MiniMates

A **Playdate Platform** for Parents

to Explore Activities

Video Link:

https://youtu.be/5g2vcD1gByA

Midterm Report

4495 Applied Research -- Section 001

Becky Jin 300374335

Feb 24, 2025

Introduction

Domain and Background

In today’s fast-paced world, where work-life balance is often a challenge, parents are increasingly seeking ways to enrich their children's lives through social interaction and engaging activities. Playdates provide a valuable opportunity for children to develop social skills, build friendships, and engage in creative play. However, for many parents, organizing these social gatherings can be a daunting task. With demanding schedules, managing multiple responsibilities, and balancing daily routines, finding the time and energy to plan playdates becomes difficult. Moreover, the process of discovering suitable playdate activities, finding the right venues, and connecting with other parents who share similar interests or parenting styles can be overwhelming.

Even though parents want their children to socialize, planning successful playdates can be harder than it seems. Parents need to think about many things, like choosing age-appropriate and fun activities, picking a good location, and making sure other families are a good match. Even when everything seems right, finding other parents who are available and interested in the same activities can take a lot of time and be frustrating. As a result, children and parents miss out on chances to connect and enjoy social time together.

To solve these problems, this project aims to create a platform called **MiniMates**, which simplifies the process of planning playdates. **MiniMates** will allow parents to explore fun and age-appropriate activities, find local venues like parks or play centers, and connect with other parents who are also looking to arrange playdates. By bringing everything into one place, MiniMates will save parents time and effort, helping them plan playdates more easily while still keeping up with their busy lives.

The goal is to make it simple for parents to find great playdate opportunities for their children, making it easier for kids to socialize and for families to enjoy time together.

Summary of the Initially Proposed Research Project

Organizing playdates can be challenging due to busy schedules, difficulty in finding suitable activities, and connecting with like-minded parents. MiniMates aims to simplify this process by providing a centralized platform where parents can explore activities, find venues, and schedule playdates effortlessly.

MiniMates will be a full-stack platform designed to:

- Allow parents to post and discover playdate events
- Provide AI-powered event recommendations based on location, preferences, and child's age
- Enable customized filtering options for activities based on time, location, and interests
- Offer secure user authentication & profile management
- Support easy event modifications & cancellations

The platform will enhance community engagement, making it easier for parents to connect and plan meaningful playdates while fostering their child's social and cognitive development.

Technology Stack

- **Front-end:** Flutter (Dart)
- **Back-end:** Node.js, Express.js
- **Database:** MongoDB
- **AI-powered Recommendations:** OpenAI
- **Operating System:** macOS

Changes to the proposal

1. Removing the Kids Matching Feature
   Initially, I planned to include a kid's matching feature, where children could find playmates based on shared hobbies, age groups, or gender preferences. However, after further evaluation, I realized that:

- Privacy concerns could arise regarding user data and children’s personal information.
- Parental preferences often influence playdate decisions more than the children's direct interaction with the app. Instead, I will prioritize the core functionality—ensuring that parents can effectively post, manage, and join playdates with ease.

2. Focusing on Playdate Management & User Profiles

With the removal of the kids matching feature, I am shifting my primary focus to enhancing playdate management by implementing:

- Post & Join Playdates – Parents can create and share events seamlessly.
- Event Management System – Users can edit, delete, or cancel their playdates efficiently.

3. Switching from MongoDB & Node.js to Firebase

Originally, I planned to build the back-end using MongoDB and Node.js. However, after further exploration, I found that Firebase provides a more flexible, scalable, and developer-friendly solution that better suits this project’s needs.

Reasons for the Switch:

Simplified back-end management – Firebase offers an integrated database, authentication, and real-time sync.

Faster development speed – As I am still improving my Flutter skills, Firebase’s built-in services reduce development complexity compared to setting up a separate server and database.

Strong authentication & security – Firebase Authentication provides secure user sign-in options, which align with the app’s need for reliable account management.

With Firebase, I can focus more on the front-end functionality and user experience rather than managing a complex back-end infrastructure.

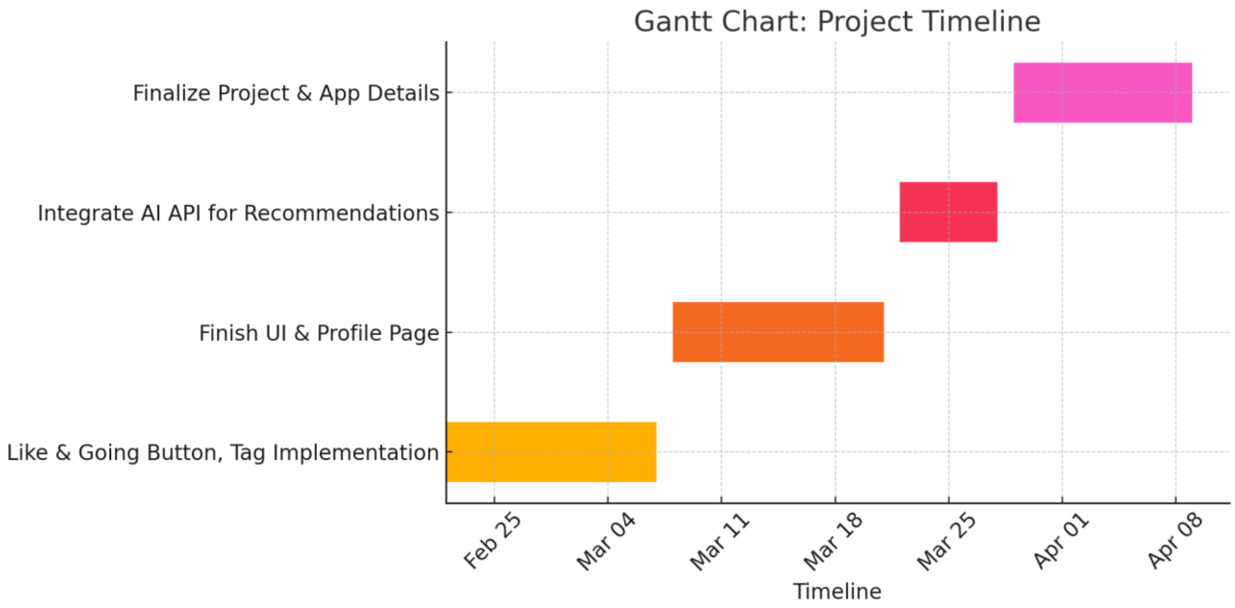
Project Planning and Timeline

Milestones, deadlines, deliverables

Till now, I finished splash page, onboarding page, login page, signup page, login, signup and sign out implementation, main page list view implementation, and the part of the post playdates implementation.

Issues Remaining:

I still have problems for the onboarding page displaying, I will figure this out later.



Feb 22 - Mar 07: Finishing the “like” and “going” button implementation, and the tag implementation.

Mar 08 - Mar 21: Finishing all the UI pages, and profile page implementation.

Mar 22 - Mar 28: Integrate AI API to generate the AI recommendations and reflect it to the filters(tags).

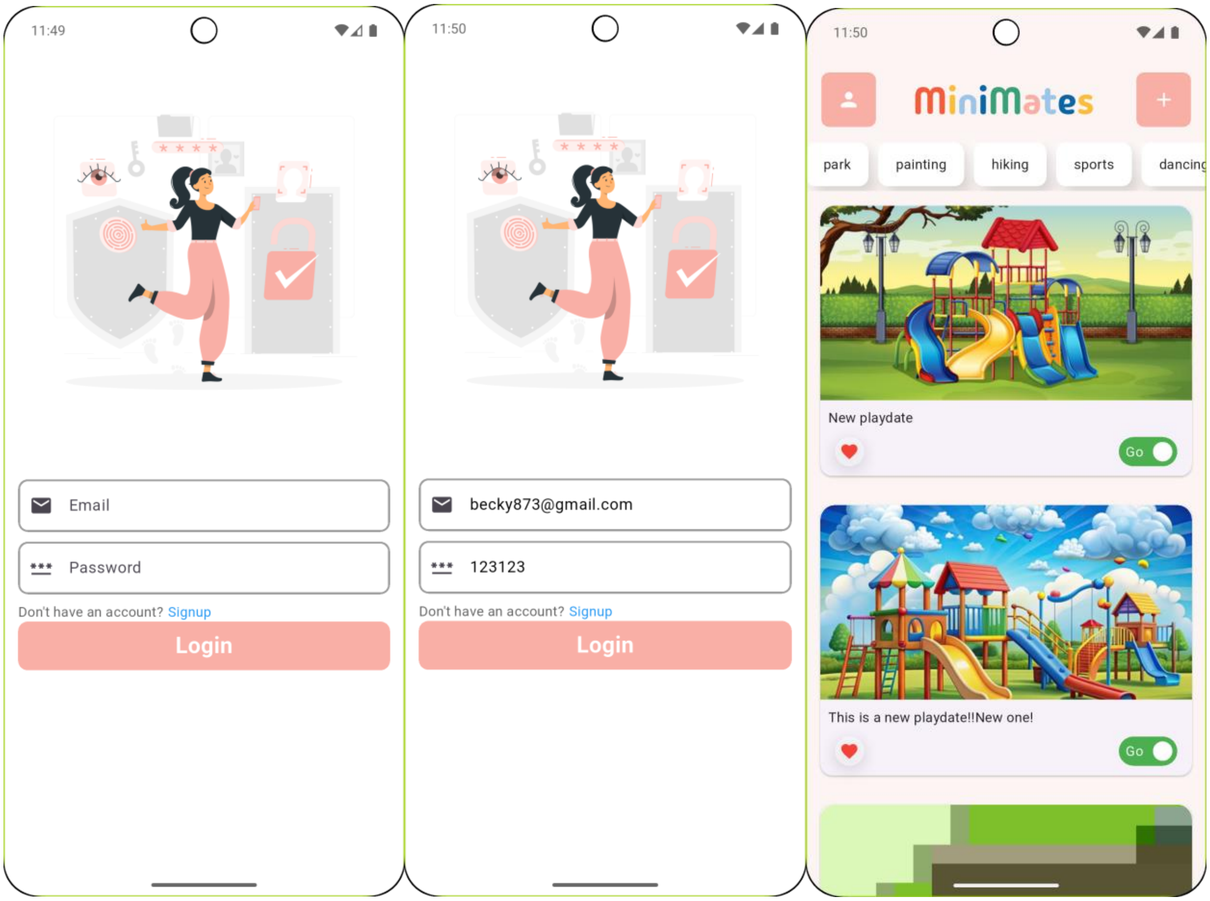
Mar 29 - Apr 09: Finalize the project and the app details.

Timeline

TBD

Implementation

Implemented Feature: Log-In Functionality



becky873@gmail.com

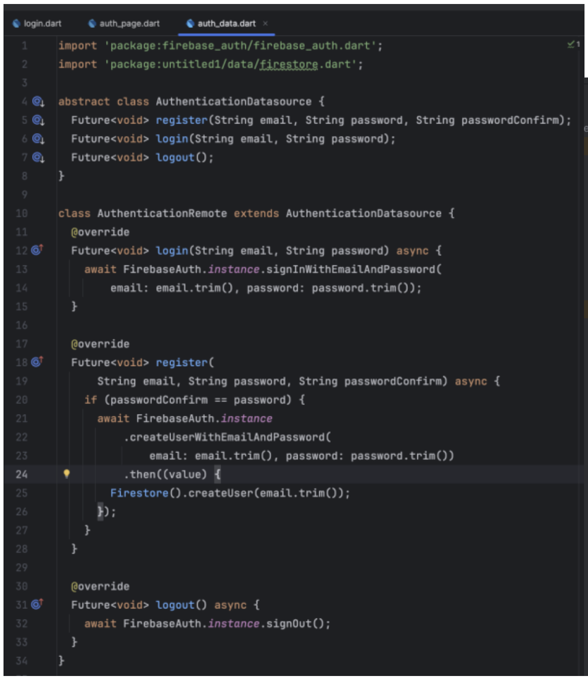
123123

New playdate

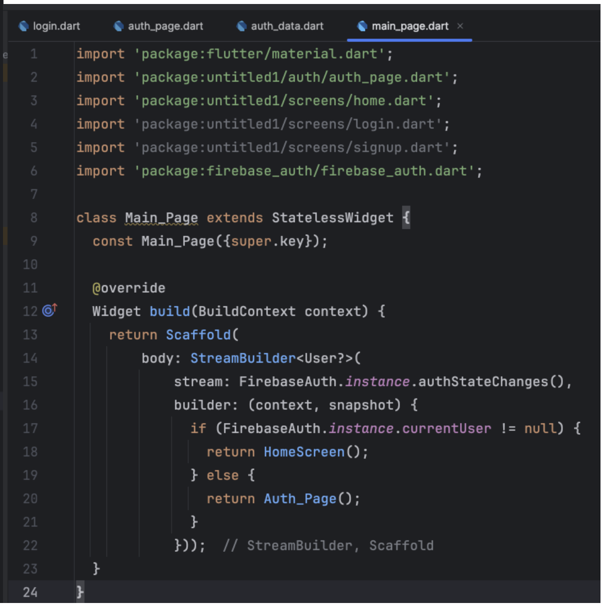
This is a new playdate!!New one!

Login Flow

LoginScreen → login(...) in AuthenticationRemote → Authenticates user with FirebaseAuth → State change detected in Main_Page → If successful, routes to HomeScreen; otherwise, user remains on LoginScreen.



```dart
import 'package:firebase_auth/firebase_auth.dart';
import 'package:untitled1/data/firestore.dart';

abstract class AuthenticationDatasource {
  Future<void> register(String email, String password, String passwordConfirm);
  Future<void> login(String email, String password);
  Future<void> logout();
}

class AuthenticationRemote extends AuthenticationDatasource {
  @override
  Future<void> login(String email, String password) async {
    await FirebaseAuth.instance.signInWithEmailAndPassword(
        email: email.trim(), password: password.trim());
  }

  @override
  Future<void> register(
      String email, String password, String passwordConfirm) async {
    if (passwordConfirm == password) {
      await FirebaseAuth.instance
          .createUserWithEmailAndPassword(
              email: email.trim(), password: password.trim())
          .then((value) {
        Firestore().createUser(email.trim());
      });
    }
  }

  @override
  Future<void> logout() async {
    await FirebaseAuth.instance.signOut();
  }
}
```



```dart
import 'package:flutter/material.dart';
import 'package:untitled1/auth/auth_page.dart';
import 'package:untitled1/screens/home.dart';
import 'package:untitled1/screens/login.dart';
import 'package:untitled1/screens/signup.dart';
import 'package:firebase_auth/firebase_auth.dart';

class Main_Page extends StatelessWidget {
  const Main_Page({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
        body: StreamBuilder<User?>(
            stream: FirebaseAuth.instance.authStateChanges(),
            builder: (context, snapshot) {
              if (FirebaseAuth.instance.currentUser != null) {
                return HomeScreen();
              } else {
                return Auth_Page();
              }
            }));  // StreamBuilder, Scaffold
  }
}
```

Implementation:

- LoginScreen (login.dart):

  Displays UI elements for email/password input, then calls sign-in method.

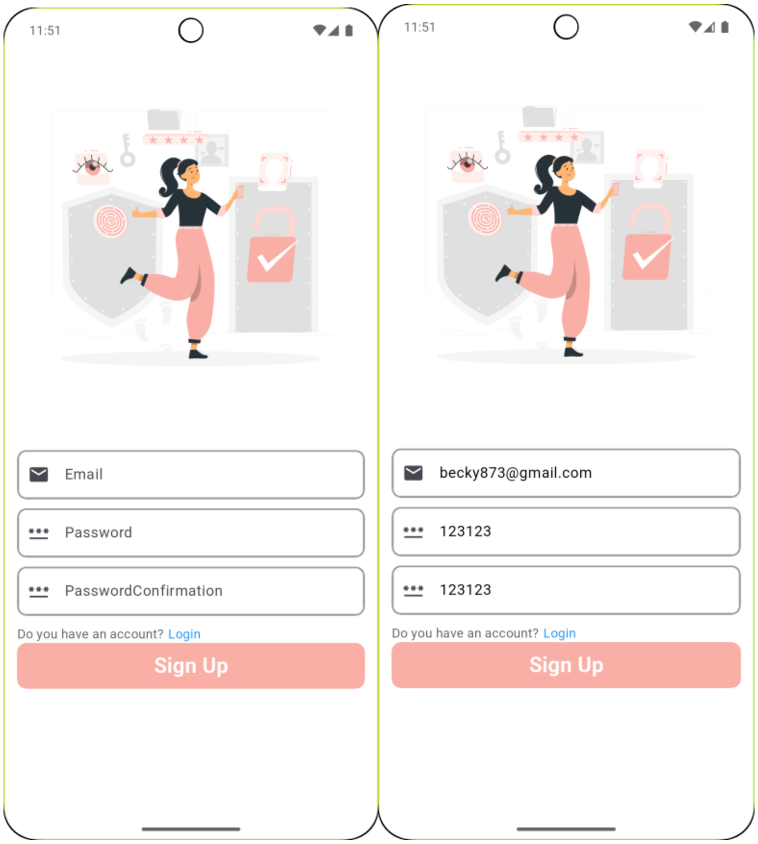
- AuthenticationRemote (auth_data.dart):

  Contains login(...) method that invokes FirebaseAuth sign-in logic.

- Main_Page (main_page.dart):

  Listens for changes in the Firebase auth state. If the user is authenticated, shows HomeScreen; otherwise, displays Auth_Page (the login/signup flow).

Implemented Feature: Sign-Up Functionality



Email

Password

PasswordConfirmation

Do you have an account? Login

becky873@gmail.com

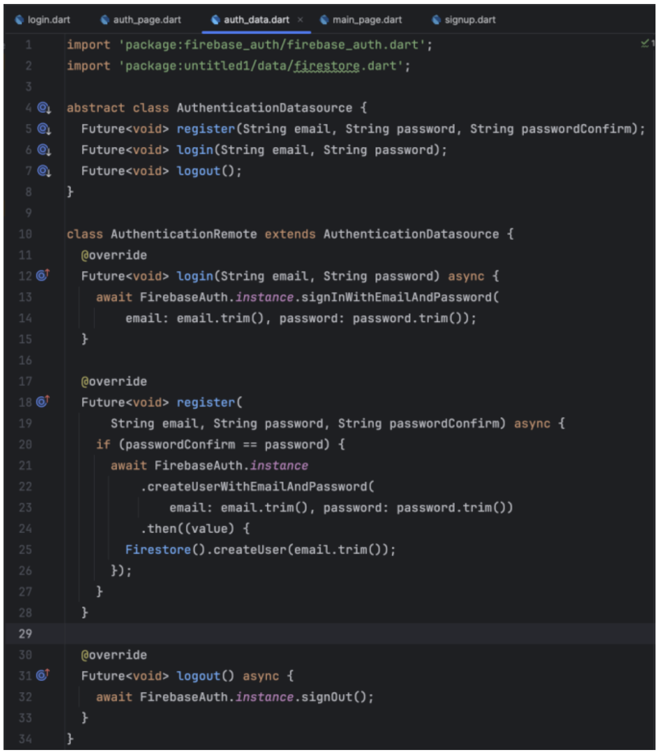
123123

123123

Do you have an account? Login

Sign-Up Flow:

- SignupScreen → register(...) in AuthenticationRemote → Creates user in FirebaseAuth & Firestore → User is authenticated → Main_Page routes to Home.



```dart
import 'package:firebase_auth/firebase_auth.dart';
import 'package:untitled1/data/firestore.dart';

abstract class AuthenticationDatasource {
  Future<void> register(String email, String password, String passwordConfirm);
  Future<void> login(String email, String password);
  Future<void> logout();
}

class AuthenticationRemote extends AuthenticationDatasource {
  @override
  Future<void> login(String email, String password) async {
    await FirebaseAuth.instance.signInWithEmailAndPassword(
        email: email.trim(), password: password.trim());
  }

  @override
  Future<void> register(
      String email, String password, String passwordConfirm) async {
    if (passwordConfirm == password) {
      await FirebaseAuth.instance
          .createUserWithEmailAndPassword(
              email: email.trim(), password: password.trim())
          .then((value) {
        Firestore().createUser(email.trim());
      });
    }
  }

  @override
  Future<void> logout() async {
    await FirebaseAuth.instance.signOut();
  }
}
```

Implementation:

- SignupScreen (signup.dart):

  Displays text fields for email, password, and password confirmation. A "Sign Up" button calls the register method in AuthenticationRemote.

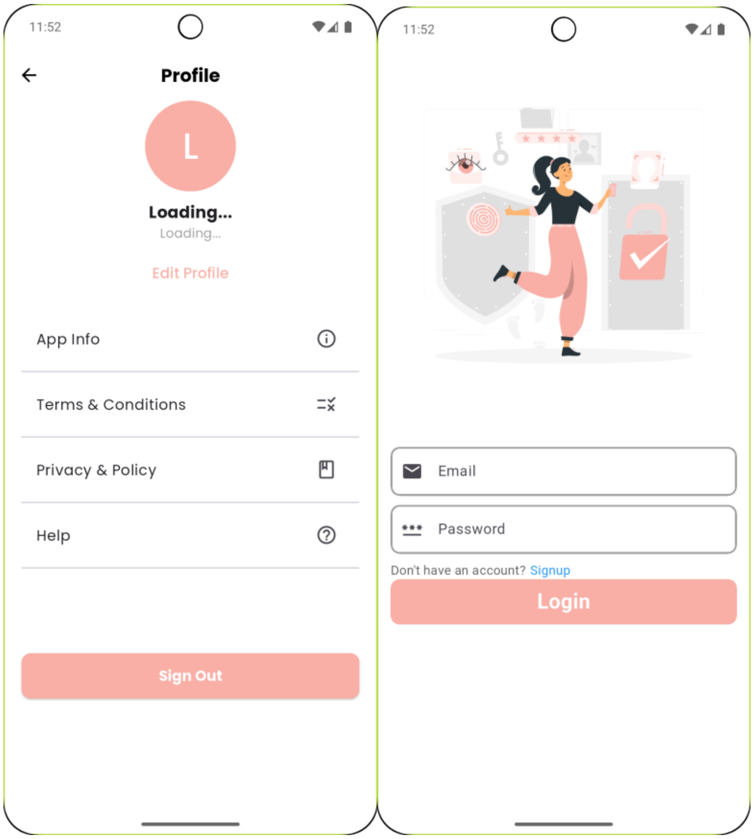
- AuthenticationRemote (auth_data.dart):

  Implements register(...), which checks if the two passwords match. Uses FirebaseAuth to create a new user account. Calls Firestore().createUser(...) to store user details in Firestore.

- Main_Page (main_page.dart):

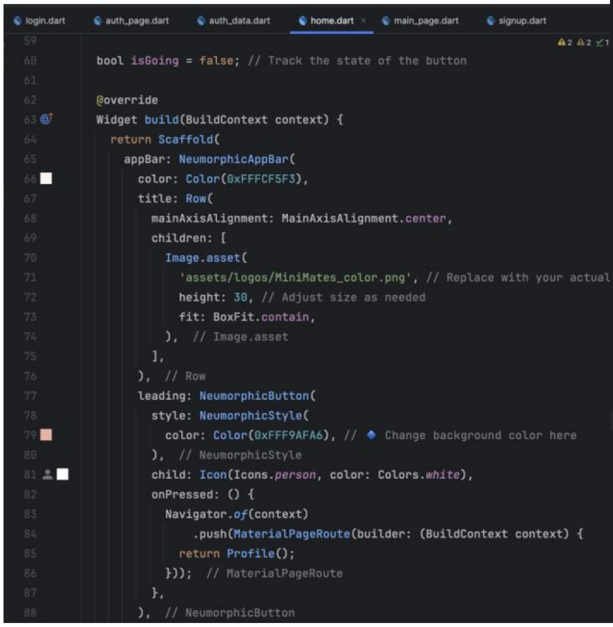
  Uses a StreamBuilder to watch FirebaseAuth changes. If a new user is successfully registered and logged in, it shows HomeScreen. Otherwise, it displays Auth_Page (login/signup toggle).

Implemented Feature: Sign-Out Functionality

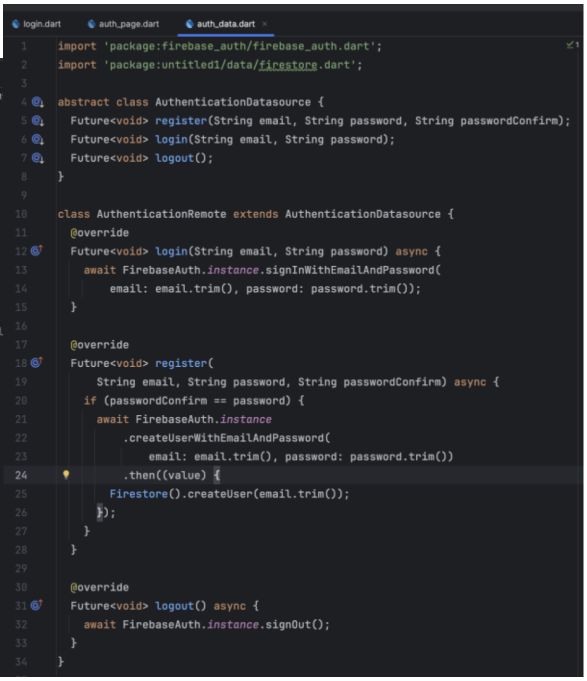


Sign-Out Flow

- HomeScreen (or similar) has a sign-out button → logout() in AuthenticationRemote → Firebase Auth sets user to null → Main_Page sees no user → goes back to Auth_Page.



```dart

bool isGoing = false; // Track the state of the button

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: NeumorphicAppBar(
      color: Color(0xFFFCF5F3),
      title: Row(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Image.asset(
            'assets/logos/MiniMates_color.png', // Replace with your actual
            height: 30, // Adjust size as needed
            fit: BoxFit.contain,
          ),  // Image.asset
        ],
      ),  // Row
      leading: NeumorphicButton(
        style: NeumorphicStyle(
          color: Color(0xFFF9AFA6), // 🔷 Change background color here
        ),  // NeumorphicStyle
        child: Icon(Icons.person, color: Colors.white),
        onPressed: () {
          Navigator.of(context)
              .push(MaterialPageRoute(builder: (BuildContext context) {
            return Profile();
          }));  // MaterialPageRoute
        },
      ),  // NeumorphicButton
```



```dart
import 'package:firebase_auth/firebase_auth.dart';
import 'package:untitled1/data/firestore.dart';

abstract class AuthenticationDatasource {
  Future<void> register(String email, String password, String passwordConfirm);
  Future<void> login(String email, String password);
  Future<void> logout();
}

class AuthenticationRemote extends AuthenticationDatasource {
  @override
  Future<void> login(String email, String password) async {
    await FirebaseAuth.instance.signInWithEmailAndPassword(
        email: email.trim(), password: password.trim());
  }

  @override
  Future<void> register(
      String email, String password, String passwordConfirm) async {
    if (passwordConfirm == password) {
      await FirebaseAuth.instance
          .createUserWithEmailAndPassword(
              email: email.trim(), password: password.trim())
          .then((value) {
        Firestore().createUser(email.trim());
      });
    }
  }

  @override
  Future<void> logout() async {
    await FirebaseAuth.instance.signOut();
  }
}
```

Implementation

HomeScreen (home.dart):

- Displays a "Logout" button.
- Calls AuthenticationRemote().logout() when tapped.

AuthenticationRemote (auth_data.dart):

- logout(...) calls FirebaseAuth.instance.signOut().
- Ends the user's session by setting the current user to null.

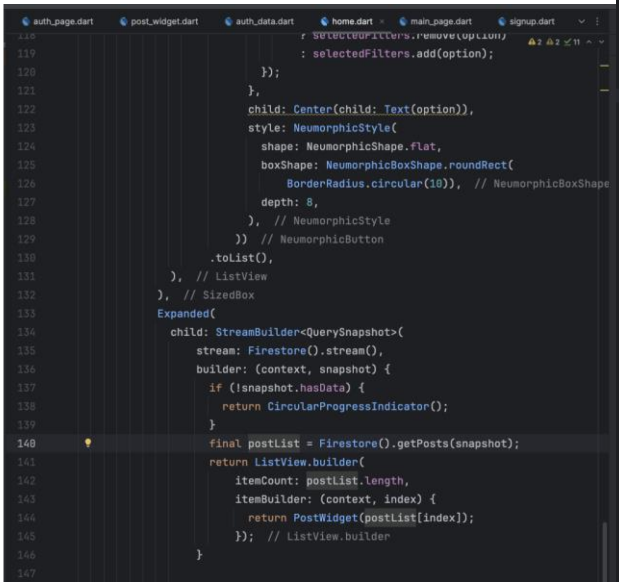
Main_Page (main_page.dart):

- Monitors FirebaseAuth state using StreamBuilder.
- If the user is logged out (currentUser == null), routes to Auth_Page; otherwise, keeps the user on HomeScreen.

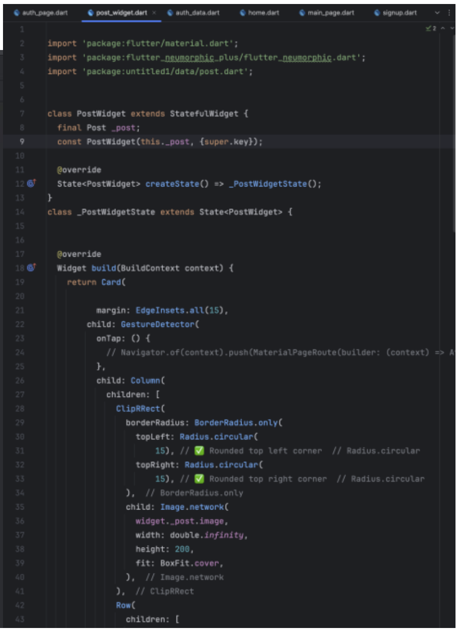
Implemented Feature: Main page list view implementation

Flow:

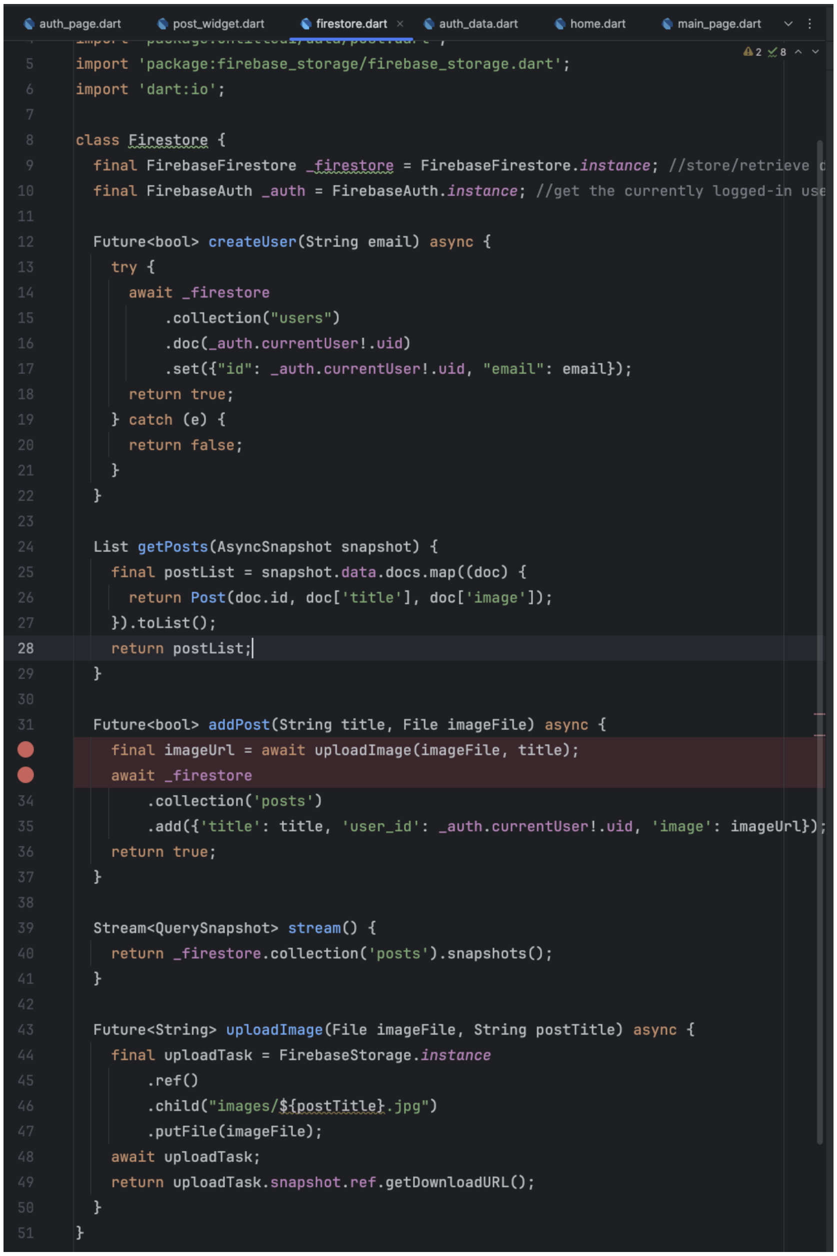
Main_Page → Fetches posts from Firestore → Displays a list of posts in ListView → Each post is shown in PostWidget



```
                      ? selectedFilters.remove(option)
                      : selectedFilters.add(option);
                });
              },
              child: Center(child: Text(option)),
              style: NeumorphicStyle(
                shape: NeumorphicShape.flat,
                boxShape: NeumorphicBoxShape.roundRect(
                    BorderRadius.circular(10)),  // NeumorphicBoxShape
                depth: 8,
              ),  // NeumorphicStyle
            ))  // NeumorphicButton
        .toList(),
  ),  // ListView
),  // SizedBox
Expanded(
  child: StreamBuilder<QuerySnapshot>(
      stream: Firestore().stream(),
      builder: (context, snapshot) {
        if (!snapshot.hasData) {
          return CircularProgressIndicator();
        }
        final postList = Firestore().getPosts(snapshot);
        return ListView.builder(
            itemCount: postList.length,
            itemBuilder: (context, index) {
              return PostWidget(postList[index]);
            });  // ListView.builder
      }
```



```

import 'package:flutter/material.dart';
import 'package:flutter_neumorphic_plus/flutter_neumorphic.dart';
import 'package:untitled1/data/post.dart';


class PostWidget extends StatefulWidget {
  final Post _post;
  const PostWidget(this._post, {super.key});

  @override
  State<PostWidget> createState() => _PostWidgetState();
}
class _PostWidgetState extends State<PostWidget> {


  @override
  Widget build(BuildContext context) {
    return Card(

          margin: EdgeInsets.all(15),
        child: GestureDetector(
          onTap: () {
            // Navigator.of(context).push(MaterialPageRoute(builder: (context) => A
          },
          child: Column(
            children: [
              ClipRRect(
                borderRadius: BorderRadius.only(
                  topLeft: Radius.circular(
                      15), // ✅ Rounded top left corner  // Radius.circular
                  topRight: Radius.circular(
                      15), // ✅ Rounded top right corner  // Radius.circular
                ),  // BorderRadius.only
                child: Image.network(
                  widget._post.image,
                  width: double.infinity,
                  height: 200,
                  fit: BoxFit.cover,
                ),  // Image.network
              ),  // ClipRRect
              Row(
                children: [
```



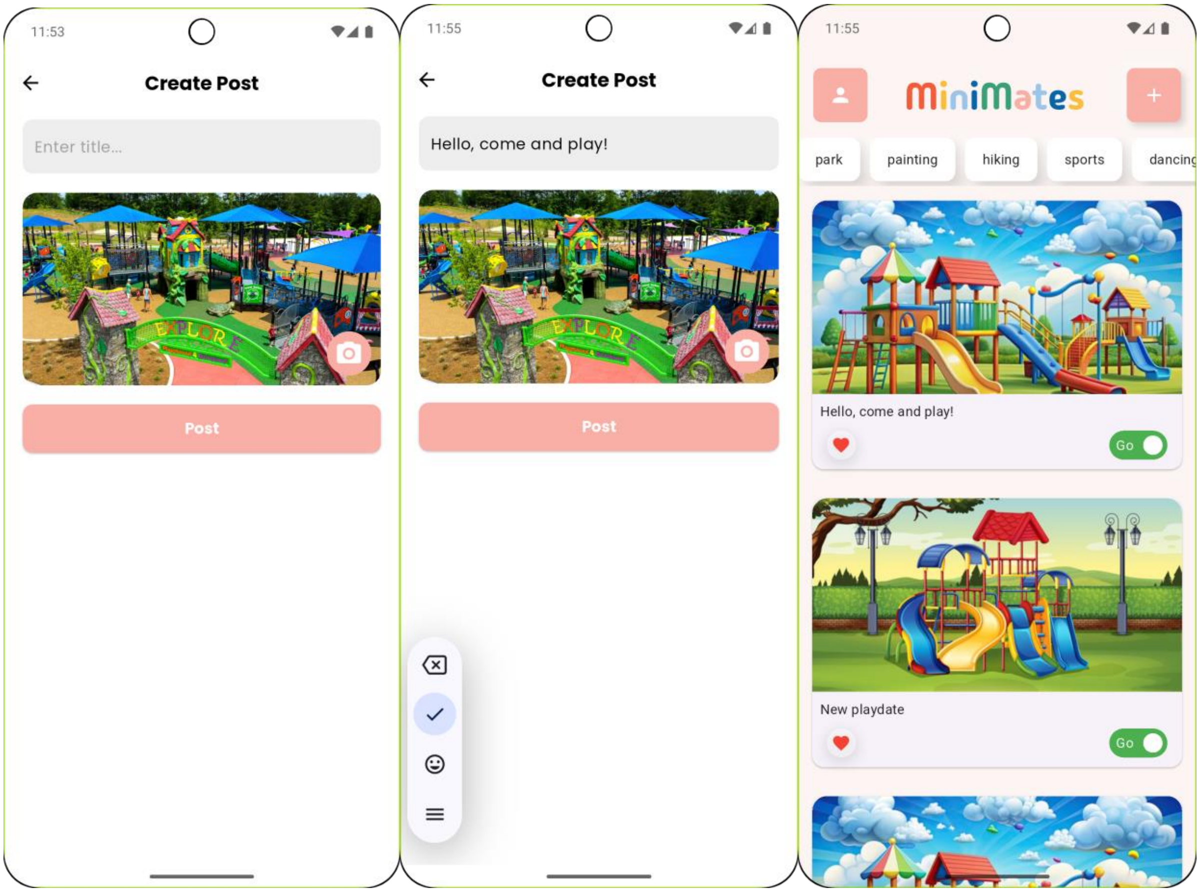
```dart
import 'package:firebase_storage/firebase_storage.dart';
import 'dart:io';

class Firestore {
  final FirebaseFirestore _firestore = FirebaseFirestore.instance; //store/retrieve d
  final FirebaseAuth _auth = FirebaseAuth.instance; //get the currently logged-in use

  Future<bool> createUser(String email) async {
    try {
      await _firestore
          .collection("users")
          .doc(_auth.currentUser!.uid)
          .set({"id": _auth.currentUser!.uid, "email": email});
      return true;
    } catch (e) {
      return false;
    }
  }

  List getPosts(AsyncSnapshot snapshot) {
    final postList = snapshot.data.docs.map((doc) {
      return Post(doc.id, doc['title'], doc['image']);
    }).toList();
    return postList;
  }

  Future<bool> addPost(String title, File imageFile) async {
    final imageUrl = await uploadImage(imageFile, title);
    await _firestore
        .collection('posts')
        .add({'title': title, 'user_id': _auth.currentUser!.uid, 'image': imageUrl});
    return true;
  }

  Stream<QuerySnapshot> stream() {
    return _firestore.collection('posts').snapshots();
  }

  Future<String> uploadImage(File imageFile, String postTitle) async {
    final uploadTask = FirebaseStorage.instance
        .ref()
        .child("images/${postTitle}.jpg")
        .putFile(imageFile);
    await uploadTask;
    return uploadTask.snapshot.ref.getDownloadURL();
  }
}
```

Implementation:

- Home_Page (home.dart)
  - Uses a StreamBuilder to listen for Firestore updates and dynamically update the UI.
  - Fetches posts and passes them to a ListView.builder.
- PostWidget (post_widget.dart)
  - Displays individual posts inside a Card with UI elements.
  - Loads images and text from Firestore.

- Firestore (firestore.dart)
    - Contains methods to fetch posts using Firestore’s snapshots() stream.

Implemented Feature: Post implementation



11:53

Create Post

Enter title...

Post

11:55

Create Post

Hello, come and play!

Post

11:55

MiniMates

park painting hiking sports dancing

Hello, come and play!

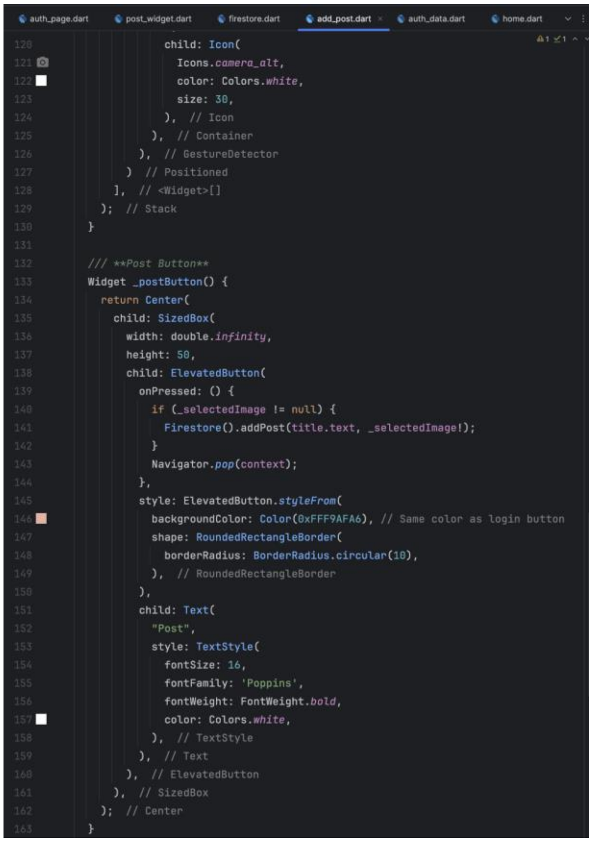
Go

New playdate

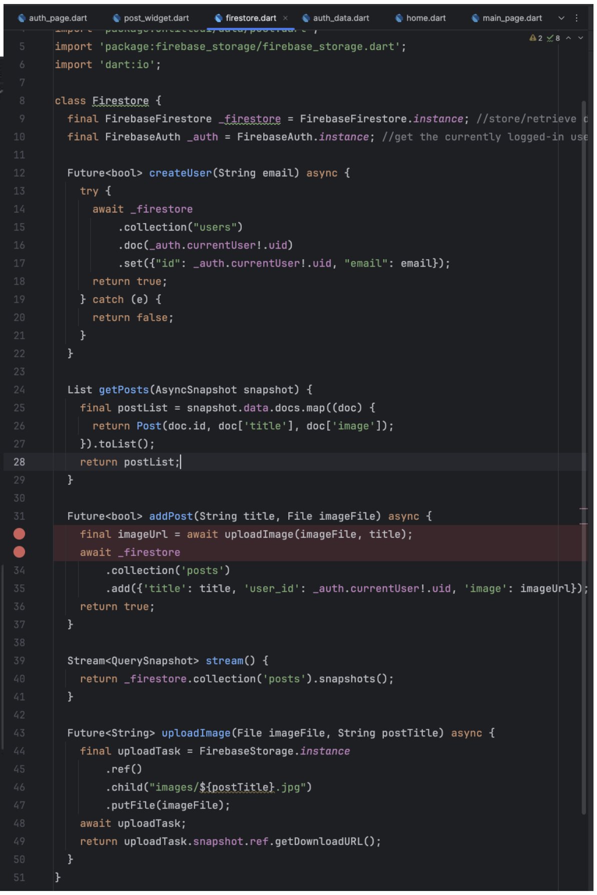
Go

Flow:

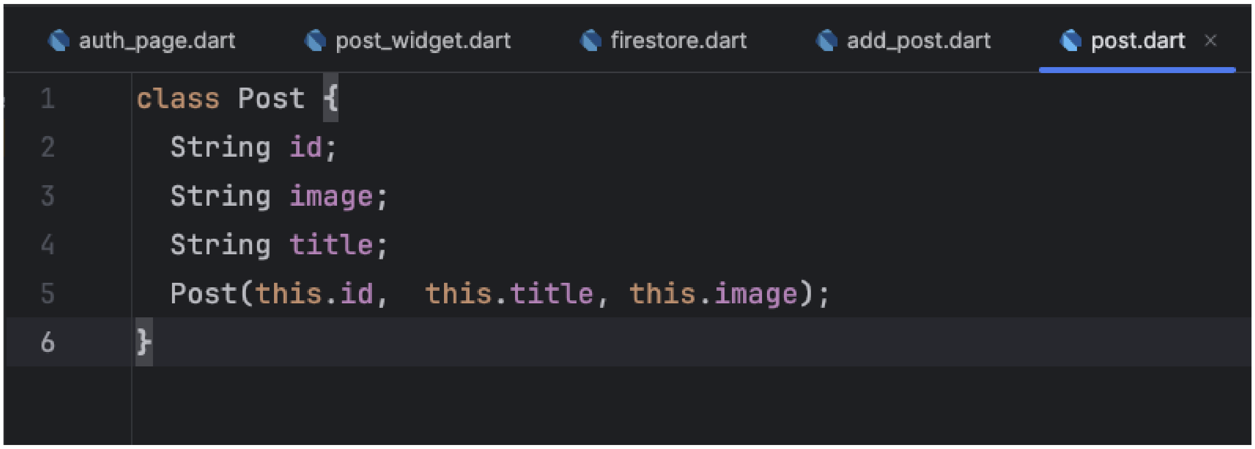
User Opens AddPost → Enters Post Title → Uploads Image → Clicks Submit → Post is saved to Firestore → Main_Page updates with new post



```dart
              child: Icon(
                Icons.camera_alt,
                color: Colors.white,
                size: 30,
              ),  // Icon
            ),  // Container
          ),  // GestureDetector
        )  // Positioned
      ],  // <Widget>[]
    );  // Stack
  }

  /// **Post Button**
  Widget _postButton() {
    return Center(
      child: SizedBox(
        width: double.infinity,
        height: 50,
        child: ElevatedButton(
          onPressed: () {
            if (_selectedImage != null) {
              Firestore().addPost(title.text, _selectedImage!);
            }
            Navigator.pop(context);
          },
          style: ElevatedButton.styleFrom(
            backgroundColor: Color(0xFFF9AFA6), // Same color as login button
            shape: RoundedRectangleBorder(
              borderRadius: BorderRadius.circular(10),
            ),  // RoundedRectangleBorder
          ),
          child: Text(
            "Post",
            style: TextStyle(
              fontSize: 16,
              fontFamily: 'Poppins',
              fontWeight: FontWeight.bold,
              color: Colors.white,
            ),  // TextStyle
          ),  // Text
        ),  // ElevatedButton
      ),  // SizedBox
    );  // Center
  }
```



```dart
import 'package:firebase_storage/firebase_storage.dart';
import 'dart:io';

class Firestore {
  final FirebaseFirestore _firestore = FirebaseFirestore.instance; //store/retrieve d
  final FirebaseAuth _auth = FirebaseAuth.instance; //get the currently logged-in use

  Future<bool> createUser(String email) async {
    try {
      await _firestore
          .collection("users")
          .doc(_auth.currentUser!.uid)
          .set({"id": _auth.currentUser!.uid, "email": email});
      return true;
    } catch (e) {
      return false;
    }
  }

  List getPosts(AsyncSnapshot snapshot) {
    final postList = snapshot.data.docs.map((doc) {
      return Post(doc.id, doc['title'], doc['image']);
    }).toList();
    return postList;
  }

  Future<bool> addPost(String title, File imageFile) async {
    final imageUrl = await uploadImage(imageFile, title);
    await _firestore
        .collection('posts')
        .add({'title': title, 'user_id': _auth.currentUser!.uid, 'image': imageUrl});
    return true;
  }

  Stream<QuerySnapshot> stream() {
    return _firestore.collection('posts').snapshots();
  }

  Future<String> uploadImage(File imageFile, String postTitle) async {
    final uploadTask = FirebaseStorage.instance
        .ref()
        .child("images/${postTitle}.jpg")
        .putFile(imageFile);
    await uploadTask;
    return uploadTask.snapshot.ref.getDownloadURL();
  }
}
```



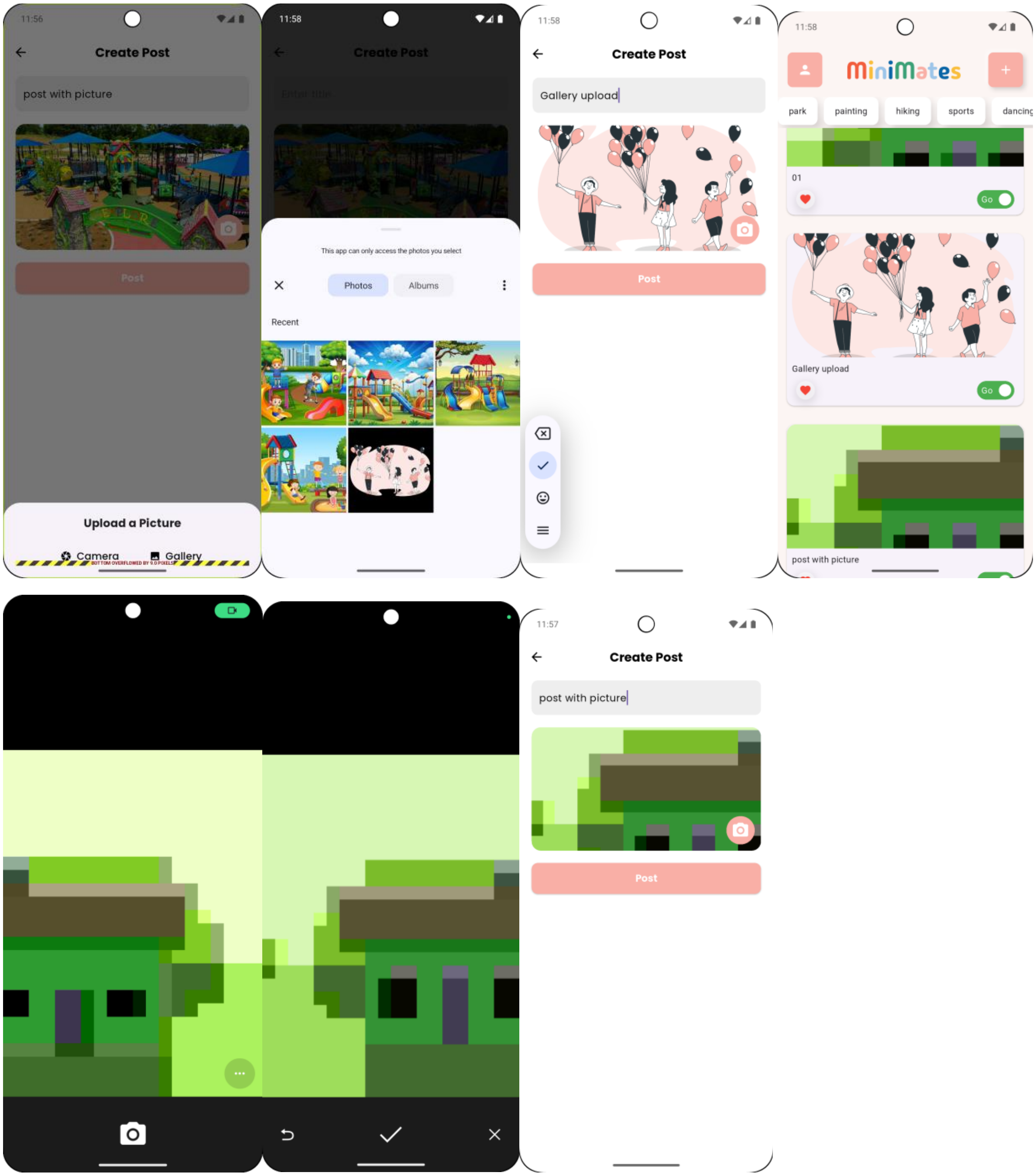
```dart
class Post {
  String id;
  String image;
  String title;
  Post(this.id,  this.title, this.image);
}
```

Implementation:

- AddPost (add_post.dart)
  - UI for users to input a title and upload an image.
  - Calls Firestore().addPost(...) to save the post.
- Post (post.dart)
  - Defines a Post class to structure post data.

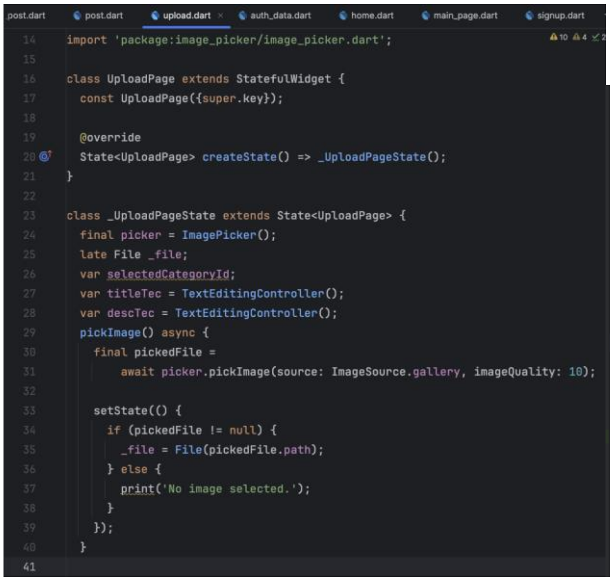
- Firestore (firestore.dart)
  - addPost(...) uploads the post’s image to Firebase Storage.
  - Saves the post’s title and image URL in Firestore.
  - Provides a stream() method to retrieve posts dynamically.

Implemented Feature: Image upload and Camera implementation

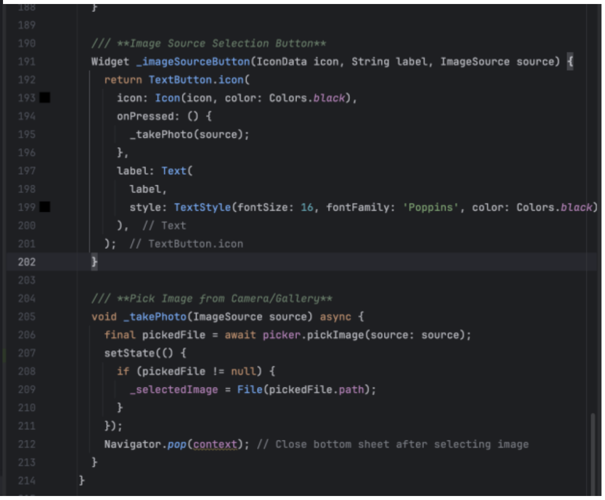


Flow:

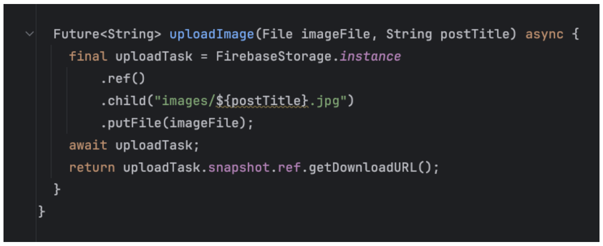
User Opens UploadPage → Taps Upload Button → Chooses Image from Gallery/Camera → Image Preview is Displayed → Image is Uploaded to Firebase Storage



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

class UploadPage extends StatefulWidget {
  const UploadPage({super.key});

  @override
  State<UploadPage> createState() => _UploadPageState();
}

class _UploadPageState extends State<UploadPage> {
  final picker = ImagePicker();
  late File _file;
  var selectedCategoryId;
  var titleTec = TextEditingController();
  var descTec = TextEditingController();
  pickImage() async {
    final pickedFile =
        await picker.pickImage(source: ImageSource.gallery, imageQuality: 10);

    setState(() {
      if (pickedFile != null) {
        _file = File(pickedFile.path);
      } else {
        print('No image selected.');
      }
    });
  }

```



```dart
  }

  /// **Image Source Selection Button**
  Widget _imageSourceButton(IconData icon, String label, ImageSource source) {
    return TextButton.icon(
      icon: Icon(icon, color: Colors.black),
      onPressed: () {
        _takePhoto(source);
      },
      label: Text(
        label,
        style: TextStyle(fontSize: 16, fontFamily: 'Poppins', color: Colors.black)
      ),  // Text
    );  // TextButton.icon
  }

  /// **Pick Image from Camera/Gallery**
  void _takePhoto(ImageSource source) async {
    final pickedFile = await picker.pickImage(source: source);
    setState(() {
      if (pickedFile != null) {
        _selectedImage = File(pickedFile.path);
      }
    });
    Navigator.pop(context); // Close bottom sheet after selecting image
  }
}
```



```dart
  Future<String> uploadImage(File imageFile, String postTitle) async {
    final uploadTask = FirebaseStorage.instance
        .ref()
        .child("images/${postTitle}.jpg")
        .putFile(imageFile);
    await uploadTask;
    return uploadTask.snapshot.ref.getDownloadURL();
  }
}
```

Implementation:

- **UploadPage (upload.dart)**
  - Uses ImagePicker to select an image from the gallery.
  - Displays the selected image in a preview box.
  - Saves the image to Firebase Storage using Firestore().uploadImage(...).
- **Firestore (firestore.dart)**
  - uploadImage(...) uploads the image to Firebase Storage.
  - Generates and returns a download URL for Firestore storage.
- **Camera(add_post.dart)**
  - OnPress _imageSourceButton will activate the _takePhoto method to take photos using the phone's camera

Work Log

Week 05(Feb 09 – Feb 16)

Date	Number of Hours	Description of work done
Feb 09, 2025	2	Doing some design of the login and signup page and searching for the appropriate videos for my login and sign-up page.
Feb 10, 2025	4	Building the Login Page UI, followed a YouTube video, and also wrote the frontend.
Feb 11, 2025	3.5	Finish the sign-up page UI and wrote the frontend using flutter.
Feb 12, 2025	2	I've learned some new things about Firebase, I've never used Firebase before, so this is a new tool for me.
Feb 12, 2025	3	After checking some videos and knowledge, I decided to use Firebase instead of the MongoDB and NodeJS.
Feb 14, 2025	4	Creating my firebase account and set up the firebase structure.
Feb 17, 2025	6	Connecting my project with the firebase, at first, I cannot get the auto generated file from firebase, but then I tried a lot of times and checked from some videos and websites, I figured it out and connected the database.

Week 05(Feb 17 – Feb 23)

Date	Number of Hours	Description of work done
Feb 17, 2025	4	Making the main page is not easy, I changed the design

		of the main page, since I want to implement the post first, so the most important thing is to display the list view from the database.
Feb 18, 2025	4	The list view is showing but the images are all the same, so I found that I need to integrate the post functionality to change / post the different images by user. I made the post page UI today.
Feb 19, 2025	3	I implement the post feature, but the image upload feature is still not working, I tried several times, but it's still not working, I can only take picture using the camera.
Feb 20, 2025	3.5	I cannot pick image from the phone's gallery, it took me a lot of time to figure this out, then finally, I found that I can directly download the image form phone's google, and save it to the gallery, and then my image upload works!!! So happy!
Feb 22, 2025	2	Working on the midterm report.
Feb 22, 2025	0.5	Update the worklog.
Feb 22, 2025	3	Prepare the Gantt chart, video setup instructions, and write my readme file.
Feb 23, 2025	4	I got some problem with my login page and the onboarding page. From the onboarding page, I cannot navigate to the login page, but if I remove the

		onboarding page, my signup is not working…..
Feb 24, 2025	4	Recording and clip the video.
Feb 24, 2025	2	Continue figuring out my problems and make it ready to submit.

References

https://www.tinytiestogether.com/?srsltid=AfmBOopHHFh44O7fvsY-NvQpw8PlkgaN391b08W6u6sUNia3dTrGj6zO

https://www.myplaydateapp.com/

https://www.golfburnaby.ca/search/results?keys=kids

https://www.youtube.com/watch?v=6af6qgziYfo&list=PLtIU0BH0pkKpitsp5jzt-yDAoXAFBkcPb&index=15