
                    margin: EdgeInsets.only(bottom: 20.0),
                    padding:
                        EdgeInsets.symmetric(vertical: 8.0, horizontal: 44.0),
                    //Transform: allows you to change how container is presented
                    //Matrix4 describes transformation of a container, e.g. rotation, scaling
                    //Rotainz: change z axis, depth into device
                    //translate changes the offset
                    //.. alows you to return the type of the previous statement instead of the translates default (voi
d)
                    transform: Matrix4.rotationZ(-8 * pi / 180)
                      ..translate(-5.0),
                    decoration: BoxDecoration(
                      borderRadius: BorderRadius.circular(20),
                      color: Colors.white70,
                      boxShadow: [
                        BoxShadow(
```

```
                    blurRadius: 8,
                    color: Colors.black26,
                    offset: Offset(0, 2),
                  )
                ],
              ),
              height: deviceSize.height * .2,
              child: FittedBox(
                fit: BoxFit.fitWidth,
                child: Text(
                  'FindMyPet',
                  style: TextStyle(
                    color: Theme.of(context).accentColor,
                    fontSize: 50,
                    fontFamily: 'Anton',
                    fontWeight: FontWeight.normal,
                  ),
                ),
              ),
            ),
          ),
          Flexible(
            flex: deviceSize.width > 600 ? 2 : 1,
            //Authentication form
            child: buildAuthCard(),
          ),
        ],
      ),
    ),
  ),
 );
}

//Auth card as part of authentication screen
//Conditional checks thoughout change look based on auth mode (sign in or sign up)
buildAuthCard() {
 final deviceSize = MediaQuery.of(context).size;
 return Card(
  shape: RoundedRectangleBorder(
   borderRadius: BorderRadius.circular(10.0),
  ),
  elevation: 8.0,
  child: Container(
   //Need more height depending on auth mode, for confirm password
   height: _authMode == AuthMode.Signup
     ? deviceSize.height * .7
```

```
      : deviceSize.height * .5,
  constraints: BoxConstraints(
      minHeight: _authMode == AuthMode.Signup
          ? deviceSize.height * .7
          : deviceSize.height * .5),
  width: deviceSize.width * 0.75,
  padding: EdgeInsets.all(16.0),
  child: Form(
    key: _formKey,
    child: SingleChildScrollView(
      child: Column(
        children: <Widget>[
          TextFormField(
            controller: emailController,
            decoration: InputDecoration(labelText: 'E-Mail'),
            keyboardType: TextInputType.emailAddress,
            validator: (value) {
              if (value.isEmpty || !value.contains('@')) {
                return 'Invalid email!';
              }
              return null;
            },
            onSaved: (value) {
              _authData['email'] = value;
            },
          ),
          TextFormField(
            decoration: InputDecoration(labelText: 'Password'),
            //So password isnt shown
            obscureText: true,
            //Controller is used in conjunction with last from filed, only in signup mode
            controller: passwordController,
            validator: (value) {
              if (value.isEmpty || value.length < 5) {
                return 'Password is too short!';
              }
              return null;
            },
            onSaved: (value) {
              _authData['password'] = value;
            },
          ),
          if (_authMode == AuthMode.Signup)
            //Confrim password
            TextFormField(
              enabled: _authMode == AuthMode.Signup,
              decoration: InputDecoration(labelText: 'Confirm Password'),
              obscureText: true,
```

```
      validator: _authMode == AuthMode.Signup
          ? (value) {
              //If the value in this form field matches the value in previous form field
              if (value != passwordController.text) {
                return 'Passwords do not match!';
              }
              return null;
            }
          : null,
    ),
  const SizedBox(
    height: 20,
  ),
  if (_isLoading)
    CircularProgressIndicator()
  else
    //This button calls the submit method
    RaisedButton(
      child: Text(
          _authMode == AuthMode.Login ? 'SIGN IN' : 'SIGN UP'),
      onPressed: _submit,
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(30),
      ),
      padding:
          EdgeInsets.symmetric(horizontal: 30.0, vertical: 8.0),
      color: Theme.of(context).primaryColor,
      textColor: Theme.of(context).primaryTextTheme.button.color,
    ),
  if (_authMode == AuthMode.Login)
    //Login in with google option
    GestureDetector(
      onTap: login,
      child: Padding(
        padding: const EdgeInsets.all(8.0),
        child: Container(
          width: MediaQuery.of(context).size.width * .4,
          height: MediaQuery.of(context).size.height * .05,
          decoration: BoxDecoration(
            image: DecorationImage(
              image: AssetImage(
                'assets/images/google_signin_button.png',
              ),
              alignment: Alignment.center,
              fit: BoxFit.cover,
            ),
          ),
        ),
```

```
    ),
  ),
//Switch betweens auth modes
FlatButton(
  child: Text(
      '${_authMode == AuthMode.Login ? 'SIGNUP' : 'LOGIN'} INSTEAD'),
  onPressed: _switchAuthMode,
  padding: EdgeInsets.symmetric(horizontal: 30.0, vertical: 4),
  //Reduces amount of surface to tap the button
  materialTapTargetSize: MaterialTapTargetSize.shrinkWrap,
  textColor: Theme.of(context).primaryColor,
),
//Change password option
if (_authMode == AuthMode.Login)
  FlatButton(
    child: Text('Change Password'),
    onPressed: () async {
      if (emailController.text == "") {
        setState(() {
          showDialog<void>(
              context: context,
              builder: (BuildContext context) {
                //Alert box to tell user to enter email in order to change password
                return AlertDialog(
                  title: Text('Change Password'),
                  content:
                      Text('Please enter a valid emai first.'),
                  actions: <Widget>[
                    FlatButton(
                      child: Text('Ok'),
                      onPressed: () {
                        Navigator.of(context).pop();
                      },
                    ),
                  ],
                );
              });
        });
      }
      //Send an email to users email to change password
      await _firebaseAuth
          .sendPasswordResetEmail(email: emailController.text)
          .then((_) {
        showDialog<void>(
            context: context,
            builder: (BuildContext context) {
              return AlertDialog(
                title: Text('Change Password'),
```

```
                        content: Text(
                            'An email to change your password has been sent to ${emailController.text}.'),
                        actions: <Widget>[
                          FlatButton(
                            child: Text('Ok'),
                            onPressed: () {
                              Navigator.of(context).pop();
                            },
                          ),
                        ],
                      );
                    });
                  });
                },
                padding:
                    EdgeInsets.symmetric(horizontal: 30.0, vertical: 4),
                //Reduces amount of surface to tap the button
                materialTapTargetSize: MaterialTapTargetSize.shrinkWrap,
                textColor: Theme.of(context).primaryColor,
              ),
            ],
          ),
        ),
      ),
    ),
  );
 }

 //Overall structure and decides which screen to build
 @override
 Widget build(BuildContext context) {
   return isAuth
       ? buildAuthScreen()
       : BaseWidget(builder: (context, sizingInfo) {
           return buildUnAuthScreen();
         });
 }
}
```

post_form.dart

```
import 'package:flutter/material.dart';
import 'dart:io';
import 'package:cached_network_image/cached_network_image.dart';
import 'package:firebase_storage/firebase_storage.dart';
import 'package:flutter_google_places/flutter_google_places.dart';
import 'package:geoflutterfire/geoflutterfire.dart';
import 'package:geolocator/geolocator.dart';
```

```dart
import 'package:google_maps_webservice/places.dart';
import 'package:image_picker/image_picker.dart';
import 'package:intl/intl.dart';
import 'package:nice_button/NiceButton.dart';
import 'package:path_provider/path_provider.dart';
import 'package:image/image.dart' as Img;
import 'package:uuid/uuid.dart';

import 'package:find_my_pet/screens/home.dart';
import 'package:find_my_pet/widgets/progress.dart';
import 'package:find_my_pet/models/user.dart';

//Create upload form and logic
class Upload extends StatefulWidget {
  final User currentUser;

  Upload({this.currentUser});

  @override
  _UploadState createState() => _UploadState();
}

//AutomaticKeepAliveClientMixin to keep page state if user navigates to new page
class _UploadState extends State<Upload>
    with AutomaticKeepAliveClientMixin<Upload> {
  //Text controllers to store from inputs
  TextEditingController titleController = TextEditingController();
  TextEditingController dateController = TextEditingController();
  TextEditingController descController = TextEditingController();
  TextEditingController locationController = TextEditingController();
  ScrollController _controller =
      ScrollController(); //Scroll controller to move to top of page when form is submitted

  //Location variables
  Geoflutterfire geo = Geoflutterfire();
  GeoFirePoint myLoc;
  DateTime _selectDate = DateTime.now();
  bool _selectIsFound = false;
  String _petStatus;
  bool isUploading = false;
  File file; //Stores user image
  final _scaffoldKey = GlobalKey<ScaffoldState>();

  //Auto generate unique id
  String postId = Uuid().v4();
  final _form = GlobalKey<FormState>();

  //Store the photo the user takes
```

```
handleTakePhoto() async {
  Navigator.pop(context);
  File file = await ImagePicker.pickImage(
    source: ImageSource.camera,
    maxHeight: 675,
    maxWidth: 960,
  );
  setState(() {
    this.file = file;
  });
}

//Stores the image the user chooses from gallery
handleChooseFromGallery() async {
  Navigator.pop(context);
  File file = await ImagePicker.pickImage(source: ImageSource.gallery);
  setState(() {
    this.file = file;
  });
}

//Dialog to display users options for choosing a photo
selectImage(parentContext) {
  return showDialog(
    context: parentContext,
    builder: (context) {
      return SimpleDialog(
        title: Center(child: Text("Choose a photo of the pet.")),
        titlePadding: EdgeInsets.only(top: 8, bottom: 8),
        children: <Widget>[
          SimpleDialogOption(
              child: Text("Photo with Camera"), onPressed: handleTakePhoto),
          SimpleDialogOption(
              child: Text("Image from Gallery"),
              onPressed: handleChooseFromGallery),
          SimpleDialogOption(
            child: Text("Cancel"),
            onPressed: () => Navigator.pop(context),
          )
        ],
      );
    },
  );
}

//Dispalys the initial screen which presents the user with post types
//Lost, found or spotted a pet. Also a description for each type
Container buildSplashScreen() {
```

```
var firstColor = Theme.of(context).primaryColor,
    secondColor = Theme.of(context).accentColor;
return Container(
  color: Theme.of(context).accentColor.withOpacity(0.3),
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.center,
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,
    children: <Widget>[
      Column(
        children: <Widget>[
          NiceButton(
            elevation: 7,
            radius: 40,
            padding: const EdgeInsets.all(15),
            text: "Missing Pet",
            gradientColors: [secondColor, firstColor],
            onPressed: () {
              setState(() {
                _petStatus = 'lost';
                selectImage(context);
              });
            },
            background: null,
          ),
          Padding(
            padding: const EdgeInsets.only(top: 8.0),
            child: Container(
              width: MediaQuery.of(context).size.width * .5,
              child: RichText(
                textAlign: TextAlign.center,
                textWidthBasis: TextWidthBasis.parent,
                text: TextSpan(
                    text: 'If you have lost your pet.',
                    style: TextStyle(
                      color: firstColor,
                      fontWeight: FontWeight.bold,
                    )),
              ),
            ),
          ),
        ],
      ),
      Column(
        crossAxisAlignment: CrossAxisAlignment.center,
        children: <Widget>[
          NiceButton(
            elevation: 7,
            radius: 40,
```

```
          padding: const EdgeInsets.all(15),
          text: "Found Pet",
          gradientColors: [secondColor, firstColor],
          onPressed: () {
            setState(() {
              _petStatus = 'found';
              selectImage(context);
            });
          },
          background: null,
        ),
        Padding(
          padding: const EdgeInsets.only(top: 8.0),
          child: Container(
            width: MediaQuery.of(context).size.width * .5,
            child: RichText(
              textAlign: TextAlign.center,
              textWidthBasis: TextWidthBasis.parent,
              maxLines: 2,
              text: TextSpan(
                  text:
                      'If you have found a pet and are looking for the owner.',
                  style: TextStyle(
                    color: firstColor,
                    fontWeight: FontWeight.bold,
                  )),
            ),
          ),
        ),
      ],
    ),
    Column(
      crossAxisAlignment: CrossAxisAlignment.center,
      children: <Widget>[
        NiceButton(
          elevation: 7,
          radius: 40,
          padding: const EdgeInsets.all(15),
          text: "Spotted Pet",
          gradientColors: [secondColor, firstColor],
          onPressed: () {
            setState(() {
              _petStatus = 'spotted';
              selectImage(context);
            });
          },
          background: null,
        ),
```

```
          Padding(
            padding: const EdgeInsets.only(top: 8.0),
            child: Container(
              width: MediaQuery.of(context).size.width * .5,
              child: RichText(
                textAlign: TextAlign.center,
                textWidthBasis: TextWidthBasis.parent,
                text: TextSpan(
                    text: 'If you spot a lost pet.',
                    style: TextStyle(
                      color: firstColor,
                      fontWeight: FontWeight.bold,
                    )),
              ),
            ),
          ),
        ],
      ),
    ],
  ),
 );
}

//Clear the image stored
clearImage() {
  setState(() {
    file = null;
  });
}

//Compress the image
compressImage() async {
  //Path to the temporary directory on the device
  final tempDir = await getTemporaryDirectory();
  final path = tempDir.path;
  Img.Image imageFile = Img.decodeImage(file.readAsBytesSync());
  final compressedImageFile = File('$path/img_$postId.jpg')
    ..writeAsBytesSync(Img.encodeJpg(imageFile, quality: 80));
  setState(() {
    file = compressedImageFile;
  });
}

//Upload the compressed image to firebase storage and return url
Future<String> uploadImage(imageFile) async {
  StorageUploadTask uploadImg =
      storageRef.child("post_$postId.jpg").putFile(imageFile);
  //Returns a snapshot containing data
```

```
  StorageTaskSnapshot storageSnap = await uploadImg.onComplete;
  String downloadUrl = await storageSnap.ref.getDownloadURL();
  return downloadUrl;
}

//Clear all text controllers to prevent memory leak
clearcontrollers() {
  titleController.clear();
  dateController.clear();
  descController.clear();
  locationController.clear();
}

//Create the post in the database
//Both in post and location collection
createPostInFirestore(
    {String imageUrl,
    String location,
    String description,
    String title,
    String date}) {
  postsRef.document(postId).setData({
    "postId": postId,
    "ownerId": currentUser.id,
    "username": currentUser.displayName,
    "imageUrl": imageUrl,
    "location": location,
    "locationSearch": location.toLowerCase(),
    "description": description,
    "title": title,
    "date": _selectDate,
    "timestamp": timestamp,
    "isFound": _selectIsFound,
    "position": myLoc.data,
    "petStatus": _petStatus,
  });
  locationsRef.document(postId).setData({
    "postId": postId,
    "location": location,
    "locationSearch": location.toLowerCase(),
    "title": title,
    "timestamp": timestamp,
    "position": myLoc.data,
    "ownerId": currentUser.id,
  });
}

//Displays a calendar to the screen to pick a date
```

```
void _presentDatePicker() {
  //context is global in this state class, doesnt need to be passed in
  showDatePicker(
    context: context,
    initialDate: DateTime.now(),
    firstDate: DateTime(2020),
    lastDate: DateTime.now(),
  ).then((pickedDate) {
    //Means user didnt pick a date
    if (pickedDate == null) {
      return;
    }
    setState(() {
      _selectDate = pickedDate;
    });
  });
}

//Carries out the functions when form submitted
handleSubmit() async {
  //Checks the validity of the form
  final isValid = _form.currentState.validate();
  if (!isValid) {
    return;
  }

  setState(() {
    isUploading = true;
  });
  await compressImage();
  String imageUrl = await uploadImage(file);
  print(_petStatus);
  createPostInFirestore(
    imageUrl: imageUrl,
    location: locationController.text,
    description: descController.text,
    title: titleController.text,
    date: dateController.text,
  );
  clearcontrollers();
  setState(() {
    file = null;
    isUploading = false;
    //Ensure new unique ID for every post
    postId = Uuid().v4();
  });
}
```

```
//Build the post form
Scaffold buildForm() {
  //Get screen width
  final inputWidth = MediaQuery.of(context).size.width;
  return Scaffold(
    key: _scaffoldKey,
    appBar: AppBar(
      backgroundColor: Colors.white70,
      leading: IconButton(
          icon: Icon(Icons.arrow_back, color: Colors.black),
          onPressed: clearImage),
      title: Text(
        "Create Post",
        style: TextStyle(color: Colors.black),
      ),
      actions: [
        FlatButton(
          //If uploading user cant press post again
          onPressed: isUploading
              ? null
              : () => {
                    //When pressed close the keybaord and scroll to top of screen
                    FocusScope.of(context).unfocus(),
                    _controller.jumpTo(_controller.position.minScrollExtent),
                    handleSubmit()
                  },
          child: Text(
            "Post",
            style: TextStyle(
              color: Colors.blueAccent,
              fontWeight: FontWeight.bold,
              fontSize: 20.0,
            ),
          ),
        ),
      ],
    ),
    body: Container(
      width: inputWidth,
      child: Form(
        key: _form,
        child: ListView(
          controller: _controller,
          children: <Widget>[
            isUploading ? linearProgress() : Text(''),
            Container(
              height: 220.0,
              width: inputWidth,
```

```
          child: Center(
            child: AspectRatio(
              aspectRatio: 16 / 9,
              child: Card(
                elevation: 1,
                //Display the users picture
                child: Container(
                  decoration: BoxDecoration(
                    image: DecorationImage(
                      fit: BoxFit.cover,
                      image: FileImage(file),
                    ),
                  ),
                ),
              ),
            ),
          ),
        ),
        Padding(
          padding: EdgeInsets.only(top: 10.0),
        ),
        ListTile(
          leading: CircleAvatar(
            backgroundImage:
                CachedNetworkImageProvider(currentUser.photoUrl),
          ),
          title: Container(
            width: inputWidth,
            child: TextFormField(
              validator: (value) {
                if (value.isEmpty) {
                  return "Please provide a name.";
                }
                return null;
              },
              controller: titleController,
              decoration: InputDecoration(
                hintText: "Your pets name...",
                border: InputBorder.none,
              ),
            ),
          ),
        ),
        Divider(),
        ListTile(
          leading: Icon(
            Icons.description,
            color: Colors.orange,
```

```
            size: 35.0,
          ),
          title: Container(
            width: inputWidth,
            child: TextFormField(
              validator: (value) {
                if (value.isEmpty) {
                  return "Please provide a description.";
                }
                return null;
              },
              controller: descController,
              keyboardType: TextInputType.multiline,
              maxLines: 3,
              decoration: InputDecoration(
                hintText: "Description...",
                border: InputBorder.none,
              ),
            ),
          ),
        ),
        Divider(),
        ListTile(
          leading: Icon(
            Icons.date_range,
            color: Colors.orange,
            size: 35.0,
          ),
          title: TextFormField(
            controller: dateController,
            enabled: false,
            decoration: InputDecoration(
              enabled: false,
              //Display the selected date
              labelText: '${DateFormat.yMd().format(_selectDate)}',
              hintText: "What date...",
              border: InputBorder.none,
            ),
          ),
          trailing: Container(
            child: FittedBox(
              fit: BoxFit.contain,
              child: Padding(
                padding: const EdgeInsets.only(
                  top: 8.0,
                  left: 8.0,
                ),
                child: FlatButton(
```

```
            textColor: Theme.of(context).primaryColor,
            child: Text(
              'Choose date',
              style: TextStyle(fontWeight: FontWeight.bold),
            ),
            onPressed: _presentDatePicker),
        ),
      ),
    ),
  ),
  Divider(),
  ListTile(
    leading: Icon(
      Icons.insert_comment,
      color: Colors.orange,
      size: 35.0,
    ),
    title: TextFormField(
      validator: (value) {
        if (value.isEmpty) {
          return "Please provide a location.";
        }
        return null;
      },
      onTap: () async {
        //Predict users location as they type
        Prediction p = await PlacesAutocomplete.show(
            context: context,
            apiKey: "AIzaSyCc8pwKufUsqJIht_D4blhyITacJeqqdrk",
            mode: Mode.overlay, // Mode.fullscreen
            language: "en-GB",
            components: [new Component(Component.country, "ie")]);

        //If they choose one store its coordinates
        if (p != null) {
          locationController.text = p.description;

          List<Placemark> placemarks = await Geolocator()
              .placemarkFromAddress(locationController.text);
          Placemark placemark = placemarks[0];
          setState(() {
            myLoc = geo.point(
                latitude: placemark.position.latitude,
                longitude: placemark.position.longitude);
          });
        }
      },
      controller: locationController,
```

```
              decoration: InputDecoration(
                hintText: "Location...",
                border: InputBorder.none,
              ),
            ),
          ),
          Container(
            width: 200.0,
            height: 100.0,
            alignment: Alignment.center,
            child: RaisedButton.icon(
              label: Text(
                "Use Current Location",
                style: TextStyle(color: Colors.white),
              ),
              shape: RoundedRectangleBorder(
                borderRadius: BorderRadius.circular(30.0),
              ),
              color: Colors.blue,
              onPressed: getUserLocation,
              icon: Icon(
                Icons.my_location,
                color: Colors.white,
              ),
            ),
          ),
        ],
      ),
    ),
  ),
 );
}

//Get the users current location
getUserLocation() async {
 Position position = await Geolocator()
    .getCurrentPosition(desiredAccuracy: LocationAccuracy.high);
 List<Placemark> placemarks = await Geolocator()
    .placemarkFromCoordinates(position.latitude, position.longitude);
 Placemark placemark = placemarks[0];
 setState(() {
  myLoc =
      geo.point(latitude: position.latitude, longitude: position.longitude);
 });

 //All possible locations subtypes
```

```
    //'${placemark.subThoroughfare} ${placemark.thoroughfare}, ${placemark.subLocality}, ${placemark.
locality}, ${placemark.subAdministrativeArea}, ${placemark.administrativeArea} ${placemark.postalCod
e}, ${placemark.country},';
    if (position == null) {
      //If gps isnt on notify the user
      SnackBar snackbar = SnackBar(
          content: Text(
        'Make sure location is turned on.',
        overflow: TextOverflow.ellipsis,
      ));
      _scaffoldKey.currentState.showSnackBar(snackbar);
    }
    String formattedAddress =
        "${placemark.locality}, ${placemark.administrativeArea} ";
    locationController.text = formattedAddress;
  }

  bool get wantKeepAlive => true;

  @override
  Widget build(BuildContext context) {
    super.build(context);
    return file == null ? buildSplashScreen() : buildForm();
  }
}
```

post_overview.dart

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';
import 'package:geoflutterfire/geoflutterfire.dart';
import 'package:location/location.dart';

import 'package:find_my_pet/UI/base_widget.dart';
import 'package:find_my_pet/screens/home.dart';
import 'package:find_my_pet/widgets/map.dart';
import 'package:find_my_pet/widgets/post.dart';
import 'package:find_my_pet/widgets/post_tile.dart';
import 'package:find_my_pet/widgets/progress.dart';

//Class to build and display lost, found and spotted pet posts
//Also contains logic for searching for posts within particular radius of the users location
class PostsOverview extends StatefulWidget {
  @override
  _PostsOverviewState createState() => _PostsOverviewState();
}

//SingleTickerProviderStateMixin is used to preserve state if user switches page
```

```
class _PostsOverviewState extends State<PostsOverview>
    with SingleTickerProviderStateMixin {
  Location location =
      new Location(); //Initializes location plugin and starts listening for events
  Geoflutterfire geo =
      Geoflutterfire(); //Store and query a set of keys based on geographic location
  List<Post> posts = [];
  List<String> ids = [];
  bool isLoading = false;
  double distance;
  double dropdownValue; //Store users selcted radius from drop down menu
  double radius = 10;
  int currentIndex;
  TabController _tabController; //Track current tab

  //Store the current users ID if it exists
  final String currentUserId = currentUser?.id;

  //Stire tabs to be used in main widget
  final List<Tab> myTabs = <Tab>[
    Tab(
      child: Container(
        alignment: Alignment.center,
        constraints: BoxConstraints.expand(),
        child: Text(
          "Lost",
        ),
      ),
    ),
    Tab(
      child: Container(
        alignment: Alignment.center,
        constraints: BoxConstraints.expand(),
        child: Text(
          "Found",
        ),
      ),
    ),
    Tab(
      child: Container(
        alignment: Alignment.center,
        constraints: BoxConstraints.expand(),
        child: Text(
          "Spotted",
        ),
      ),
    ),
  ];
```

```
//Get posts and initialise tab contoller when page initially loads
@override
void initState() {
  super.initState();
  getPosts(currentIndex == null ? 0 : currentIndex);
  _tabController = TabController(vsync: this, length: myTabs.length);
  _tabController.addListener(_handleTabs);
}

//Dispose of tab controller to prevent memory leak
@override
void dispose() {
  _tabController.dispose();
  super.dispose();
}

//Function to store current tab when user changes
//And reload posts based on that tab
_handleTabs() async {
  if (currentIndex != _tabController.index) {
    setState(() {
      currentIndex = _tabController.index;
    });
    await getPosts(currentIndex);
  }
}

//Function to get the document IDs of all posts that are within a particular radius to the user
//Creates a geo point for the users location and the posts location using latitude and longitude
//Then calculates the distance between the two points
//If it is within the users set radius the ID is added to a list
getIds(QuerySnapshot checkDistance) async {
  var pos2 = await location.getLocation();
  var point;
  checkDistance.documents.forEach((element) {
    GeoPoint pos1 = element['position']['geopoint'];
    point = geo.point(latitude: pos1.latitude, longitude: pos1.longitude);
    distance = point.distance(lat: pos2.latitude, lng: pos2.longitude);
    if (distance < radius) {
      setState(() {
        ids.add(element.documentID);
      });
    }
  });
}

//Function to get posts based on users selected tab and radius
```

```
getPosts(int index) async {
  setState(() {
    isLoading = true;
  });

  List<Post> tempPosts = [];
  //Depending on users tab it will gather the required post types form the database
  QuerySnapshot snapshot;
  if (index == 0) {
    snapshot = await postsRef
        .orderBy("timestamp", descending: true)
        .where("petStatus", isEqualTo: 'lost')
        .getDocuments();
  } else if (index == 1) {
    snapshot = await postsRef
        .orderBy("timestamp", descending: true)
        .where("petStatus", isEqualTo: 'found')
        .getDocuments();
  } else if (index == 2) {
    snapshot = await postsRef
        .orderBy("timestamp", descending: true)
        .where("petStatus", isEqualTo: 'spotted')
        .getDocuments();
  }

  //Using the returned data it will then check and store the IDs of all those posts within
  //the users chosen radius
  await getIds(snapshot);

  ids.forEach((id) {
    snapshot.documents.forEach((element) {
      if (element.documentID == id) {
        tempPosts.add(Post.fromDocument(element));
      }
    });
  });

  //Clears unneeded data
  setState(() {
    isLoading = false;
    snapshot = null;
    posts = tempPosts.toList();
    tempPosts.clear();
    ids.clear();
  });
}

//Function to build all the returned posts
```

```
buildPosts() {
  if (isLoading) {
    return circularProgress();
  }
  //If there are no posts returned show a screen that states this
  if (posts.isEmpty) {
    return Column(
      crossAxisAlignment: CrossAxisAlignment.center,
      children: <Widget>[
        Text(
          "No posts available",
          textAlign: TextAlign.center,
          style: TextStyle(
            color: Colors.black54,
            fontStyle: FontStyle.italic,
            fontWeight: FontWeight.w600,
            fontSize: 60.0,
          ),
        ),
        Icon(
          Icons.pets,
          size: MediaQuery.of(context).size.width * .2,
          color: Colors.black54,
        )
      ],
    );
  }
  //Pass each post to the PostTile class to build its layout
  //Then add it to a grid tile list
  List<GridTile> gridTiles = [];
  posts.forEach((post) {
    gridTiles.add(GridTile(
      child: PostTile(post),
    ));
  });
  //Create a dynamically sized grid view using the grid list stored above
  return Expanded(
    child: GridView.count(
      crossAxisCount: 2,
      childAspectRatio: (MediaQuery.of(context).size.width /
          (MediaQuery.of(context).size.height * .65)),
      mainAxisSpacing: 1.5,
      crossAxisSpacing: 1.5,
      shrinkWrap: true,
      children: gridTiles,
    ),
  );
}
```

```
//Build the post overview screen
@override
Widget build(BuildContext context) {
  return BaseWidget(builder: (context, sizingInfo) {
    return Scaffold(
      appBar: AppBar(
        title: Text(
          'FindMyPet',
          style: TextStyle(
            color: Colors.white,
            fontSize: 22.0,
          ),
          overflow: TextOverflow.ellipsis,
        ),
        //One action in the app bar, to store a drop down menu
        actions: <Widget>[
          Container(
            //Dynamic sizing and styling
            width: MediaQuery.of(context).size.width * .25,
            padding: EdgeInsets.symmetric(horizontal: 10, vertical: 5),
            decoration: BoxDecoration(
              borderRadius: BorderRadius.only(
                  topLeft: Radius.circular(40),
                  topRight: Radius.circular(40),
                  bottomLeft: Radius.circular(40),
                  bottomRight: Radius.circular(40)),
              color: Colors.white30,
            ),
            //Dropdown button to display radius options
            child: DropdownButton<double>(
              hint: Text("Radius"),
              value: dropdownValue,
              icon: Icon(Icons.arrow_drop_down),
              iconSize: 24,
              elevation: 10,
              style:
                  TextStyle(color: Colors.black, fontWeight: FontWeight.bold),
              underline: SizedBox(),
              onChanged: (double newValue) {
                //If user selects a new radius store it and getPosts based on new value
                setState(() {
                  dropdownValue = newValue;
                  radius = newValue;
                  getPosts(currentIndex);
                });
              },
              items: <double>[5, 10, 20, 50]
```

```
            .map<DropdownMenuItem<double>>((double value) {
          return DropdownMenuItem<double>(
            value: value,
            child: Text('${value.toInt()} km'),
          );
        }).toList(),
      ),
    )
  ],
  centerTitle: true,
  backgroundColor: Theme.of(context).accentColor,
),
body: Column(
  children: <Widget>[
    //Layout and styling of tabs
    Container(
      constraints: BoxConstraints.expand(height: 50),
      child: TabBar(
        controller: _tabController,
        labelColor: Theme.of(context).primaryColor,
        unselectedLabelColor: Theme.of(context).accentColor,
        labelStyle: TextStyle(fontSize: 18),
        unselectedLabelStyle: TextStyle(fontSize: 14),
        tabs: myTabs,
      ),
    ),
    Expanded(
      //Tab View that displays the previously initialised tabs
      //Contoller set to track user selection
      child: TabBarView(
        controller: _tabController,
        children: myTabs.map((Tab tab) {
          return Column(
            //Build posts for each tab
            children: <Widget>[
              buildPosts(),
            ],
          );
        }).toList(),
      ),
    ),
  ],
),
//Floating button that if tapped opens a map that displays each post and where its located
floatingActionButton: FloatingActionButton.extended(
  heroTag: 'btn2',
  onPressed: () => Navigator.push(
    context,
```

```
          MaterialPageRoute(
            builder: (context) => ViewMap(),
          ),
        ),
        label: Text('Map'),
        icon: Icon(
          Icons.navigation,
        ),
        backgroundColor: Theme.of(context).primaryColor,
      ),
      floatingActionButtonLocation: FloatingActionButtonLocation.endFloat,
    );
  });
 }
}
```

post_screen.dart

```
import 'package:flutter/material.dart';

import 'package:find_my_pet/widgets/header.dart';
import 'package:find_my_pet/widgets/post.dart';
import 'package:find_my_pet/widgets/progress.dart';
import 'package:find_my_pet/screens/home.dart';

//Class that displays the page structure for a post page
//But calls on the PostTile class to create the post layout and styling
class PostScreen extends StatelessWidget {
 final String userId;
 final String postId;

 PostScreen({this.userId, this.postId});

 @override
 Widget build(BuildContext context) {
  //Uses future builder to create a posts data based on post id
  return FutureBuilder(
    future: postsRef.document(postId).get(),
    builder: (context, snapshot) {
      if (!snapshot.hasData) {
        return circularProgress();
      }
      Post post = Post.fromDocument(snapshot.data);
      return Center(
          child: Scaffold(
        appBar: header(context, titleText: post.title),
        body: ListView(
          children: <Widget>[
```

```
            Container(child: post),
          ],
        ),
      ));
    },
  );
 }
}
```

profile.dart

```
import 'package:cached_network_image/cached_network_image.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:nice_button/NiceButton.dart';

import 'package:find_my_pet/models/user.dart';
import 'package:find_my_pet/screens/edit_post.dart';
import 'package:find_my_pet/screens/edit_profile.dart';
import 'package:find_my_pet/screens/home.dart';
import 'package:find_my_pet/widgets/header.dart';
import 'package:find_my_pet/widgets/post.dart';
import 'package:find_my_pet/widgets/post_tile.dart';
import 'package:find_my_pet/widgets/progress.dart';

//Class that contains the users profile layout and logic
class Profile extends StatefulWidget {
 final String profileId;

 Profile({this.profileId});

 @override
 _ProfileState createState() => _ProfileState();
}

class _ProfileState extends State<Profile> {
 final String currentUserId = currentUser?.id;
 bool isLoading = false;
 List<Post> posts = [];
 bool postGridOrientation = true;

 //Get profile posts when page is initialised
 @override
 void initState() {
  super.initState();
  getProfilePosts();
 }
```

```
//Function to get profile posts for the current user
getProfilePosts() async {
  setState(() {
    isLoading = true;
  });
  //Get a snapshot of the posts the current user owns
  QuerySnapshot snapshot = await postsRef
      .where("ownerId", isEqualTo: currentUserId)
      .getDocuments();
  setState(() {
    isLoading = false;
    //Get each post returned from the snapshot and turn it into a post widget, then add it to the list
    posts = snapshot.documents.map((doc) => Post.fromDocument(doc)).toList();
  });
}

//Function to navigate to the edit profile screen
editProfile() {
  Navigator.push(
      context,
      MaterialPageRoute(
          builder: (context) => EditProfile(currentUserId: currentUserId)));
}

//Function to log the user out
logout() async {
  await googleSignIn.signOut();
  await FirebaseAuth.instance.signOut();
  Navigator.push(context, MaterialPageRoute(builder: (context) => Home()));
}

//Function to build a button. Used to remove code duplication
//Depending on params passed to it, it will be an edit or logout button
buildProfileButton(bool edit, String text) {
  bool isProfileOwner = currentUserId == widget.profileId;
  if (isProfileOwner) {
    return Container(
      height: MediaQuery.of(context).size.height * .055,
      width: MediaQuery.of(context).size.width * .5,
      child: NiceButton(
        background: edit ? Colors.blue : Colors.red,
        elevation: 4,
        onPressed: edit ? editProfile : logout,
        text: text,
        textColor: Colors.white,
        fontSize: 15,
        radius: 10,
```

```
        padding: EdgeInsets.all(3),
      ),
    );
  } else {
    return Text('');
  }
}

//Function to build the users profile header
buildProfileHeader() {
  return FutureBuilder(
      //get user info based on ID
      future: usersRef.document(widget.profileId).get(),
      builder: (context, snapshot) {
        if (!snapshot.hasData) {
          return circularProgress();
        }
        User user = User.fromDocument(snapshot.data);
        return Padding(
          padding: EdgeInsets.all(15),
          child: Column(children: <Widget>[
            //Builds the users profile image
            Row(
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
              crossAxisAlignment: CrossAxisAlignment.start,
              children: <Widget>[
                Column(
                  children: <Widget>[
                    Row(
                      children: <Widget>[
                        CircleAvatar(
                          radius: 40,
                          backgroundColor: Colors.grey,
                          backgroundImage:
                              CachedNetworkImageProvider(user.photoUrl),
                        ),
                      ],
                    ),
                  ],
                ),
                //Builds the edit and logout button
                Column(
                  crossAxisAlignment: CrossAxisAlignment.center,
                  children: <Widget>[
                    buildProfileButton(true, 'Edit Profile'),
                    const SizedBox(height: 10),
                    buildProfileButton(false, 'Logout'),
                  ],
```

```
              ),
            ],
          ),
          //Displays users name
          Column(
            crossAxisAlignment: CrossAxisAlignment.center,
            children: <Widget>[
              Container(
                alignment: Alignment.centerLeft,
                padding: EdgeInsets.only(top: 12, left: 10),
                child: Text(
                  //If user hasnt set a name it will be stated
                  user.displayName != null
                      ? user.displayName
                      : 'No Display Name',
                  style: TextStyle(
                    fontWeight: FontWeight.bold,
                    fontSize: 16,
                  ),
                ),
              ),
            ],
          ),
        ]),
      );
    });
}

//Function that sets the users selected viewing preference, grid or list
buildTogglePostOrientation() {
  return Row(
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,
    children: <Widget>[
      IconButton(
        icon: Icon(Icons.grid_on),
        color: postGridOrientation
            ? Theme.of(context).primaryColor
            : Colors.grey,
        onPressed: () {
          setState(() {
            this.postGridOrientation = true;
          });
        },
      ),
      IconButton(
        icon: Icon(Icons.list),
        color: !postGridOrientation
            ? Theme.of(context).primaryColor
```

```
            : Colors.grey,
        onPressed: () {
          setState(() {
            this.postGridOrientation = false;
          });
        },
      ),
    ],
  );
}

//Function to delete a post
//It is delted from multiple areas within the database
//Its image,post,feed notifications, comments and map location
deletePost(String postId, String ownerId) async {
  //Delete the post
  postsRef.document(postId).get().then((doc) {
    if (doc.exists) {
      doc.reference.delete();
    }
  });
  //Delete its stored image
  storageRef.child("post_$postId.jpg").delete();
  //Delete all feed notifications
  QuerySnapshot feedSnapshot = await feedRef
      .document(ownerId)
      .collection('comments')
      .where('postId', isEqualTo: postId)
      .getDocuments();
  feedSnapshot.documents.forEach((doc) {
    if (doc.exists) {
      doc.reference.delete();
    }
  });

  await commentsRef.document(postId).delete();
  await locationsRef.document(postId).delete();

  QuerySnapshot commentSnapshot = await commentsRef
      .document(postId)
      .collection('comments')
      .getDocuments();
  commentSnapshot.documents.forEach((doc) {
    if (doc.exists) {
      doc.reference.delete();
    }
  });
}
```

```
//Function to build the post image the user owns
buildProfilePosts() {
  if (isLoading) {
    return circularProgress();
  } else if (posts.isEmpty) {
    //If the user has no posts
    return Column(
      mainAxisAlignment: MainAxisAlignment.center,
      crossAxisAlignment: CrossAxisAlignment.center,
      children: <Widget>[
        Text(
          "No posts available",
          textAlign: TextAlign.center,
          style: TextStyle(
            color: Colors.black54,
            fontStyle: FontStyle.italic,
            fontWeight: FontWeight.w600,
            fontSize: 60.0,
          ),
        ),
        Icon(
          Icons.pets,
          size: MediaQuery.of(context).size.width * .2,
          color: Colors.black54,
        ),
      ],
    );
  } else if (postGridOrientation) {
    //if the user chooses to view posts as a grid
    List<GridTile> gridTiles = [];
    //Stores each post in a grid tile list
    posts.forEach((post) {
      gridTiles.add(GridTile(
        child: PostTile(post),
        footer: Row(
          mainAxisAlignment: MainAxisAlignment.end,
          children: <Widget>[
            //An edit button on the post so the user can make changes
            IconButton(
              icon: Icon(
                Icons.edit,
                color: Colors.blue,
              ),
              onPressed: () async {
                setState(() {
                  isLoading = true;
                });
```

```
        if (isLoading) {
          //Navigate to edit screen if icon is pressed
          await Navigator.push(
            context,
            MaterialPageRoute(
              builder: (context) => EditPost(
                postId: post.postId,
                imageUrl: post.imageUrl,
                description: post.description,
                title: post.title,
              ),
            ),
          );
        }

        setState(() {
          isLoading = false;
        });
        getProfilePosts();
      },
    ),
    //A delete button on the post to remove it and all other aspects from the database
    IconButton(
      icon: Icon(
        Icons.delete,
        color: Colors.redAccent,
      ),
      onPressed: () {
        showDialog(
            context: context,
            builder: (BuildContext context) {
              //An alert box to confirm the user wants to delete the post
              return AlertDialog(
                title: Text('Delete Post'),
                content: Text('Are you sure?'),
                actions: <Widget>[
                  FlatButton(
                    child: Text('Cancel'),
                    onPressed: () {
                      Navigator.of(context).pop();
                    },
                  ),
                  FlatButton(
                    child: Text('Yes'),
                    onPressed: () async {
                      setState(() {
                        isLoading = true;
                      });
```

```
                        if (isLoading) {
                          circularProgress();

                          deletePost(post.postId, post.ownerId);
                          setState(() {
                            isLoading = false;
                          });
                        }
                        Navigator.of(context).pop();
                        getProfilePosts();
                      },
                    )
                  ],
                  elevation: 15,
                );
              });
            }),
        ],
      ),
    ));
  });
  //Display the stored posts in a grid view
  return Expanded(
    child: GridView(
      children: gridTiles,
      gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
        crossAxisCount: 2,
        childAspectRatio: (MediaQuery.of(context).size.width /
            (MediaQuery.of(context).size.height * .7)),
      ),
    ),
  );
 } else if (!postGridOrientation) {
  //Display the posts in a list view
  return Expanded(
    child: GridView(
      children: posts,
      gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
        crossAxisCount: 1,
        childAspectRatio: ((MediaQuery.of(context).size.width * .2) /
            (MediaQuery.of(context).size.height * .3)),
      ),
    ),
  );
 }
}

//Builds overall structure of the screen
```

```
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: header(context, titleText: "Profile"),
      body: Column(
        children: <Widget>[
          buildProfileHeader(),
          Divider(
            height: 0,
          ),
          buildTogglePostOrientation(),
          Divider(height: 0),
          buildProfilePosts(),
        ],
      ),
    );
  }
}
```

search.dart

```
import 'package:flutter/material.dart';
import 'package:cached_network_image/cached_network_image.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:geolocator/geolocator.dart';

import 'package:find_my_pet/screens/post_screen.dart';
import 'package:find_my_pet/widgets/post.dart';
import 'package:find_my_pet/screens/home.dart';
import 'package:find_my_pet/widgets/progress.dart';

//Class to search for posts based on location
class Search extends StatefulWidget {
  @override
  _SearchState createState() => _SearchState();
}

class _SearchState extends State<Search>
    with AutomaticKeepAliveClientMixin<Search> {
  TextEditingController searchController = TextEditingController();
  Future<QuerySnapshot> searchResultsFuture;

  //Function to retrieve the posts that are greater than or equal to the users search
  handleSearch(String query) {
    Future<QuerySnapshot> posts = postsRef
        .where("locationSearch", isGreaterThanOrEqualTo: query.toLowerCase())
        .getDocuments();
```

```
  setState(() {
    searchResultsFuture = posts;
  });
 }

 clearSearch() {
  searchController.clear();
 }

 //Function to get the users location
 getUserLocation() async {
  //Get users current position
  Position position = await Geolocator()
      .getCurrentPosition(desiredAccuracy: LocationAccuracy.high);
  //Get the details of the current postion and store it in a list
  List<Placemark> placemarks = await Geolocator()
      .placemarkFromCoordinates(position.latitude, position.longitude);
  //Store details of the postion
  Placemark placemark = placemarks[0];
  //All possible locations subtypes
  //'${placemark.subThoroughfare} ${placemark.thoroughfare}, ${placemark.subLocality}, ${placemark.
locality}, ${placemark.subAdministrativeArea}, ${placemark.administrativeArea} ${placemark.postalCod
e}, ${placemark.country},';

  //Produce only the users town, county and country
  String formattedAddress = "${placemark.locality}, ${placemark.country}";
  searchController.text = formattedAddress;
 }

 //Function to build the search bar in the app bar widget
 AppBar buildSearchField() {
  return AppBar(
    backgroundColor: Colors.white,
    title: TextFormField(
      controller: searchController,
      decoration: InputDecoration(
        hintText: "Search by location...",
        filled: true,
        //If location icon pressed it generates the users location
        prefixIcon: IconButton(
          icon: Icon(
            Icons.location_on,
            size: 25,
          ),
          onPressed: getUserLocation,
        ),
        //Clears the text field
        suffixIcon: IconButton(
```

```
          icon: Icon(Icons.clear),
          onPressed: clearSearch,
        ),
      ),
      onFieldSubmitted: handleSearch,
    ),
  );
}

//Function that builds a container to display when there is no content
Container buildNoContent() {
  return Container(
    child: Stack(
      children: <Widget>[
        Container(
          child: Center(
            child: ListView(
              shrinkWrap: true,
              children: <Widget>[
                Text(
                  "Search post locations",
                  textAlign: TextAlign.center,
                  style: TextStyle(
                    color: Colors.white70,
                    fontStyle: FontStyle.italic,
                    fontWeight: FontWeight.w600,
                    fontSize: 60.0,
                  ),
                ),
                Icon(
                  Icons.search,
                  size: MediaQuery.of(context).size.width * .3,
                  color: Colors.white70,
                ),
              ],
            ),
          ),
        ),
      ],
    ),
  );
}

//Build the search results based on users input
buildSearchResults() {
  //Uses a future builder to get all posts that relate to user input
  //Then stores each as a post object in a list
  return FutureBuilder(
```

```
      future: searchResultsFuture,
      builder: (context, snapshot) {
        if (!snapshot.hasData) {
          return circularProgress();
        }
        List<LocationResult> searchResults = [];
        snapshot.data.documents.forEach((doc) {
          Post location = Post.fromDocument(doc);
          LocationResult searchResult = LocationResult(location);
          searchResults.add(searchResult);
        });
        //Displays the generated search results to the screen
        return ListView(
          children: searchResults,
        );
      },
    );
  }

  //Variable needed to keep state alive if user leaves page
  bool get wantKeepAlive => true;

  //Builds the overall screen based on if user is searching or not
  @override
  Widget build(BuildContext context) {
    super.build(context);
    return Scaffold(
      backgroundColor: Theme.of(context).primaryColor.withOpacity(0.8),
      appBar: buildSearchField(),
      body:
          searchResultsFuture == null ? buildNoContent() : buildSearchResults(),
    );
  }
}

//Class to create individual post search results
class LocationResult extends StatelessWidget {
  final Post location;

  LocationResult(this.location);

  //Function to navigate to post if tapped
  showPost(context) {
    Navigator.push(
        context,
        MaterialPageRoute(
            builder: (context) =>
                PostScreen(postId: location.postId, userId: location.ownerId)));
```

```
}

//Build the structure of each search result
//Post title, location, image and pet status(lost/found)
@override
Widget build(BuildContext context) {
 return Container(
   color: Theme.of(context).primaryColor.withOpacity(0.7),
   child: Column(
    children: <Widget>[
     GestureDetector(
      onTap: () => showPost(context),
      child: ListTile(
       leading: CircleAvatar(
        backgroundColor: Colors.grey,
        backgroundImage: CachedNetworkImageProvider(location.imageUrl),
       ),
       title: Text(
        location.title,
        style:
          TextStyle(color: Colors.white, fontWeight: FontWeight.bold),
       ),
       subtitle: Text(
        location.location,
        style: TextStyle(color: Colors.white),
       ),
       trailing: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
         Card(
          color: Theme.of(context).accentColor.withBlue(150),
          elevation: 3,
          child: Padding(
           padding: const EdgeInsets.all(4.0),
           child: Text(
            location.isFound ? "Found" : "Lost",
            style: TextStyle(fontSize: 15.0),
           ),
          ),
         ),
        ],
       ),
      ),
     ),
     Divider(
      height: 2.0,
      color: Colors.white54,
     ),
```

```
        ],
      ),
    );
  }
}
```

map.dart

```
import 'dart:async';
import 'package:flutter/material.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:geoflutterfire/geoflutterfire.dart';
import 'package:google_maps_flutter/google_maps_flutter.dart';
import 'package:location/location.dart';
import 'package:rxdart/rxdart.dart';
import 'package:uuid/uuid.dart';

import 'package:find_my_pet/screens/home.dart';
import 'package:find_my_pet/screens/post_screen.dart';

class ViewMap extends StatefulWidget {
  @override
  _ViewMapState createState() => _ViewMapState();
}

class _ViewMapState extends State<ViewMap> {
  GoogleMapController mapController;
  Location location = new Location();
  Geoflutterfire geo = Geoflutterfire();
  //StreamController that captures the latest item that has been added to the controller,
  BehaviorSubject<double> radius = BehaviorSubject.seeded(100);
  Stream<dynamic> query;
  StreamSubscription subscription;
  Map<MarkerId, Marker> markers =
      <MarkerId, Marker>{}; //Store a list of post markers
  String postId = Uuid().v4();
  double lat;
  double lng;

  //Get the distance between two locations
  getDistance(
      double currentLat, double currentLng, double postLat, double postLng) {
    var point = geo.point(latitude: currentLat, longitude: currentLng);
    var distance = point.distance(lat: postLat, lng: postLng);
    return distance;
  }

  //Build the overall map view
```

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    body: Stack(
      children: <Widget>[
        GoogleMap(
          initialCameraPosition: CameraPosition(
            target: LatLng(53.1871, -6.80366),
            zoom: 15,
          ),
          onMapCreated: _onMapCreated,
          myLocationEnabled: true,
          mapType: MapType.normal,
          //Set markers on the map, aka post locations
          markers: Set<Marker>.of(markers.values),
        ),
        Positioned(
            bottom: 50,
            right: 10,
            //Button to navigate back to home screen
            child: FlatButton(
              child: Icon(Icons.arrow_back, color: Colors.white),
              color: Colors.green,
              onPressed: () => Navigator.of(context).pop(),
            )),
        //Create a slider to zoom in and out of map
        Positioned(
            bottom: 50,
            left: 10,
            child: Slider(
              min: 100,
              max: 500,
              divisions: 4,
              value: radius.value,
              label: 'Radius ${radius.value}km',
              activeColor: Colors.green,
              inactiveColor: Colors.green.withOpacity(.7),
              onChanged: _updateQuery,
            ))
      ],
    ),
  );
}

//When the map is created set the controller
_onMapCreated(GoogleMapController controller) {
  _startQuery();
  setState(() {
```

```
    mapController = controller;
  });
}

//set the initial query to show markers
_startQuery() async {
  var pos = await location.getLocation();

  setState(() {
    lat = pos.latitude;
    lng = pos.longitude;
  });

  GeoFirePoint center = geo.point(latitude: lat, longitude: lng);

  //Sub to query. Which sets the radius from a given position
  subscription = radius.switchMap((rad) {
    return geo.collection(collectionRef: locationsRef).within(
          center: center,
          radius: 400,
          field: 'position',
          strictMode: true,
        );
  }).listen(_updateMarkers);
}

//Change the markers dispayed as the map zooms in and out
_updateQuery(value) {
  final zoomMap = {
    100.0: 12.0,
    200.0: 10.0,
    300.0: 7.0,
    400.0: 6.0,
    500.0: 5.0,
  };
  final zoom = zoomMap[value];
  mapController.moveCamera(CameraUpdate.zoomTo(zoom));

  setState(() {
    radius.add(value);
  });
}

//Update the markers displayed and the information each contains
_updateMarkers(List<DocumentSnapshot> docList) {
  //For each marker give it the below layout and add it to the list
  docList.forEach((DocumentSnapshot doc) {
    GeoPoint pos = doc.data['position']['geopoint'];
```

```
    //Get the distance between user and post location
    var distance = getDistance(lat, lng, pos.latitude, pos.longitude);

    var markId = MarkerId(Uuid().v4());
    var marker = Marker(
      markerId: markId,
      position: LatLng(
        pos.latitude,
        pos.longitude,
      ),
      icon: BitmapDescriptor.defaultMarker,
      //The information displayed when marker is tapped
      infoWindow: InfoWindow(
          title: doc['title'],
          snippet: '$distance km from you.',
          //If tapped on navigate to the post
          onTap: () => {
              Navigator.push(
                  context,
                  MaterialPageRoute(
                      builder: (context) => PostScreen(
                          postId: doc['postId'], userId: doc['ownerId'])))
            }),
    );
    setState(() {
      markers[markId] = marker;
    });
  });
 }

 //Dispose of subscription to prevent memory leak
 @override
 void dispose() {
  subscription.cancel();
  super.dispose();
 }
}
```

Widgets

post_tile.dart

```
import 'package:flutter/material.dart';
import 'package:timeago/timeago.dart' as timeago;

import 'package:find_my_pet/screens/post_screen.dart';
import 'package:find_my_pet/widgets/custom_image.dart';
import 'package:find_my_pet/widgets/post.dart';
```

```
//How a post is displayed as a tile
class PostTile extends StatelessWidget {
 final Post post;

 PostTile(this.post);

 //Navigate to post screen
 showPost(context) {
  Navigator.push(
      context,
      MaterialPageRoute(
          builder: (context) =>
              PostScreen(postId: post.postId, userId: post.ownerId)));
 }

 //Builds the post tile layout and styling
 @override
 Widget build(BuildContext context) {
  //Storing screen heigh and width for dynamic sizing
  var height = MediaQuery.of(context).size.height;
  var width = MediaQuery.of(context).size.width;
  //If the tile is pressed anywhere it navigate to the post screen
  return InkWell(
    onTap: () => showPost(context),
    child: Container(
      height: height,
      width: width,
      child: Card(
          shape: RoundedRectangleBorder(
            borderRadius: BorderRadius.circular(15),
          ),
          elevation: 4,
          margin: EdgeInsets.all(4),
          child: Column(
            children: <Widget>[
              Stack(
                children: <Widget>[
                  //Creates a rounded-rectangular clip.
                  ClipRRect(
                    borderRadius: BorderRadius.only(
                      topLeft: Radius.circular(15),
                      topRight: Radius.circular(15),
                    ),
                    //Display the post image
                    child: Container(
                      height: height * .2,
                      width: double.infinity,
```

```
            child: cachedNetworkImage(context, post.imageUrl),
          ),
        ),
      ],
    ),
    Padding(
      padding: const EdgeInsets.only(
        top: 5.0,
        left: 5,
        right: 5,
      ),
      child: Container(
        width: MediaQuery.of(context).size.width,
        height: height * .03,
        child: Row(
          //Posts title
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          children: <Widget>[
            Container(
              width: MediaQuery.of(context).size.width / 5,
              child: Text(
                post.title,
                textWidthBasis: TextWidthBasis.longestLine,
                overflow: TextOverflow.clip,
                maxLines: 1,
                style: TextStyle(
                  fontWeight: FontWeight.bold,
                  fontSize: 16,
                  color: Theme.of(context).accentColor,
                ),
              ),
            ),
            //Display time since post was posted
            Container(
              width: MediaQuery.of(context).size.width / 5,
              child: Text(
                '${timeago.format(post.date)}',
                textWidthBasis: TextWidthBasis.longestLine,
                overflow: TextOverflow.fade,
                maxLines: 1,
                style: TextStyle(
                    fontStyle: FontStyle.italic,
                    fontWeight: FontWeight.w400,
                    fontSize: 12),
              ),
            ),
          ],
        ),
```

```
              ),
            ),
            //Display the posts location
            Padding(
              padding: const EdgeInsets.only(
                  top: 5.0, left: 5, right: 5, bottom: 0),
              child: Container(
                height: height * .03,
                child: Row(
                  mainAxisAlignment: MainAxisAlignment.center,
                  children: <Widget>[
                    Expanded(
                      child: Container(
                        child: Text(
                          post.location,
                          softWrap: false,
                          overflow: TextOverflow.fade,
                          style: TextStyle(
                            fontWeight: FontWeight.bold,
                            fontSize: 12,
                          ),
                        ),
                      ),
                    ),
                  ],
                ),
              ),
            ),
          ],
        )),
    ),
  );
 }
}
```

post.dart

```
import 'package:flutter/material.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:geoflutterfire/geoflutterfire.dart';
import 'package:location/location.dart';
import 'package:intl/intl.dart';

import 'package:find_my_pet/screens/comments.dart';
import 'package:find_my_pet/screens/home.dart';
import 'package:find_my_pet/widgets/custom_image.dart';
import 'package:find_my_pet/widgets/progress.dart';
```

```
//Data Model and widget so methods can be applied before passing to state class
class Post extends StatefulWidget {
  final String title;
  final String username;
  final String ownerId;
  final String postId;
  final DateTime date;
  final String description;
  final String location;
  final String imageUrl;
  final bool isFound;
  final Map loc;
  final String petStatus;

  Post({
    this.title,
    this.username,
    this.ownerId,
    this.postId,
    this.date,
    this.description,
    this.location,
    this.imageUrl,
    this.isFound,
    this.loc,
    this.petStatus,
  });

  //factory describes creating an instance from a document
  factory Post.fromDocument(DocumentSnapshot doc) {
    Timestamp dateTimestamp = doc['date'];
    DateTime convertedDate = DateTime.parse(dateTimestamp.toDate().toString());
    return Post(
      title: doc['title'],
      username: doc['username'],
      ownerId: doc['ownerId'],
      postId: doc['postId'],
      date: convertedDate,
      description: doc['description'],
      location: doc['location'],
      imageUrl: doc['imageUrl'],
      isFound: doc['isFound'],
      loc: doc['position'],
      petStatus: doc['petStatus'],
    );
  }
  @override
  _PostState createState() => _PostState(
```

```
      title: this.title,
      username: this.username,
      ownerId: this.ownerId,
      postId: this.postId,
      date: this.date,
      description: this.description,
      location: this.location,
      imageUrl: this.imageUrl,
      isFound: this.isFound,
      loc: this.loc,
      petStatus: this.petStatus,
    );
}

class _PostState extends State<Post> {
  final String title;
  final String username;
  final String ownerId;
  final String postId;
  final DateTime date;
  final String description;
  final String location;
  final String imageUrl;
  final bool isFound;
  final Map loc;
  final String petStatus;

  _PostState({
    this.title,
    this.username,
    this.ownerId,
    this.postId,
    this.date,
    this.description,
    this.location,
    this.imageUrl,
    this.isFound,
    this.loc,
    this.petStatus,
  });

  Geoflutterfire geo = Geoflutterfire();
  Location location2 = new Location();

  //Build the posts header
  buildPostHeader() {
    //Uses a future builder to first get posts info from firebase then display it
    return FutureBuilder(
```

```
    future: usersRef.document(ownerId).get(),
    builder: (context, snapshot) {
      //Show loading spinner if no data
      if (!snapshot.hasData) {
        return circularProgress();
      }
      //The top structure of the post
      return ListTile(
        title: Row(
          children: <Widget>[
            //The posts title
            Text(
              title,
              style: TextStyle(
                color: Theme.of(context).accentColor,
                fontWeight: FontWeight.bold,
              ),
              overflow: TextOverflow.ellipsis,
            ),
          ],
        ),
        //The posts location
        subtitle: Text(
          location,
          overflow: TextOverflow.ellipsis,
          maxLines: 1,
        ),
        //A button which leads to a comments screen
        trailing: RawMaterialButton(
          onPressed: () => showComments(context,
              postId: postId, ownerId: ownerId, imageUrl: imageUrl),
          child: Icon(
            Icons.comment,
            size: 28.0,
            color: Colors.white,
          ),
          shape: CircleBorder(),
          elevation: 2,
          fillColor: Theme.of(context).accentColor,
          padding: EdgeInsets.all(10),
        ),
      );
    },
  );
}

//Handles getting the post image and layout
buildPostImage() {
```

```
  return Container(
    alignment: Alignment.center,
    height: MediaQuery.of(context).size.height * .4,
    child: cachedNetworkImage(context, imageUrl),
  );
}

//Gets the number of comments for the post
getCommentCount() async {
  await commentsRef
      .document(postId)
      .collection('comments')
      .getDocuments()
      .then((snapshot) {
    return snapshot.documents.length;
  });
}

//Gets the distance of the post from the user
getDistance() async {
  GeoPoint pos1 = loc['geopoint'];

  var pos2 = await location2.getLocation();
  var point = geo.point(latitude: pos1.latitude, longitude: pos1.longitude);
  var distance = point.distance(lat: pos2.latitude, lng: pos2.longitude);
  return distance;
}

//Build the bottom of the post
buildPostFooter() {
  return Column(
    children: <Widget>[
      Padding(
        padding: const EdgeInsets.all(12.0),
        child: Row(
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          children: <Widget>[
            Container(
              child: RichText(
                text: TextSpan(
                    style: TextStyle(color: Colors.black),
                    children: <TextSpan>[
                      //Displays the data in a friendly format
                      TextSpan(
                          text: 'Date: ',
                          style: new TextStyle(fontWeight: FontWeight.bold)),
                      TextSpan(text: '${DateFormat.MMMEd().format(date)}'),
                    ]),
```

```
          ),
        ),
        //Displays the distance of post from user
        Container(
          margin: EdgeInsets.only(left: 10.0),
          //Uses a future builder to get distance before displaying
          child: FutureBuilder(
            future: getDistance(),
            builder: (context, snapshot) {
              if (snapshot.hasError) return Text('${snapshot.error}');
              if (snapshot.hasData)
                return Text(
                  "${snapshot.data} km away",
                  style: TextStyle(
                    color: Colors.black,
                    fontWeight: FontWeight.bold,
                  ),
                );
              return const CircularProgressIndicator();
            },
          )),
      ],
    ),
  ),
  Divider(
    height: 2,
    thickness: 2,
    endIndent: 10,
    indent: 10,
  ),
  //displays owner information
  Padding(
    padding: const EdgeInsets.only(bottom: 12.0, top: 10, left: 12),
    child: Column(
      children: <Widget>[
        Container(
          alignment: Alignment.centerLeft,
          child: RichText(
            text: TextSpan(
              style: TextStyle(color: Colors.black),
              children: <TextSpan>[
                TextSpan(
                  text: 'Owner:\t\t',
                  style: new TextStyle(fontWeight: FontWeight.bold)),
                TextSpan(text: '$username'),
              ]),
          ),
        ),
```

```
          //Displays post description
          Padding(
            padding: const EdgeInsets.only(top: 10.0, bottom: 10),
            child: Container(
              alignment: Alignment.centerLeft,
              child: RichText(
                text: TextSpan(
                    style: TextStyle(
                      color: Colors.black,
                    ),
                    children: <TextSpan>[
                      TextSpan(
                          text: 'Description:\n',
                          style:
                              new TextStyle(fontWeight: FontWeight.bold)),
                      TextSpan(text: '$description'),
                    ]),
              ),
            ),
          ),
        ],
      ),
    ),
  ],
 );
}

//The overall structure of screen
@override
Widget build(BuildContext context) {
 return Card(
   margin: EdgeInsets.all(5),
   elevation: 5,
   child: Column(
     children: <Widget>[
       buildPostImage(),
       buildPostHeader(),
       Divider(
         height: 2,
         thickness: 2,
         endIndent: 10,
         indent: 10,
       ),
       buildPostFooter(),
     ],
   ),
 );
}
```

```
}

//Navigate to the comments screen and passes necessary information to build comments
showComments(BuildContext context,
    {String postId, String ownerId, String imageUrl}) {
  Navigator.push(context, MaterialPageRoute(
    builder: (context) {
      return Comments(
        postId: postId,
        postOwnerId: ownerId,
        postImageUrl: imageUrl,
      );
    },
  ));
}
```

custom_image.dart

```
import 'package:cached_network_image/cached_network_image.dart';
import 'package:flutter/material.dart';

//Gets the image from firebase and provides styling
//If loading displays a loading indicator, if there is an error it displays error message
Widget cachedNetworkImage(context, imageUrl) {
  return Card(
    elevation: 10,
    margin: EdgeInsets.all(1),
    child: CachedNetworkImage(
      imageUrl: imageUrl,
      height: MediaQuery.of(context).size.height * .4,
      width: double.infinity,
      fit: BoxFit.cover,
      placeholder: (context, url) => Padding(
        child: CircularProgressIndicator(),
        padding: EdgeInsets.all(40),
      ),
      //If image doesn load
      errorWidget: (context, url, error) => Icon(Icons.error),
    ),
  );
}
```

header.dart

```
import 'package:flutter/material.dart';

//Default layout for each pages header to stop code repitition
```

```
//Which excepts different titles and conditionals as parameters
AppBar header(context,
    {bool isAppTitle = false, String titleText, removeBackButton = false}) {
  return AppBar(
    automaticallyImplyLeading: removeBackButton ? false : true,
    title: Text(
      isAppTitle ? "FindMyPet" : titleText,
      style: TextStyle(
        color: Colors.white,
        fontFamily: isAppTitle ? "Signatra" : "",
        fontSize: isAppTitle ? 50.0 : 22.0,
      ),
      overflow: TextOverflow.ellipsis,
    ),
    centerTitle: true,
    backgroundColor: Theme.of(context).accentColor,
  );
}
```

Progress.dart

```
import 'package:flutter/material.dart';
//Used to store loading indicators

//produces a circular loading spinner
Container circularProgress() {
  return Container(
      alignment: Alignment.center,
      padding: EdgeInsets.only(top: 10.0),
      child: CircularProgressIndicator(
        valueColor: AlwaysStoppedAnimation(Colors.purple),
      ));
}

//produces a linear loading spinner
Container linearProgress() {
  return Container(
    padding: EdgeInsets.only(bottom: 10.0),
    child: LinearProgressIndicator(
      valueColor: AlwaysStoppedAnimation(Colors.purple),
    ),
  );
}
```

main.dart

```
import 'package:flutter/material.dart';
```

```dart
import 'package:flutter/services.dart';
import 'package:find_my_pet/screens/home.dart';

//The main that starts the run process
void main() {
  //Sets the app to be only viewed in portrait mode
  WidgetsFlutterBinding.ensureInitialized();
  SystemChrome.setPreferredOrientations([DeviceOrientation.portraitUp])
      .then((_) {
    runApp(MyApp());
  });
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    //Setting theme data to reflect thoughout the app
    return MaterialApp(
      title: 'FindMyPet',
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        primarySwatch: Colors.deepPurple,
        accentColor: Colors.teal,
      ),
      home: Home(),
    );
  }
}
```

Sizing

base_widget.dart

```dart
import 'package:flutter/material.dart';
import '../UI/sizing.dart';
import '../utils/ui_utils.dart';

//Built to wrap around project widgets to get sizing information
class BaseWidget extends StatelessWidget {
  final Widget Function(
      BuildContext context, SizingInformation sizingInformation) builder;

  const BaseWidget({Key key, this.builder}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    var mediaQuery = MediaQuery.of(context);
    //Returns sizing, orientation, device, screen, and widget information
    return LayoutBuilder(builder: (context, boxConstraints) {
```

```
    var sizingInformation = SizingInformation(
        orientation: mediaQuery.orientation,
        deviceScreenType: getDeviceType(mediaQuery),
        screenSize: mediaQuery.size,
        localWidgetSize:
            Size(boxConstraints.maxWidth, boxConstraints.maxHeight));
    return builder(context, sizingInformation);
   });
  }
}
```

screen_type.dart

```
enum DeviceScreenType { Mobile, Tablet, Desktop }
```

sizing.dart

```
import 'package:flutter/material.dart';
import './screen_type.dart';

//For sizing types
class SizingInformation {
  final Orientation orientation;
  final DeviceScreenType deviceScreenType;
  final Size screenSize;
  final Size localWidgetSize;

  SizingInformation({
    this.orientation,
    this.deviceScreenType,
    this.screenSize,
    this.localWidgetSize,
  });

  //Overriding toString method to show sizing information during production
  @override
  String toString() {
    return 'ScreenSize: $screenSize, widgetSize $localWidgetSize, orientation: $orientation, deviceScreenT
ype $deviceScreenType';
  }
}
```

ui_utils.dart

```
import 'package:flutter/widgets.dart';
import '../UI/screen_type.dart';
```

```
//Determines device type based on screen width
DeviceScreenType getDeviceType(MediaQueryData mediaQuery) {
  var orientation = mediaQuery.orientation;

  //Fixed device width, changes with orientation
  double deviceWidth = 0;

  if (orientation == Orientation.landscape) {
    deviceWidth = mediaQuery.size.height;
  } else {
    deviceWidth = mediaQuery.size.width;
  }

  if (deviceWidth > 950) {
    return DeviceScreenType.Desktop;
  }

  if (deviceWidth > 600) {
    return DeviceScreenType.Tablet;
  }

  return DeviceScreenType.Mobile;
}
```

Packages

pubspec.yaml

```
name: find_my_pet
description: A new Flutter project.

version: 1.0.0+1

environment:
  sdk: ">=2.2.2 <3.0.0"

dependencies:
  flutter:
    sdk: flutter
  provider: ^4.0.0
  intl: ^0.16.1
  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^0.1.2
  # For storing data to device
  shared_preferences: ^0.5.6+3
  # A composable, Future-based library for making HTTP requests.
  http: ^0.12.0+4
  # Firebase Core API, which enables connecting to multiple Firebase apps.
```

```
  firebase_core: ^0.4.4+3
  #Firebase required packages
  firebase_analytics: ^5.0.11
  firebase_auth: ^0.15.5+3
  cloud_firestore: ^0.13.4+2
  firebase_storage: ^3.1.5
  firebase_messaging: ^6.0.13
  #Image package
  image_picker: ^0.6.4
  #Google sign in package
  google_sign_in: ^4.4.0
  #Location packages
  flutter_google_places: ^0.2.4
  google_maps_flutter: ^0.5.25+3
  google_maps_webservice: ^0.0.16
  geoflutterfire: ^2.1.0
  location: ^3.0.2
  nice_button: ^0.1.7
  # Provides access to the platform specific location services
  geolocator: ^5.3.1
  # Simple, fast generation of RFC4122 UUIDs.
  uuid: ^2.0.4
  #ability to load, save and manipulate images in a variety of different file formats
  image: ^2.1.4
  #animation library for Flutter
  animator: ^1.0.0+5
  # Finding commonly used locations on the filesystem
  path_provider: ^1.6.5
  # Convinient timestamps
  timeago: ^2.0.26
  # how images from the internet and keep them in the cache directory.
  cached_network_image:
  flutter_svg:
  device_preview: ^0.4.4

dev_dependencies:
  flutter_test:
    sdk: flutter

flutter:
  uses-material-design: true

  fonts:
    - family: Lato
      fonts:
        - asset: assets/fonts/Lato-Regular.ttf
        - asset: assets/fonts/Lato-bold.ttf
          weight: 700
```

```
  - family: Anton
    fonts:
      - asset: assets/fonts/Anton-Regular.ttf
  - family: Signatra
    fonts:
      - asset: assets/fonts/Signatra.ttf
 assets:
  - assets/images/google_signin_button.png
```

Models

user.dart

```
import 'package:cloud_firestore/cloud_firestore.dart';

class User {
  final String id;
  final String username;
  final String email;
  final String photoUrl;
  final String displayName;

  User({
    this.id,
    this.username,
    this.email,
    this.photoUrl,
    this.displayName,
  });

  //factory describes creating an instance from a firebase document
  factory User.fromDocument(DocumentSnapshot doc) {
    return User(
      id: doc['id'],
      email: doc['email'],
      username: doc['username'],
      photoUrl: doc['photoUrl'],
      displayName: doc['displayName'],
    );
  }
}
```

Firebase Functions

index.js

```
const functions = require('firebase-functions');
const admin = require('firebase-admin');
admin.initializeApp(functions.config().firebase);
var GeohashDistance = require('geohash-distance');
```

```
//Function to display notifications for new comments on a users post
exports.onCreateActivityFeedItem = functions.firestore
    .document('/feed/{userId}/comments/{activityFeedItem}')
    .onCreate(async (snapshot, context) => {
        //When a feed item is added run this function
        console.log('Activity Feed Item Cerated', snapshot.data());

        //Get the user connected to the feed
        const userId = context.params.userId;

        const userRef = admin.firestore().doc(`users/${userId}`);
        const doc = await userRef.get();

        //Check if user has a notification token. Send if they do
        const androidNotificationToken = doc.data().androidNotificationToken;
        if (androidNotificationToken) {
            //send notification
            sendNotification(androidNotificationToken, snapshot.data());
        } else {
            console.log("No token for user, notification could not be sent");
        }

        function sendNotification(androidNotificationToken, activityFeedItem) {
            //Body of the notification message
            const body = `${activityFeedItem.displayName} replied: ${activityFeedItem.commentData}`;

            //Create message for push notification
            const message = {
                notification: { body: body },
                //To just be sent to this person
                token: androidNotificationToken,
                data: { recipient: userId },
            };

            //send message with firebase admin
            admin.messaging().send(message).then(response => {
                console.log("Sent Message", response);
            }).catch(error => { console.log("Error", error); })
        }
    })

//Function to display notifications to users within a 10km radius of the posts location
exports.onCreatePost = functions.firestore
    .document('/posts/{postId}')
    .onCreate(async (postSnap, context) => {
        //When a post is added run this function
        console.log('New Post Created', postSnap.data());
```

```
    var postLoc = postSnap.data().position.geohash;
    var userRef = admin.firestore().collection(`users`);
    userRef.get().then(snapshot => {
      //For each user, get their location and check if they are within the set radius of the post
      snapshot.forEach(doc => {
        var userLoc = doc.data().position.geohash;
        var distance = GeohashDistance.inKm(userLoc, postLoc);
        //If conditionals are met, send notification
        if (distance < 10 && postSnap.data().petStatus != "found" && postSnap.data().petStatus != "spo
tted" && postSnap.data().ownerId != doc.data().id) {
          if (doc.data().androidNotificationToken) {
            //send notification
            sendPostNotification(doc.data().androidNotificationToken, doc.data().id, postSnap.data());
          } else {
            console.log("No token for user, notification could not be sent");
          }
        }
        else {
          console.log("Further than 15km");
        }

      });
    });

    function sendPostNotification(androidNotificationToken, id, postInformation) {

      const body = `A pet has gone missing nearby. Please keep an eye out, it's name is ${postInformatio
n.title}.`;

      //Create message for push notification
      const message = {
        notification: { body: body, image: postInformation.imageUrl },
        token: androidNotificationToken,
        data: { recipient: id }
      };
      //send message with firebase admin
      admin.messaging().send(message).then(response => {
        console.log("Sent Message for post", response);
      }).catch(error => { console.log("Error", error); })
    }
  }
  )
```