MiniMates

A **Playdate Platform** for Parents

to Explore Activities

Final Report

4495 Applied Research -- Section 001

Becky Jin 300374335

Apr 13, 2025

1. Introduction

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.

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.

2. Project Summary

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.

MiniMates is a full-stack platform designed to:

- Allow parents to post and discover playdate events
- Provide **AI-powered (Gemini)** event recommendations based on users' preferences choose in the profile
- Enable customized tags options for activities based on activity types
- Offer secure user authentication & profile management
- Support easy event modifications & cancellations
- Chatting with the post author or friends
- Posts management including editing and deleting
- Comment, reply, and like in a nested format
- Add their own events with geolocation
- View posts with real-time joined participant counts

Technology Stack

- **Front-end:** Flutter (Dart)
- **Back-end:** Firebase
- **Database:** Firebase
- **AI-powered Recommendations:** Gemini
- **Operating System:** macOS

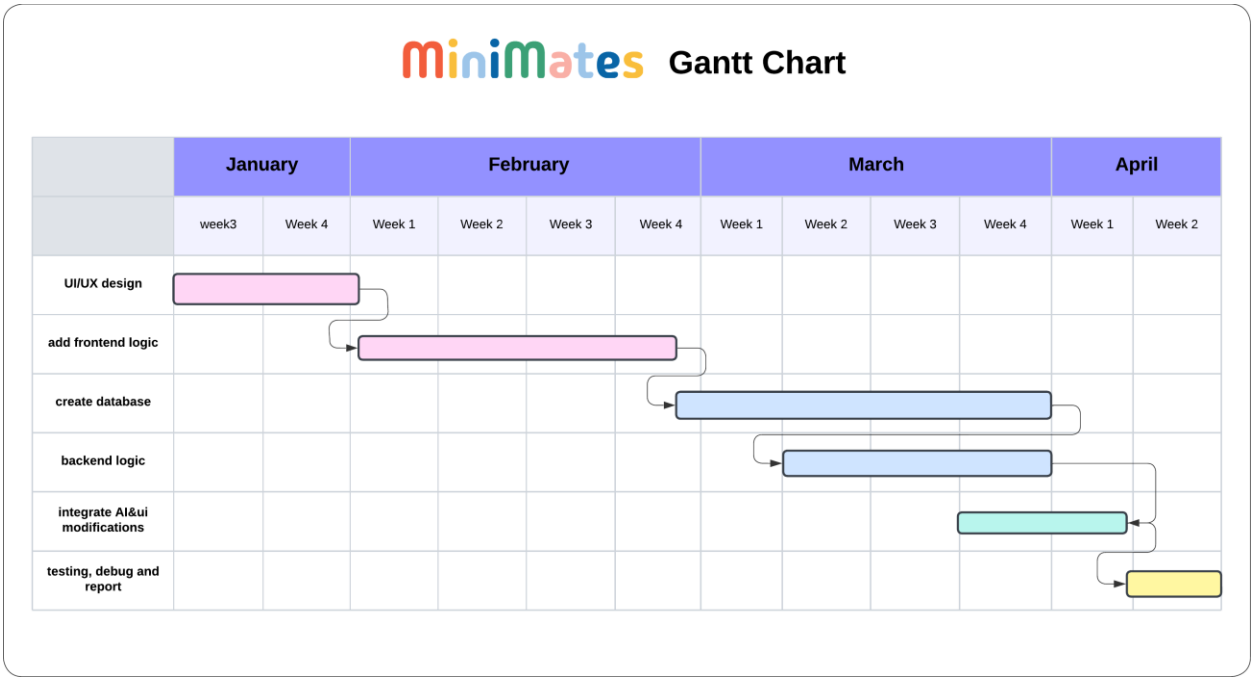
3. Changes to Proposal

Original	Final	Reason
Kids Matching Feature	No kids matching	Privacy concerns
Text-based locations	Switched to using geolocation (lat/lng) and Google Maps	Improves accuracy and context
No comment section	Added threaded replies and likes on comments	Better social interaction
No chatting	Added chatting feature	Better connection
Data analysis	No analysis and Integrated Gemini AI to enhance suggestions	The hardness of preparing the data, and do the data analysis

4. Project Completion Timeline

Phase	Dates	Milestones	Deliverables
Planning & Setup	**2.9-2.25**	Started project setup and authentication	- Project proposal outline - Firebase connected and tested - Login and signup page
UI Design	**2.26-3.3**	- BottomNavigationBar - Designed home, add post, and profile screens	- Functional app navigation - UI layout for Home/Add/Profile pages
Authentication & User Data	**3.4-3.7**	- Sign In & Sign Up forms - Stored user info in Firestore - Profile loading with real user data	- Working Sign Up & Sign In - Form validations
Add Post & Firestore Integration	**3.8-3.14**	- Enabled join/unjoin of events - Created real-time joined counter using StreamBuilder	- Add post page
Joined Events + Real-Time Updates	**3.15-3.18**	Integrated Gemini AI for "May Like"	- We worked on Gemini model prompt, parsing post IDs, and marking "✨ Recommended".

Comments & Replies System	**3.19-3.24**	- Built nested comments and replies - Enabled real-time comment streaming - Like system for both comments and replies	- Real-time comment/reply system
Gemini Integration	**3.25-3.29**	- Displayed recommended posts on Home	- callGeminiModel() logic - “May Like” button with badge - Auto-trigger recommendations
Profile Features & Settings	**3.30-4.2**	- Built MyPosts screen with long-press edit/delete - Policy & Contact added in Profile	- MyPosts page - Image upload to Storage - Policy UI with app logo
Final Polish & Testing	**4.3-4.10**	- Bug fixes & UI clean-up - work logs - user guide - Final screenshots and report writing	- Final working app build - Research report (15+ pages) - Presentation slides - user guide



5. Implemented Features

5.1 AI-Powered Post Recommendations Feature

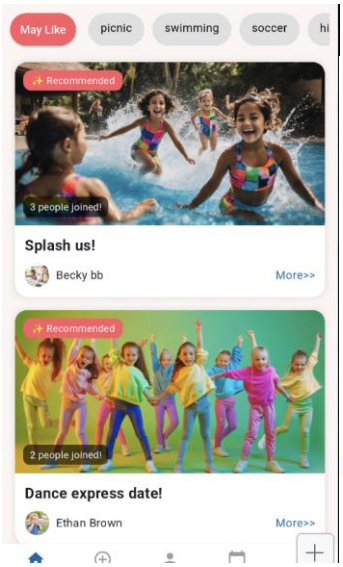
Gemini-powered personalized suggestions based on user preferences.

- created a method called callGeminiModel() that sends the user's preferences and available post data to Gemini and receives a list of recommended post IDs. After

receiving the response, I use setState() to update the UI in real time, enabling a smooth and dynamic user experience.

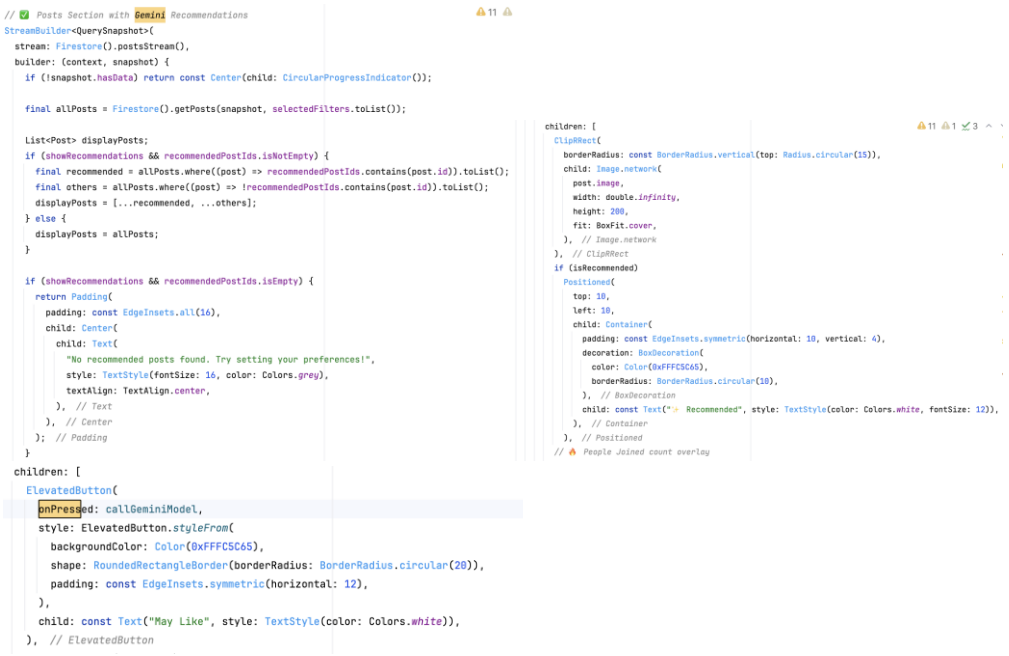
- Inside initState(), I call _loadUserProfile(), which checks if the user has saved preferences and then triggers the Gemini model accordingly.
- Additionally, when the user taps the **"May Like"** button, the app calls callGeminiModel() again. It updates a list called displayPosts, which prioritizes showing recommended posts (with a " ✨ Recommended" badge) at the top, followed by all other posts.

Screenshots:




```
Future<void> callGeminiModel() async {
  final model = GenerativeModel(
    model: 'gemini-2.5-pro-exp-03-25',
    apiKey: 'AIzaSyBwmZTKKRECKZwT3SjyEzDsZF7Uk2yq_T0',
  );
  final postList = await Firestore().postList();
  final user = await Firestore().getMySelf();
  final postListJson = postList.map((post) => post.toMap()).toList();
  final preferenceJson = user.preferences;
  final prompt =
      "I have a list of posts in JSON format: $postListJson and my preference tags are "
      "$preferenceJson. Please recommend the posts that I like and output their ID in a list only";

  final response = await model.generateContent([Content.text(prompt)]);
  final text = response.text ?? "";
  final ids = RegExp(r'\w{20,}').allMatches(text).map((match) => match.group(0)!).toList();

  setState(() {
    recommendedPostIds = ids;
    showRecommendations = true;
  });

  print("🔮 Recommended Post IDs: $recommendedPostIds");
}
```

```
if (userData.containsKey('preferences') && userData['preferences'].isNotEmpty) {
  callGeminiModel();
}
```



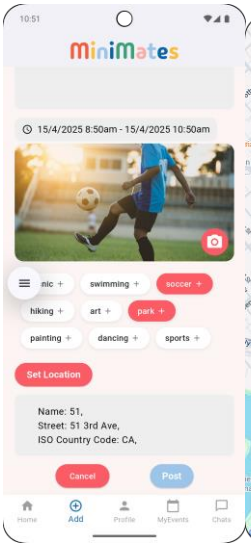
```
// ✅ Posts Section with Gemini Recommendations
StreamBuilder<QuerySnapshot>(
  stream: Firestore().postsStream(),
  builder: (context, snapshot) {
    if (!snapshot.hasData) return const Center(child: CircularProgressIndicator());

    final allPosts = Firestore().getPosts(snapshot, selectedFilters.toList());

    List<Post> displayPosts;
    if (showRecommendations && recommendedPostIds.isNotEmpty) {
      final recommended = allPosts.where((post) => recommendedPostIds.contains(post.id)).toList();
      final others = allPosts.where((post) => !recommendedPostIds.contains(post.id)).toList();
      displayPosts = [...recommended, ...others];
    } else {
      displayPosts = allPosts;
    }

    if (showRecommendations && recommendedPostIds.isEmpty) {
      return Padding(
        padding: const EdgeInsets.all(16),
        child: Center(
          child: Text(
            "No recommended posts found. Try setting your preferences!",
            style: TextStyle(fontSize: 16, color: Colors.grey),
            textAlign: TextAlign.center,
          ),  // Text
        ),  // Center
      );  // Padding
    }
children: [
  ElevatedButton(
    onPressed: callGeminiModel,
    style: ElevatedButton.styleFrom(
      backgroundColor: Color(0xFFFC5C65),
      shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(20)),
      padding: const EdgeInsets.symmetric(horizontal: 12),
    ),
    child: const Text("May Like", style: TextStyle(color: Colors.white)),
  ),  // ElevatedButton
```

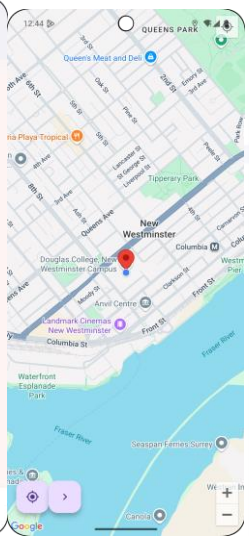
```
children: [
  ClipRRect(
    borderRadius: const BorderRadius.vertical(top: Radius.circular(15)),
    child: Image.network(
      post.image,
      width: double.infinity,
      height: 200,
      fit: BoxFit.cover,
    ),  // Image.network
  ),  // ClipRRect
  if (isRecommended)
    Positioned(
      top: 10,
      left: 10,
      child: Container(
        padding: const EdgeInsets.symmetric(horizontal: 10, vertical: 4),
        decoration: BoxDecoration(
          color: Color(0xFFFC5C65),
          borderRadius: BorderRadius.circular(10),
        ),  // BoxDecoration
        child: const Text("⭐ Recommended", style: TextStyle(color: Colors.white, fontSize: 12)),
      ),  // Container
    ),  // Positioned
  // 🔥 People Joined count overlay
```

5.2 Add Post with Geolocation Feature

- To implement the geolocation feature, I used the **google_maps_flutter** package to allow users to select a location on a map, and the **geocoding package** to convert latitude and longitude into a readable address that gets saved along with the post.
- Getting Lat/Lng from Google Map Selection (in AddPost)
- Saving Post to Firestore with Location Info
- Storing Location in TextEditingController
- UI Button to Trigger Location Selection

Screenshots:






```
SizedBox(
  child: ElevatedButton(
    onPressed: () {
      Navigator.of(context)
          .push(MaterialPageRoute(builder: (BuildContext context) => GoogleMapFlutter()))
          .then((location) {
        longitude = location.longitude;
        latitude = location.latitude;
        placemarkFromCoordinates(latitude!, longitude!).then((placemarks) {
          var output = 'No results found.';
          if (placemarks.isNotEmpty) {
            output = placemarks[0].toString();
          }
          setState(() {
            address.text = output;
          });
        });
        setState(() {
          longitude = location.longitude;
          latitude = location.latitude;
        });
      });
    },
```

```
Future<DocumentReference> addPost(
    String title,
    String desc,
    File imageFile,
    List<String> tags,
    double latitude,
    double longitude,
    String address,
    int spot,
    DateTime startTime,
    DateTime endTime,
    ) async {
  final imageUrl = await uploadImage(imageFile, title);
  return await _firestore.collection('posts').add({
    'title': title,
    'user_id': _auth.currentUser!.uid,
    'image': imageUrl,
    'description': desc,
    'tags': tags,
    'timestamp': FieldValue.serverTimestamp(),
    'latitude': latitude,
    'longitude': longitude,
    'address': address,
    'spot': spot,
    'start_time': startTime,
    'end_time': endTime,
  });
}
```

```
    body: GoogleMap(
      initialCameraPosition:
          CameraPosition(target: myCurrentPosition, zoom: 15),
      onMapCreated: (controller) {
        mapController = controller;
        // addMarker('Current Location', myCurrentPosition);
      },
      markers: {selectedMarker},
      onTap: (latin) {
        selectLocation(latin);
      },
      myLocationButtonEnabled: true,
      myLocationEnabled: true,
    ),  // GoogleMap
  );  // Scaffold
}
```

```
    final address = TextEditingController();
    double? longitude;
    double? latitude;
```

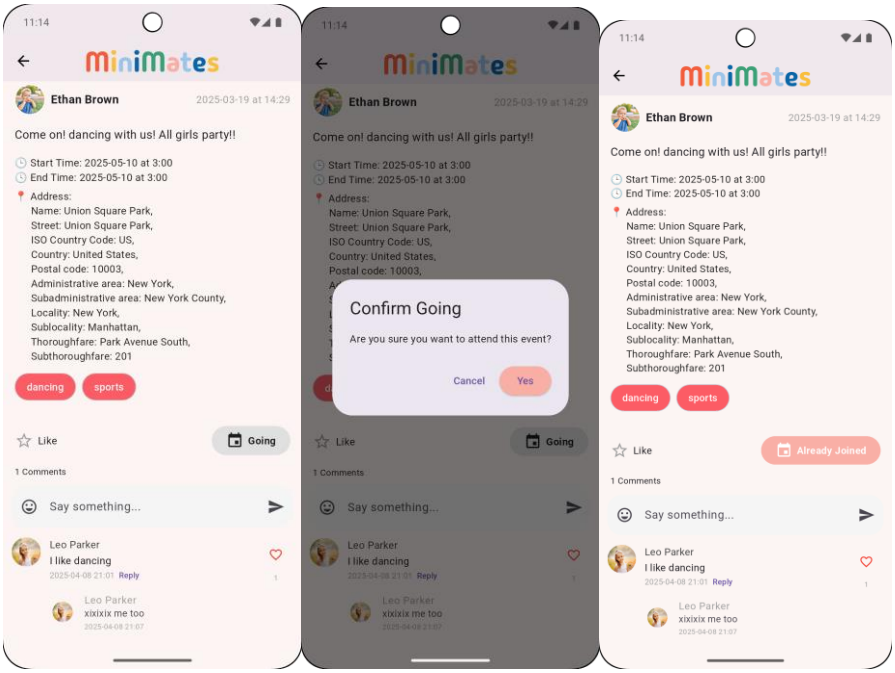
```
selectLocation(LatLng position) async {
  // var marker = Marker(markerId: MarkerId(markerName), position: position);
  // _markers.putIfAbsent(markerName, () => marker);
  myCurrentPosition = position;
  selectedMarker =
      Marker(markerId: MarkerId('Selected Location'), position: position);
  setState(() {});
}

Future<Position> getCurrentLocation() async {
  var permission = await Geolocator.checkPermission();
  if (permission == LocationPermission.denied) {
    permission = await Geolocator.requestPermission();
    if (permission == LocationPermission.denied) {
      return Future.error('Location permission denied');
    }
  }
  return await Geolocator.getCurrentPosition();
}
```

```
    style: ElevatedButton.styleFrom(
      backgroundColor: Color(0xFFFC5C65),
      shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(20)),
      padding: EdgeInsets.symmetric(horizontal: 20, vertical: 3),
    ),
    child: Text("Set Location", style: TextStyle(fontSize: 16, fontWeight: FontWeight.bold, color: Colors.white)),
  ),  // ElevatedButton
```

5.3 Joined Event Count (Real-time) Feature

- JoinCountStream() provides a live stream of people who joined an event
- Display using StreamBuilder in the home page

Screenshot:




```dart
//provides a live stream of people who joined an event
Stream<int> joinedCountStream(String postId) {
  return _firestore
      .collection('users_posts')
      .where('post_id', isEqualTo: postId)
      .where('is_going', isEqualTo: true)
      .snapshots()
      .map((snapshot) => snapshot.docs.length);
}
```

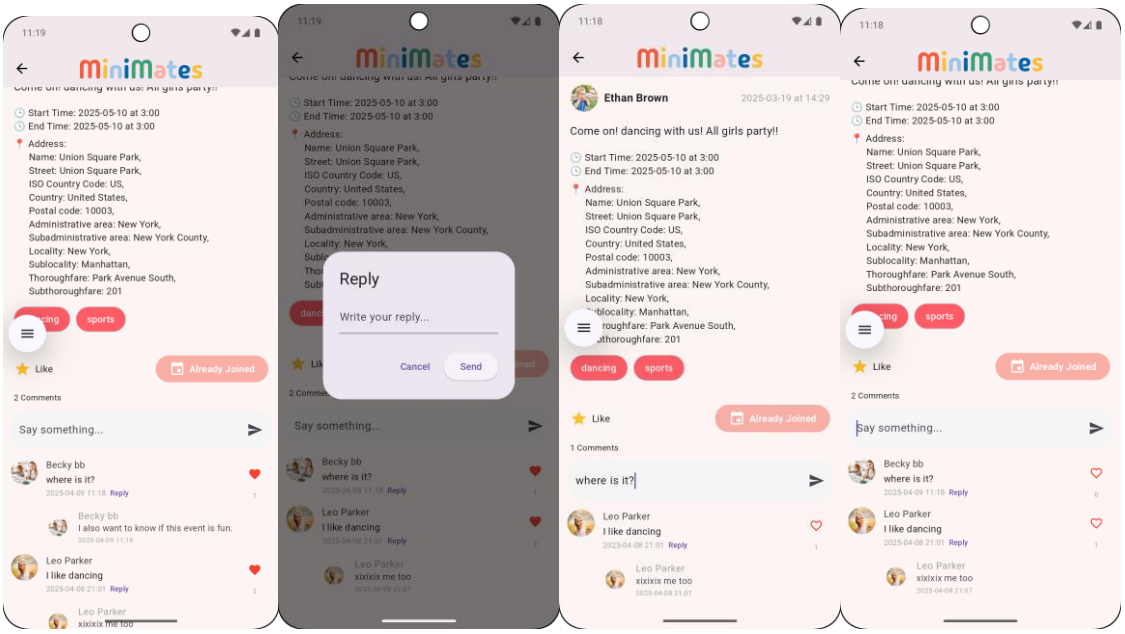
```dart
child: StreamBuilder<int>(
  stream: Firestore().joinedCountStream(post.id),
  builder: (context, snapshot) {
    if (!snapshot.hasData) return const SizedBox.shrink();
    final count = snapshot.data!;
    return Container(
      padding: const EdgeInsets.symmetric(horizontal: 8, vertical: 4),
      decoration: BoxDecoration(
        color: Colors.black.withOpacity(0.6),
        borderRadius: BorderRadius.circular(8),
      ), // BoxDecoration
      child: Text(
        "$count people joined!",
        style: const TextStyle(color: Colors.white, fontSize: 12),
      ), // Text
    ); // Container
  },
), // StreamBuilder
```

5.4 Comment System with Replies and Likes Features

- addComment() -- Adds a new comment to the post's comments subcollection.
- getComments() -- Provides a stream of all comments (real-time updates)User author badge
- addReply() -- Adds a reply to a specific comment
- getReplies() -- Stream for replies under a comment

- toggleCommentLike(), toggleReplyLike() -- Toggle like/unlike for comments or replies
- StreamBuilder shows all comments, ‘Reply’ button for the reply dialogue. Nested under each comment using getReplies() stream

Screenshots:



Firestore:

```dart
Future<void> addComment(String postId, String content, bool isAuthor) async {
  final user = FirebaseAuth.instance.currentUser!;
  final userData = await getUserById(user.uid);

  await FirebaseFirestore.instance
      .collection('posts')
      .doc(postId)
      .collection('comments')
      .add({
    'user_id': user.uid,
    'user_name': userData.userName,
    'user_photo': userData.profilePicture,
    'content': content,
    'timestamp': Timestamp.now(),
    'is_author': isAuthor,
    'likes': [],
  });
}

Stream<QuerySnapshot> getComments(String postId) {
  return FirebaseFirestore.instance
      .collection('posts')
      .doc(postId)
      .collection('comments')
      .orderBy('timestamp', descending: true)
      .snapshots();
}
```

```dart
Future<void> addReply({
  required String postId,
  required String commentId,
  required String content,
}) async {
  final user = FirebaseAuth.instance.currentUser!;
  final userData = await getUserById(user.uid);

  await _firestore
      .collection('posts')
      .doc(postId)
      .collection('comments')
      .doc(commentId)
      .collection('replies')
      .add({
    'user_id': user.uid,
    'user_name': userData.userName,
    'user_photo': userData.profilePicture,
    'content': content,
    'timestamp': FieldValue.serverTimestamp(),
    'likes': [],
  });
}
Stream<QuerySnapshot> getReplies(String postId, String commentId) {
  return _firestore
      .collection('posts')
      .doc(postId)
      .collection('comments')
      .doc(commentId)
      .collection('replies')
      .orderBy('timestamp', descending: false)
      .snapshots();
}
```

```dart
Future<void> toggleCommentLike(String postId, String commentId) async {
  final userId = FirebaseAuth.instance.currentUser!.uid;
  final commentRef = FirebaseFirestore.instance
      .collection('posts')
      .doc(postId)
      .collection('comments')
      .doc(commentId);

  final doc = await commentRef.get();
  List likes = doc['likes'] ?? [];

  if (likes.contains(userId)) {
    await commentRef.update({'likes': FieldValue.arrayRemove([userId])});
  } else {
    await commentRef.update({'likes': FieldValue.arrayUnion([userId])});
  }
}

Future<void> toggleReplyLike(String postId, String commentId, String replyId) async {
  final userId = FirebaseAuth.instance.currentUser!.uid;
  final replyRef = FirebaseFirestore.instance
      .collection('posts')
      .doc(postId)
      .collection('comments')
      .doc(commentId)
      .collection('replies')
      .doc(replyId);

  final doc = await replyRef.get();
  List likes = doc['likes'] ?? [];

  if (likes.contains(userId)) {
    await replyRef.update({'likes': FieldValue.arrayRemove([userId])});
  } else {
    await replyRef.update({'likes': FieldValue.arrayUnion([userId])});
  }
}
```

```dart
void _showReplyDialog(String postId, String commentId) {
  final replyController = TextEditingController();

  showDialog(
    context: context,
    builder: (context) => AlertDialog(
      title: Text("Reply"),
      content: TextField(
        controller: replyController,
        decoration: InputDecoration(hintText: 'Write your reply...'),
      ),  // TextField
      actions: [
        TextButton(
          child: Text("Cancel"),
          onPressed: () => Navigator.pop(context),
        ),  // TextButton
        ElevatedButton(
          child: Text("Send"),
          onPressed: () async {
            final replyText = replyController.text.trim();

            // Always close the dialog
            Navigator.pop(context);

            // Save reply only if not empty
            if (replyText.isNotEmpty) {
              await Firestore().addReply(
                postId: postId,
                commentId: commentId,
                content: replyText,
              );
            }
```

```dart
StreamBuilder<QuerySnapshot>(
  stream: Firestore().getComments(widget.postID),
  builder: (context, snapshot) {
    final commentCount = snapshot.hasData ? snapshot.data!.docs.length : 0;

    return Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Padding(
          padding: const EdgeInsets.symmetric(horizontal: 16, vertical: 4),
          child: Text("$commentCount Comments", style: TextStyle(fontSize: 12, fontWeight: FontWeight.
        ),  // Padding
        Padding(
          padding: const EdgeInsets.symmetric(horizontal: 12, vertical: 8),
          child: Row(
            children: [
              Expanded(
                child: TextField(
                  controller: _commentController,
                  decoration: InputDecoration(
                    hintText: 'Say something...',

                    suffixIcon: IconButton(
                      icon: Icon(Icons.send),
                      onPressed: () async {
                        final content = _commentController.text.trim();
                        if (content.isNotEmpty) {
                          final isAuthor = FirebaseAuth.instance.currentUser!.uid == widget.userID;
                          await Firestore().addComment(widget.postID, content, isAuthor);
                          _commentController.clear();
                        }
                      },
                    ),  // IconButton
```

Ui:

```dart
StreamBuilder<QuerySnapshot>(
  stream: Firestore().getComments(widget.postID),
  builder: (context, snapshot) {
    if (!snapshot.hasData) return CircularProgressIndicator();

    final comments = snapshot.data!.docs;
    return ListView.builder(
      shrinkWrap: true,
      physics: NeverScrollableScrollPhysics(),
      itemCount: comments.length,
      itemBuilder: (context, index) {
        final comment = comments[index];
        fixOldReplies(widget.postID, comment.id);
        return Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Padding(
              padding: const EdgeInsets.symmetric(horizontal: 12, vertical: 6),
              child: Row(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  // Avatar
                  CircleAvatar(
                    backgroundImage: comment['user_photo'] != null
                        ? NetworkImage(comment['user_photo'])
                        : AssetImage('assets/images/profile.jpg') as ImageProvider,
                  ),  // CircleAvatar

                  SizedBox(width: 12),

                  // Comment content
                  Expanded(
                    child: Column(
                      crossAxisAlignment: CrossAxisAlignment.start,
                      children: [
                        // Name and Author badge
```

```dart
Expanded(
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      // Name and Author badge
      Row(
        children: [
          Text(comment['user_name'], style: TextStyle(fontSize: 14, color: Colors.grey[800])),
          if (comment['is_author'] == true)
            Padding(
              padding: const EdgeInsets.only(left: 6.0),
              child: Container(
                padding: EdgeInsets.symmetric(horizontal: 6, vertical: 2),
                decoration: BoxDecoration(
                  color: Colors.red[100],
                  borderRadius: BorderRadius.circular(6),
                ),  // BoxDecoration
                child: Text("Author", style: TextStyle(fontSize: 10)),
              ),  // Container
            ),  // Padding
        ],
      ),  // Row
      SizedBox(height: 2),
      Text(comment['content'], style: TextStyle(fontSize: 14)),
      SizedBox(height: 2),
      Row(
        children: [
          Text(
            comment['timestamp'] != null
                ? DateFormat('yyyy-MM-dd HH:mm').format((comment['timestamp'] as Timestamp).to
                : 'Sending...',
            style: TextStyle(fontSize: 11, color: Colors.grey),
```

```dart
Column(
  mainAxisSize: MainAxisSize.min,
  children: [
    IconButton(
      icon: Icon(
        (comment['likes'] as List).contains(FirebaseAuth.instance.currentUser!.uid)
            ? Icons.favorite
            : Icons.favorite_border,
        color: Colors.red,
        size: 20,
      ),  // Icon
      padding: EdgeInsets.zero,
      constraints: BoxConstraints(), // remove default height spacing
      onPressed: () => Firestore().toggleCommentLike(widget.postID, comment.id),
    ),  // IconButton
    Text(
      '${(comment['likes'] as List).length}',
      style: TextStyle(fontSize: 10, color: Colors.grey),
    ),  // Text
  ],
),  // Column
```

```dart
child: StreamBuilder<QuerySnapshot>(
  stream: Firestore().getReplies(widget.postID, comment.id), // 🔁 getReplies method in Firestor
  builder: (context, replySnapshot) {
    if (!replySnapshot.hasData) return SizedBox();

    return Column(
      children: replySnapshot.data!.docs.map((replyDoc) {
        final reply = replyDoc.data() as Map<String, dynamic>;
        return ListTile(
          dense: true,
          contentPadding: EdgeInsets.symmetric(horizontal: 8),
          leading: CircleAvatar(
            radius: 14,
            backgroundImage: reply['user_photo'] != null
                ? NetworkImage(reply['user_photo'])
                : AssetImage('assets/images/profile.jpg') as ImageProvider,
          ),  // CircleAvatar
          title: Text(reply['user_name'], style: TextStyle(fontSize: 14,color: Colors.grey)),
          subtitle: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            mainAxisSize: MainAxisSize.min,
            children: [
              Text(reply['content'], style: TextStyle(fontSize: 13, height: 1.2)),
              SizedBox(height: 2),
              Text(
                reply['timestamp'] != null
                    ? DateFormat('yyyy-MM-dd HH:mm').format((reply['timestamp'] as Timestamp).to
                    : 'Sending...',
                style: TextStyle(fontSize: 10, color: Colors.grey),
              ),  // Text
            ],
          ),  // Column
```

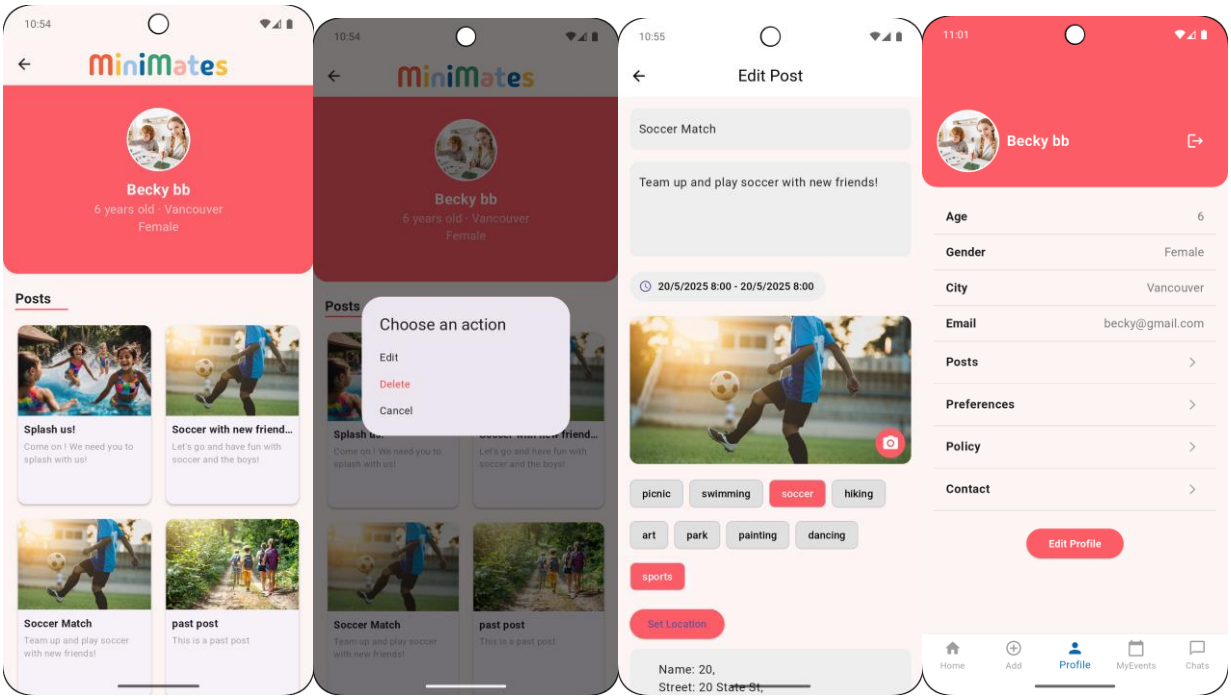
5.5 Profile Page -> Posts management

- Navigates from the Profile page to the My Posts screen when the 'Posts' list tile is tapped.
- _myPosts = _firestore.myPostList() -- Fetches the current user's posts from Firestore using their uid.
- _firestore.getMySelf() -- Fetches the current user's info.
- onLongPress: () -- Triggers a dialog on long press offering 'Edit' or 'Delete' options for a post.
- _handlePostAction – call the showdialogue to display the edit, delete and cancel options.

Post Edit:

- Users can edit text fields (title, description, address).
- They can change event time using a date-time picker (showOmniDateTimeRangePicker).
- They can select tags via toggling styled Chip widgets.
- They can update the location using Google Maps (GoogleMapFlutter), which returns coordinates and address.
- A new image can be selected using ImagePicker (camera or gallery).

Screenshots:


MiniMates
Becky bb
6 years old · Vancouver
Female
Posts
Splash us!
Come on ! We need you to splash with us!
Soccer with new friend...
Let's go and have fun with soccer and the boys!
Soccer Match
Team up and play soccer with new friends!
past post
This is a past post
Choose an action
Edit
Delete
Cancel
Edit Post
Soccer Match
Team up and play soccer with new friends!
20/5/2025 8:00 - 20/5/2025 8:00
picnic
swimming
soccer
hiking
art
park
painting
dancing
sports
Set Location
Name: 20,
Street: 20 State St,
Becky bb
Age 6
Gender Female
City Vancouver
Email becky@gmail.com
Posts
Preferences
Policy
Contact
Edit Profile
Home
Add
Profile
MyEvents
Chats

Code:

```dart
Future<void> _updatePost() async {
  try {
    String imageUrl = widget.post.image;

    // ✅ Upload new image if selected
    if (_selectedImage != null) {
      final fileName = 'images/${widget.post.title}_${DateTime.now().millisecondsSinceEpoch}.jpg';
      final ref = FirebaseStorage.instance.ref().child(fileName);
      final uploadTask = await ref.putFile(_selectedImage!);
      imageUrl = await uploadTask.ref.getDownloadURL();
    }

    // ✅ Update Firestore with new data (including image URL)
    await FirebaseFirestore.instance.collection('posts').doc(widget.post.id).update({
      'title': title.text,
      'description': desc.text,
      'tags': selectedFilters.toList(),
      'longitude': longitude,
      'latitude': latitude,
      'address': address.text,
      'start_time': _startTime,
      'end_time': _endTime,
      'image': imageUrl, // ✅ Store the updated image
    });

    Navigator.of(context).pushReplacement(MaterialPageRoute(builder: (_) => MyPostsPage()));
  } catch (e) {
    print('❌ Error updating post: $e');
  }
}
```

```dart
Future<void> _pickDateRange() async {
  final dates = await showOmniDateTimeRangePicker(
    context: context,
    startInitialDate: _startTime ?? DateTime.now(),
    endInitialDate: _endTime ?? DateTime.now().add(const Duration(hours: 1)),
  );
  if (dates != null && dates.length == 2) {
    setState(() {
      _startTime = dates[0];
      _endTime = dates[1];
    });
  }
}

String formatDate(DateTime? dt) {
  if (dt == null) return "";
  return '${dt.day}/${dt.month}/${dt.year} ${dt.hour}:${dt.minute.toString().padLeft(2, '0')}';
}
```

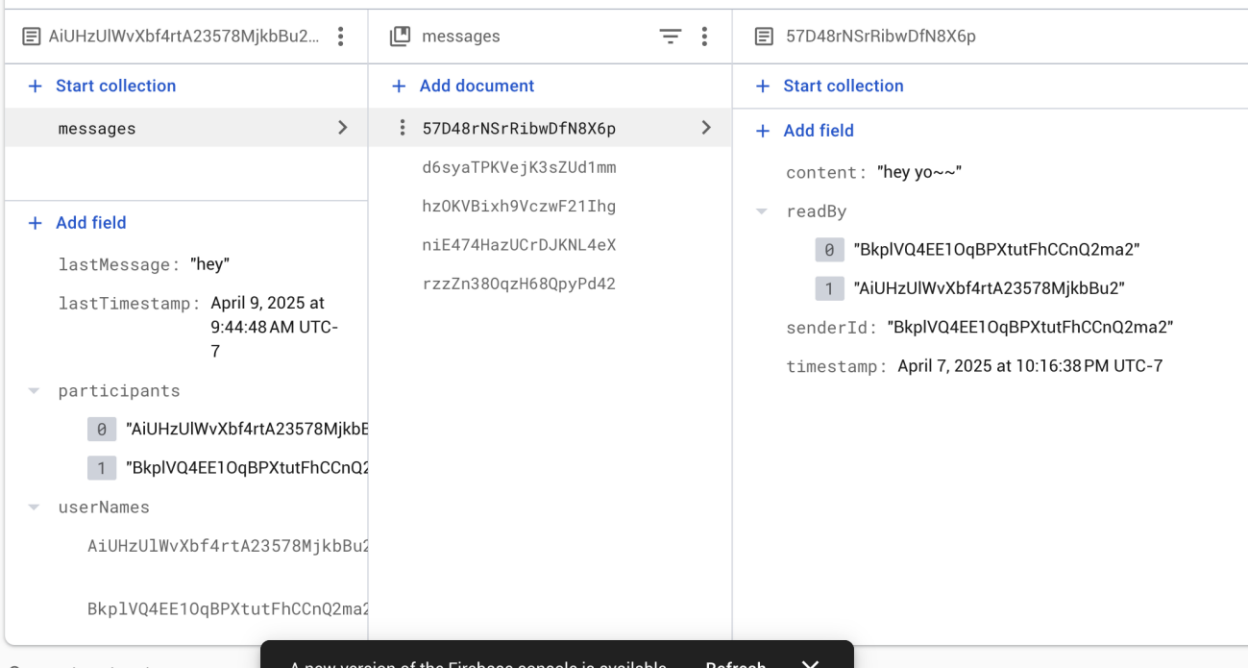
```dart
Future<void> _pickImage(ImageSource source) async {
  final picked = await picker.pickImage(source: source);
  if (picked != null) {
    setState(() {
      _selectedImage = File(picked.path);
    });
  }
}

void _showImagePickerDialog() {
  showDialog(
    context: context,
    builder: (_) => AlertDialog(
      title: const Text('Choose Image Source'),
      content: Column(
        mainAxisSize: MainAxisSize.min,
        children: [
          TextButton.icon(
            icon: const Icon(Icons.camera_alt),
            label: const Text("Camera"),
            onPressed: () async {
              Navigator.pop(context); // ✅ Close the dialog FIRST
              await _pickImage(ImageSource.camera);
            },
          ), // TextButton.icon
          TextButton.icon(
            icon: const Icon(Icons.image),
            label: const Text("Gallery"),
            onPressed: () async {
              Navigator.pop(context); // ✅ Close the dialog FIRST
              await _pickImage(ImageSource.gallery);
            },
          ), // TextButton.icon
        ],
      ), // Column
    ), // AlertDialog
  );
```

```dart
void _refreshData() {
  _myPosts = _firestore.myPostList();
  _userFuture = _firestore.getMySelf();
}

Future<void> _handlePostAction(BuildContext context, Post post) async {
  final result = await showDialog<String>(
    context: context,
    builder: (context) => SimpleDialog(
      title: const Text("Choose an action"),
      children: [
        SimpleDialogOption(
          onPressed: () => Navigator.pop(context, 'edit'),
          child: const Text("Edit"),
        ), // SimpleDialogOption
        SimpleDialogOption(
          onPressed: () => Navigator.pop(context, 'delete'),
          child: const Text("Delete", style: TextStyle(color: Colors.red)),
        ), // SimpleDialogOption
        SimpleDialogOption(
          onPressed: () => Navigator.pop(context, 'cancel'),
          child: const Text("Cancel"),
        ), // SimpleDialogOption
      ],
    ), // SimpleDialog
  );

  if (result == 'edit') {
    await Navigator.push(
      context,
      MaterialPageRoute(builder: (_) => EditPostPage(post: post)),
    );
    _refreshData();
  } else if (result == 'delete') {
    await FirebaseFirestore.instance.collection('posts').doc(post.id).delete();
    _refreshData();
  }
}
```

5.6 Chatting page implementations

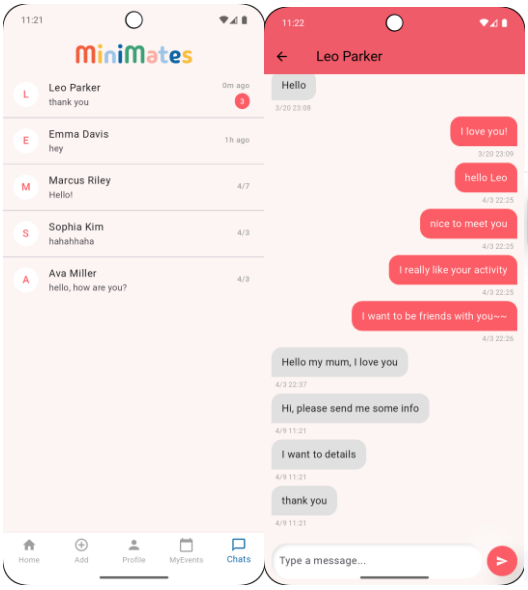
Implementation includes: 1-on-1 messaging, Real-time updates, Unread message counters, Read receipts, Chat history persistence via Firebase.

Firebase firestore structure:



- When click ‘send’ icon, call the method ‘sendMessage()’, and then implement the ‘sendMessage()’ defined in the chat_service class, then the new message is added to the messages collection.
- We use a StreamBuilder ‘getMessages()’to listen to message changes in real-time
- We count unread messages by filtering those that: Were not sent by the current user. Do not include the current user's ID in the readBy list.
- StreamBuilder<List<ChatSummary>> help display the updated chat list.
- FieldValue.arrayUnion([...]) -- A Firestore function that adds values to an array only if they’re not already in it (to prevent duplicates)

Screenshots:




```
Stream<List<ChatSummary>> getUserChats() {
                                                    User!.uid;

  return _firestore
      .collection('chats')
      .where('participants', arrayContains: currentUserId)
      .orderBy('lastTimestamp', descending: true)
      .snapshots()
      .map((snapshot) {
    return snapshot.docs.map((doc) {
      return ChatSummary.fromFirestore(doc);
    }).toList();
  });
}
```

Close. ⌥Click to Close Others (⌘W)

```
Future<void> sendMessage(String receiverId, String message) async {
  final currentUser = FirebaseAuth.instance.currentUser!;
  final currentUserId = currentUser.uid;
  final chatId = _getChatId(currentUserId, receiverId);

  final messageData = {
    'senderId': currentUserId,
    'content': message,
    'timestamp': FieldValue.serverTimestamp(),
    'readBy': [currentUserId],
  };

  // 1. Send the message
  await _firestore
      .collection('chats')
      .doc(chatId)
      .collection('messages')
      .add(messageData);

  // 2. Fetch usernames
  final currentUserSnap = await _firestore.collection('users').doc(currentUserId).get();
  final receiverSnap = await _firestore.collection('users').doc(receiverId).get();
  final currentUserName = currentUserSnap['userName'] ?? 'Unknown';
  final receiverUserName = receiverSnap['userName'] ?? 'Unknown';

  // 3. Create or update the main chat doc
  await _firestore.collection('chats').doc(chatId).set({
    'participants': [currentUserId, receiverId],
    'userNames': {
      currentUserId: currentUserName,
      receiverId: receiverUserName,
    },
    'lastMessage': message,
    'lastTimestamp': FieldValue.serverTimestamp(),
  }, SetOptions(merge: true));
}
```

```
Future<int> getUnreadCount(String chatId, String userId) async {
  final snapshot = await _firestore
      .collection('chats')
      .doc(chatId)
      .collection('messages')
      .get();

  final unreadMessages = snapshot.docs.where((doc) {
    final data = doc.data();
    final senderId = data['senderId'];
    final readBy = List<String>.from(data['readBy'] ?? []);

    // Show unread if: user hasn't read and it's not their own message
    return senderId != userId && !readBy.contains(userId);
  }).toList();

  return unreadMessages.length;
}
Future<void> markMessagesAsRead(String receiverId) async {
  final currentUserId = _auth.currentUser!.uid;
  final chatId = _getChatId(currentUserId, receiverId);

  final unreadMessages = await _firestore
      .collection('chats')
      .doc(chatId)
      .collection('messages')
      .where('senderId', isEqualTo: receiverId)
      .get();

  for (var doc in unreadMessages.docs) {
    final data = doc.data();
    // Check if 'readBy' exists
    final readBy = data.containsKey('readBy')
        ? List<String>.from(data['readBy'])
        : [];

    if (!readBy.contains(currentUserId)) {
      await doc.reference.update({
        'readBy': FieldValue.arrayUnion([currentUserId]),
        //A Firestore function that adds values to an array only if they're not
```

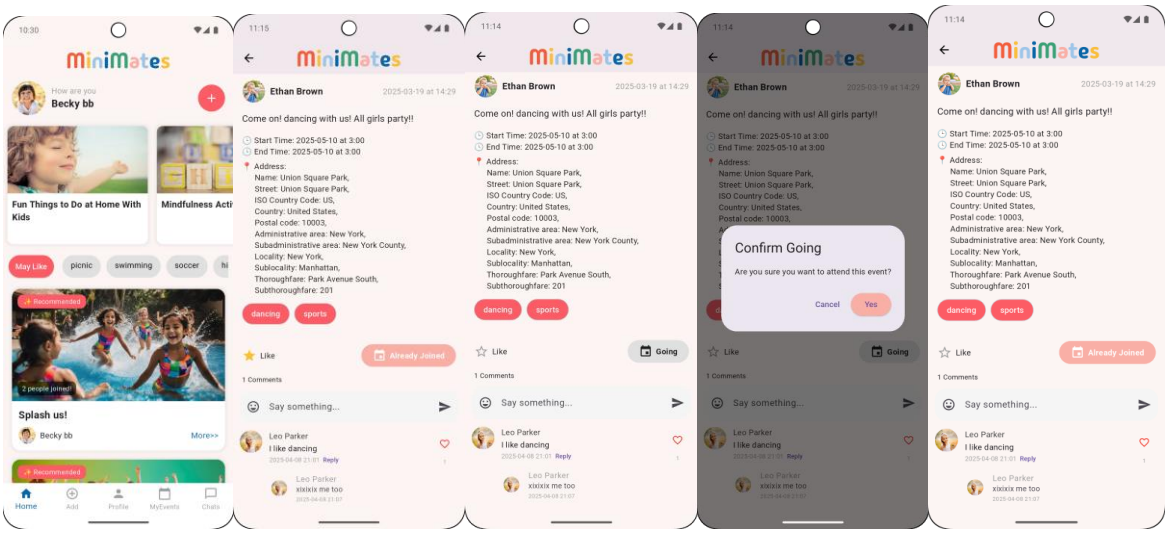
```
body: StreamBuilder<List<ChatSummary>>(
  stream: _chatService.getUserChats(),
  builder: (context, snapshot) {
    if (snapshot.connectionState == ConnectionState.waiting) {
      return const Center(child: CircularProgressIndicator());
    }

    if (!snapshot.hasData || snapshot.data!.isEmpty) {
      return const Center(child: Text("No chats yet"));
    }

    final chats = snapshot.data!;
    return ListView.separated(
      itemCount: chats.length,
      separatorBuilder: (_, __) => const Divider(height: 1),
      itemBuilder: (context, index) {
        final chat = chats[index];
        final partnerId = chat.getOtherUserId(currentUserId);
        if (partnerId == null) {
          return SizedBox(); // 👈 skip rendering this chat if no other user
        }
        final partnerName = chat.getOtherUserName(currentUserId);

        return ListTile(
          leading: CircleAvatar(
            backgroundColor: Colors.white,
            child: Text(partnerName[0],style: TextStyle(color: TColors.primary),)
          ),  // CircleAvatar
          title: Text(partnerName),
          subtitle: Text(chat.lastMessage),
          trailing: FutureBuilder<int>(
            future: _chatService.getUnreadCount(chat.chatId, currentUserId),
            builder: (context, snapshot) {

              if (snapshot.connectionState == ConnectionState.waiting) {
                print("⏳ Loading unread count...");
```

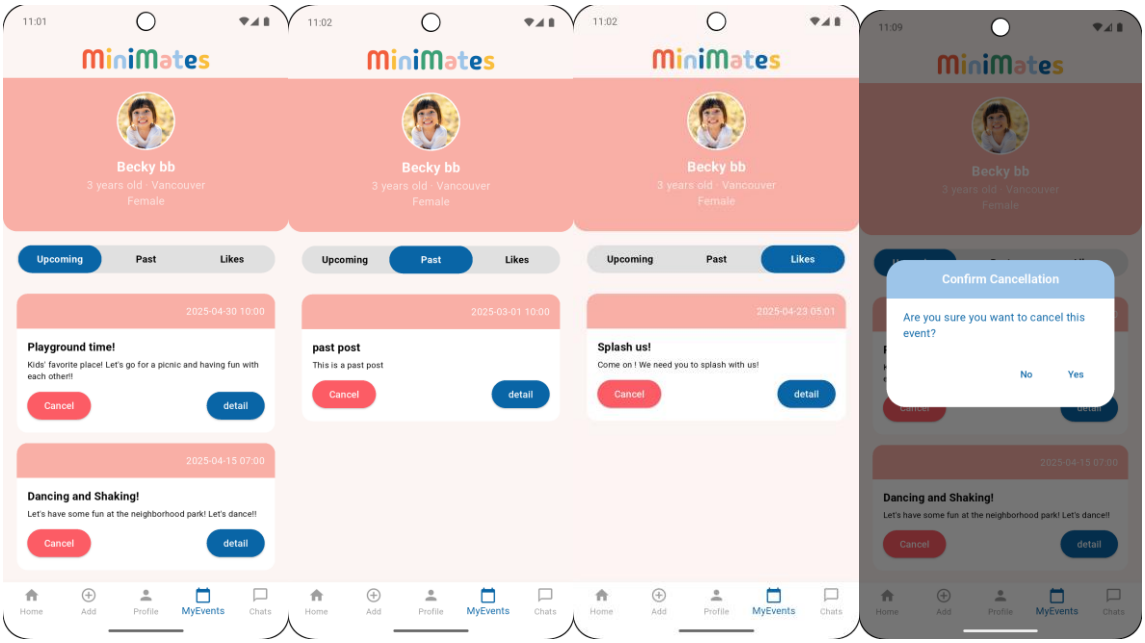
```
void sendMessage() async {
  if (_messageController.text.isNotEmpty) {
    await _chatService.sendMessage(
      widget.receiverUser.id,
      _messageController.text,
    );
    _messageController.clear();
  }
}

@override
void initState() {
  super.initState();
  markMessagesAsRead(); // mark all as read when opening
}
void markMessagesAsRead() async {
  print('Marking messages as read...');
  await _chatService.markMessagesAsRead(widget.receiverUser.id);
  print('Messages marked as read');
}
```

```
return StreamBuilder(
  stream: _chatService.getMessages(widget.receiverUser.id),
  builder: (context, snapshot) {
    //checks if the app is still waiting for data from Firebase.
    if (snapshot.connectionState == ConnectionState.waiting) {
      return Center(child: CircularProgressIndicator()); // loading
    }

    if (!snapshot.hasData || snapshot.data!.docs.isEmpty) {
      return Center(child: Text("No messages yet"));
    }

    final messages = snapshot.data!.docs;
    messages.sort((a, b) {
      final aTime = a['timestamp'] as Timestamp?;
      final bTime = b['timestamp'] as Timestamp?;
      return (aTime?.toDate() ?? DateTime.now())
          .compareTo(bTime?.toDate() ?? DateTime.now());
    });

    return ListView(
      padding: EdgeInsets.all(12),
      children: messages.map((document) => _buildMessageItem(document)).toList(),
    );  // ListView
  },
```

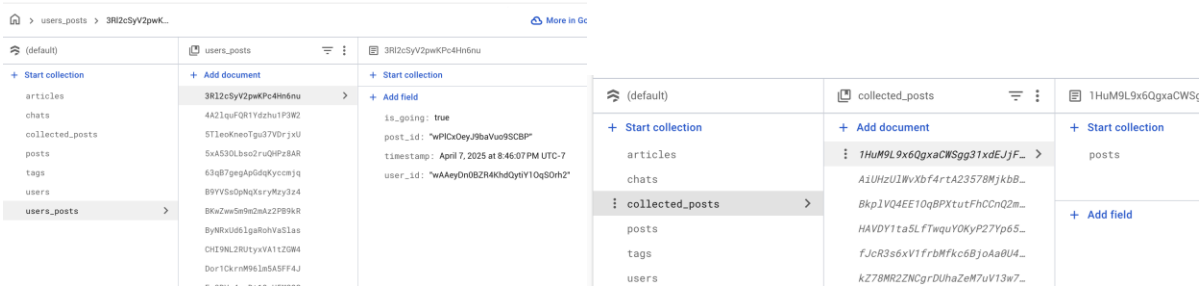
```
Stream<QuerySnapshot> getMessages(String receiverId) {
  final senderId = _auth.currentUser!.uid;
  List<String> ids = [senderId, receiverId];
  ids.sort();
  String chatId = ids.join('_');
  return _firestore.collection('chats').doc(chatId).collection('messages').orderBy('timestamp', descending: true).snapshots();
}
```

5.7 My events management





Firestore structure:



Implementation:

- AuthState + Firestore listeners
- _getCurrentList() -- This method determines what kind of post list to show based on the current tab (_selectedFilter)

```
Future<List<Post>> _getCurrentList() async {
  List<Post> posts;
  final now = DateTime.now();
  switch (_selectedFilter) {
    case 0:
      posts = await Firestore().joinedPostList();
      posts = posts.where((post) => post.startTime != null && post.startTime!.isAfter(now)).toList();
      break;
    case 1:
      posts = await Firestore().joinedPostList();
      posts = posts.where((post) => post.startTime != null && post.startTime!.isBefore(now)).toList();
      break;
    case 2:
      posts = await Firestore().getCollectedPosts();
      break;
    default:
      posts = [];
  }
  return posts;
}
```

- Firestore().joinedPostList() -- Fetches joined documents from users_posts

```
Future<int> getJoinedCount(String postId) async {
  final snapshot = await FirebaseFirestore.instance
      .collection('users_posts')
      .where('post_id', isEqualTo: postId)        // ✅ MATCHES your Firestore
      .where('is_going', isEqualTo: true)
      .get();

  return snapshot.docs.length;
}

//provides a live stream of people who joined an event
Stream<int> joinedCountStream(String postId) {
  return _firestore
      .collection('users_posts')
      .where('post_id', isEqualTo: postId)
      .where('is_going', isEqualTo: true)
      .snapshots()
      .map((snapshot) => snapshot.docs.length);
}
```

```
Future<void> unjoinEvent(String postId) async {
  String userId = _auth.currentUser!.uid;
  final snapshot = await _firestore
      .collection('users_posts')
      .where('user_id', isEqualTo: userId)
      .where('post_id', isEqualTo: postId)
      .get();

  for (var doc in snapshot.docs) {
    await doc.reference.delete();
  }


}
```

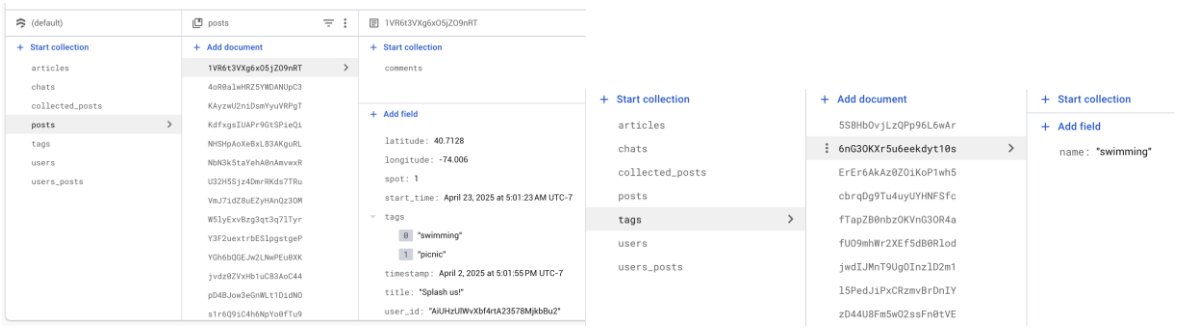
```
Future<List<Post>> joinedPostList() async {
  String user_id = _auth.currentUser!.uid;
  final snapshot = await _firestore
      .collection('users_posts')
      .where('user_id', isEqualTo: user_id)
      .where('is_going', isEqualTo: true)
      .get();

  final List<String> going_post_id_list =
  snapshot.docs.map((doc) => doc['post_id'] as String).toList();

  if (going_post_id_list.isEmpty) {
    developer.log("⚠️ No joined events found for user: $user_id");
    return [];
  }

  try {
    final postSnapshot = await _firestore
        .collection('posts')
        .where(FieldPath.documentId, whereIn: going_post_id_list)
        .get();

    return postSnapshot.docs.map((doc) => Post.fromFirestore(doc)).toList();
  } catch (e) {
    developer.log("❌ Firestore Query Error: $e");
    return [];
  }
}
```

- CollectList function and uncollect

```
Future<void> collectPost(String postID) async {
  try {
    await _firestore
        .collection('collected_posts')
        .doc(_auth.currentUser!.uid)
        .collection('posts')
        .doc(postID)
        .set({
      'post_id': postID,
      'timestamp': FieldValue.serverTimestamp(),
    });
  } catch (e) {
    print("❌ Error collecting post: $e");
  }
}

Future<void> uncollectPost(String postID) async {
  try {
    await _firestore
        .collection('collected_posts')
        .doc(_auth.currentUser!.uid)
        .collection('posts')
        .doc(postID)
        .delete();
  } catch (e) {
    print("❌ Error uncollecting post: $e");
  }
}

Future<bool> isPostCollected(String postID) async {
  final doc = await _firestore
      .collection('collected_posts')
      .doc(_auth.currentUser!.uid)
      .collection('posts')
      .doc(postID)
      .get();
  return doc.exists;
```

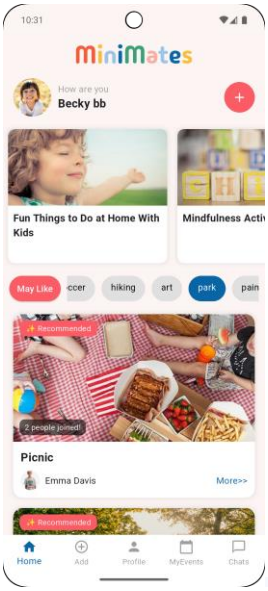
5.8 Tags filter

Firestore structure:



Implementation:

- User selects filters (tags) on the Home screen (e.g., "Sports", "Music").
- The app fetches posts from Firestore using StreamBuilder.
- Posts are filtered client-side by selected tags.
- GetPosts() returns all the filtered posts based on selected tag list.
- TagList() Fetches all tags from tags collection.




```dart
List<Post> getPosts(AsyncSnapshot snapshot, List<String> tags) {
  return snapshot.data.docs.map<Post>((doc) => Post.fromFirestore(doc)).where((post) {
    if (tags.isEmpty) return true;
    return Set.from(tags).intersection(Set.from(post.tags)).isNotEmpty;
  }).toList();
}
```

```dart
Future<List<Tag>> tagList() async {
  final snapshot = await _firestore.collection('tags').get();
  return snapshot.docs.map((doc) => Tag(doc.get('name'))).toList();
}
```

6. Evaluation Techniques

User-Centered Evaluation:

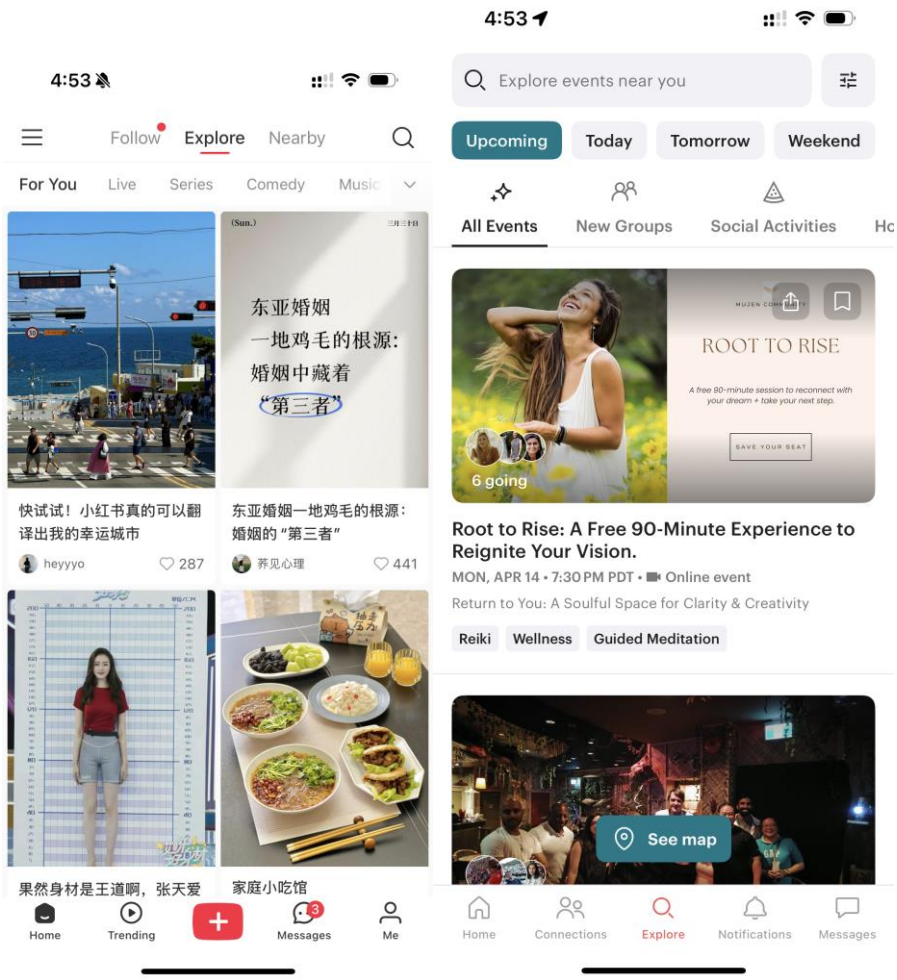
- Interviewed informal usability testing with 2 peers
- Collected quick feedback via direct interaction
- Adjusted:
    - Trigger Gemini on app start
    - Improved add post validation

Using existing benchmarks

I used two existing applications to help my app -- -- Meetup and REDnote

Compared to these apps, I added the comments feature and the joined spots counting features for my app.

Also, the chatting feature for the better user social connection.



Questionnaire using google form

I used this google form to do user research and I collected 4 user answer sheets.

https://docs.google.com/forms/d/e/1FAIpQLSeNO3wgfLEm8I2pn9vdU45CHjCDCs8U52l4yLVesqWBNqlTlA/viewform

I received responses from four people, and one of them mentioned wanting an activity map feature. Although I don't have time to implement it right now, I plan to continue working on the app after the final to include this feature.

7. Reflection and Discussion

- **Challenges**:
  - Real-time updates causing widget rebuild issues
  - Firebase rules and permission handling
  - Workflow and user flow adjustments
- **Lessons Learned**:
  - **Plan State Management Early:** Taking time to design how data flows through the app (especially with providers and real-time updates) saves a lot of debugging later.
  - **Real-Time Features Are Powerful but Tricky:** Implementing features like live "joined" counters and chat unread badges improved UX but required deep understanding of Firestore streams and performance impacts.
  - **Time Management is Key:** Balancing feature development, bug fixes, and testing under time constraints taught the importance of prioritizing and scoping features realistically.
- **Most Satisfying Part**:
  - Watching the “May Like” recommendations update in real time felt magical.
  - Chatting feature makes my app looks more realistic

8. Work Log

Week 02 (Jan 15 – Jan 21)

Date	Number of Hours	Description of work done
Jan 15, 2025	2	Do research to confirm the technology hardness.
Jan 18, 2025	2	-Finish the draft proposal. -App VI design (color system, logo and slogan.)
Jan 19, 2025	1.5	Draw the Gantt chart.

Jan 20, 2025	3	Write the introduction and the proposal modification.
Jan 21, 2025	5	-UI design draft using Figma. -Made the proposal slides.

Repo check in

This week, I completed the project proposal, which includes the app's visual identity (VI) design, the proposal report, the proposal slides, and the UI design draft in Figma. I also created a draft prototype for my app.

Week 03 (Jan 22 – Jan 29)

Date	Number of Hours	Description of work done
Jan 22, 2025	1	Discuss the app proposal.
Jan 23, 2025	2	Revise the proposal and the Gantt chart.
Jan 24, 2025	1	Learn the flutter from youtube video.
Jan 26, 2025	4	Learn the flutter from youtube video.
Jan 28, 2025	2	Learn the flutter E-commerce app.

Repo check in

This week, I revised and submitted the project proposal. And I started to setup my project structure and folders. I actually have no idea about it, so I find a YouTube video to learn about it. And I plan to start building my app from next week.

Week 04(Jan 30– Feb 04)

Date	Number of Hours	Description of work done
Jan 30, 2025	5	Got some problems with the APG version and the JAVA SDK Version

Feb 02, 2025	3	Build the project structure, and tried the flutter native splash to make the splash screen, but failed.
Feb 03, 2025	5	Done the splash screen, but cannot remove the default one…
Feb 04, 2025	4	Tried to remove the default one, but failed. I decided to continue with the signin sign up screen.

Repo check in

This week, it took me a lot of time to resolve the version problem, but the issue persists. I shouldn't waste my time on this, next time I will ask others for the help. And next week my plan is to finish the login signup page, and the main page for the app, hope I can finish on time.

Week 05(Feb 04 – Feb 08)

Date	Number of Hours	Description of work done
Feb 04, 2025	3	Doing some design of the onBoarding page.
Feb 05, 2025	4	Learn how to build the onboarding page and started to do it.
Feb 07, 2025	3.5	Finish the 3 pages onboarding page.
Feb 08, 2025	6	Tried to push the project to the github, but since I created two github accounts, I got the problem with the git push. I cannot push it to the new git account and after tried several hours, I can only push it to the main folder…

Repo check in

This week, it took me a lot of time to working with the git, and I got some problems with the different account and keys switching, and I didn't submit the work log on time, I'm so sorry that I missed it. I'll try my best to figure it out on time next week.

Week 06(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.

Repo check in

This week, I worked on designing and developing the login and signup pages, following tutorials and implementing the frontend in Flutter. Since I was new to Firebase, I spent time learning its functionalities and ultimately decided to use it instead of MongoDB and Node.js. After setting up my Firebase account and configuring the structure, I faced challenges connecting it to my project, particularly with the auto-generated files. However, after multiple attempts and researching solutions, I successfully integrated Firebase with my project.

Week 07(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.

Repo check in

This week is a busy week, I focused on implementing the post feature, ensuring the list view displayed correctly. I faced issues with image uploads, where only the camera worked, but later found a fix by saving images from Google to the gallery. I worked on the midterm report, updated my worklog, and prepared the Gantt chart, video setup, and README file. Towards the end of the week, I encountered navigation issues between the onboarding and login pages, which also affected signup. I spent time debugging these problems while recording and editing the project video before finalizing everything for submission.

Week 08(Feb 24 – Mar 2)

Date	Number of Hours	Description of work done
Feb 24, 2025	4.5	Add the edit profile page logic, add the preference page, the upload image is not included
Feb 25, 2025	4	Created the preference page .
Feb 27, 2025	2	Modify all the UI system.
Feb 28, 2025	5.5	Implemented the "Going" button feature to allow users to join events and reflect status in Fire store.
Mar 1, 2025	3.5	Created the "Collect" button feature to allow users to collect and uncollected posts.
Mar 2, 2025	2	Built the "My Events" page to display collected and upcoming events.

Repo check-in:

This week I make the going and collect working first, and create the profile page and edit profile page. I'm going to make the logic working next week and enhance my app UI again.

Week 09(Mar 3 – Mar 9)

Date	Number of Hours	Description of work done
Mar 3, 2025	4.5	Completed the "My Events" page, displaying collected and upcoming events correctly.
Mar 4, 2025	2	Added additional fields to the "Add Post" page (e.g., tags, event type, age range).
Mar 6, 2025	2	Fixed Firestore issues related to storing and retrieving post data.
Mar 7, 2025	3	Set up article content storage in Firestore and linked it to the home page.

Mar 8, 2025	2.5	Created the article detail page and connected it with the Firestore data.
Mar 9, 2025	1.5	Debugged data loading issues and improved article display formatting.

Repo check-in:

My events page is done, and the list are displaying properly, and the article list and detail page are done. I got some issues about the article data loading but It works well now.

Week 10(Mar 10 – Mar 16)

Date	Number of Hours	Description of work done
Mar 10, 2025	3	Enabled profile image selection and upload to Firebase Storage. Linked the uploaded profile images to the user profile and display page(still not finished).
Mar 11, 2025	4.5	Create the bottom nav bar for the app.
Mar 12, 2025	4	Write python code to generate the user and posts.
Mar 13, 2025	4	Enhanced the "Add Post" page with improved validation and layout updates(date picker and duration picker).
Mar 14, 2025	2.5	Modified the signup page to include the username field and store it in Firestore.
Mar 15, 2025	3	Working on the chat feature now...

Repo check-in:

The bottom nav bar took me some time because I have to handle the relationship between the four pages. And the basic functions are done, I think the next maybe integrate the chatting and the mapping.

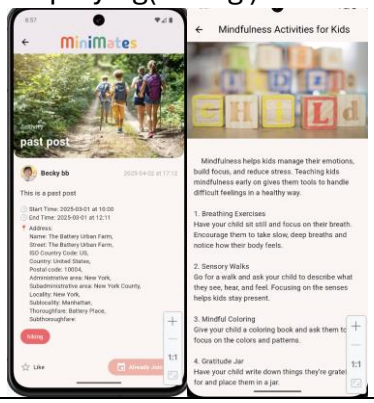
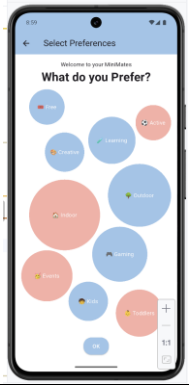
Week 11(Mar 23 – Mar 29)

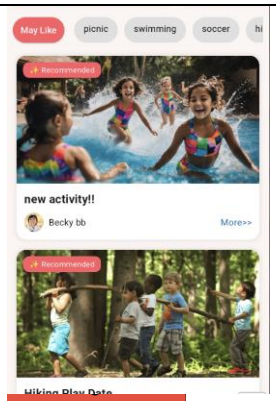
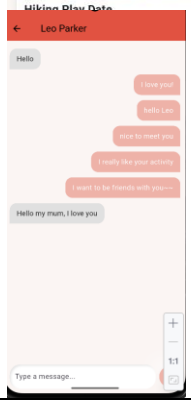
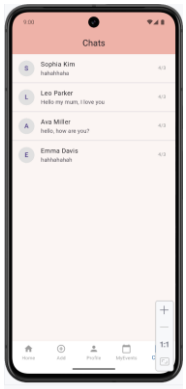
Date	Number of Hours	Description of work done
Mar 24, 2025	3.5	I learned map video and research for the location picker, and I decided to use the geocoding library.
Mar 25, 2025	3	Map started following the YouTube video.
Mar 27, 2025	4	I can go into the map screen but still cannot pick the location.
Mar 28, 2025	5.5	The location picker is working but cannot be stored in the firebase. And after picking the location, the other post content disappeared.
Mar 29, 2025	4	Map done. The address is still very long, I will still work on it next week.

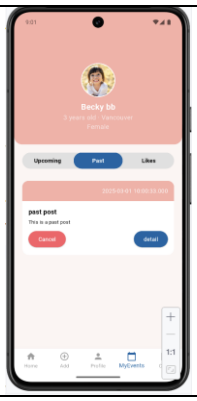
Repo check-in:

This week, I was working on the map feature, since after consulting with you, I want to try my best to extend my app, although I'm still very slow, but I can see my app is becoming more and more complex and real. The next week I will focus on the AI part and the add some details to make my app more realistic.

Week 12(Mar 30 – Apr 6)

Date	Number of Hours	Description of work done
Mar 30, 2025	4	Fix the post details, include duration and location displaying. And also the article details displaying(debug). 
Mar 31, 2025	2	Preference page Ui adjust(the bubbles). 
Apr 1, 2025	4	Integrate AI to generate the recommend activities list.
Apr 2, 2025	4.5	Display the list to the home page. Adjust chatting window Ui.

		 
Apr 3, 2025	2	Chatting history page added. 
Apr 5, 2025	3	Chatting history page implemented.
Apr 6, 2025	4	Chatting unread message marked. My events page card time display, and change posts to past posts, and implemented.

Repo check-in:

This week, I was working on enriching the app from details and finishing the chatting feature. I implemented the chat history and the new message signal showing. I also want to enable the notifications next week before Wednesday, and fix some minor things like the my-events card cancel button.

Week 13(Apr 7 – Apr 12)

Date	Number of Hours	Description of work done
Apr 7, 2025	4	Ui and color system adjustment. Comments feature designed.
Apr 8, 2025	5	Trying to add the comments feature into the app. Create the Firestore structure.
Apr 9, 2025	4	Adjust the validation in add post.
Apr 10, 2025	5.5	Drawing the app user guide.
Apr 11, 2025	4	Working on the report.
Apr 12, 2025	5	Working on the slides.

Repo check-in:

This week, I was working on the rest features and report. I made a user guide for my app and also tried the user flow based on the app.

9. Appendix

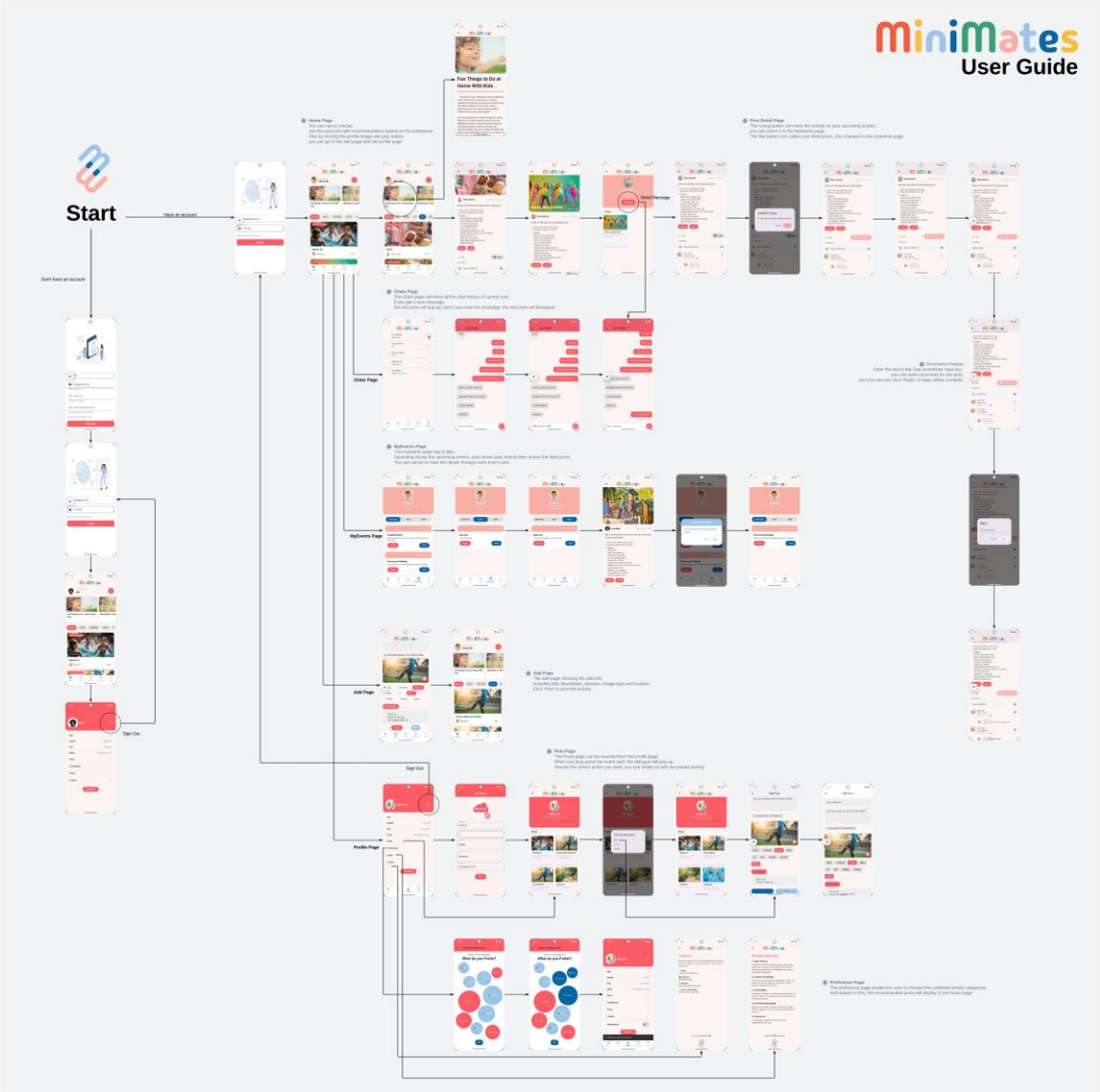
Appendix A -- -- Minimates User Instructions

Location in Repository:

/DocumentsAndReports/ Minimates_user_guide.pdf

This document provides step-by-step instructions for users on how to navigate and use the MiniMates application. It covers creating posts, joining activities, profile customization, and viewing recommendations.

Content:



Appendix B -- -- Installation Instructions

Location in Repository:

W25_4495_S1_BeckyJ/README.md

Content:

MiniMates - A Playdate Platform for Parents

MiniMates is a Flutter-based mobile application designed to help parents effortlessly plan playdates, discover activities, and connect with other parents. The app provides an intuitive event management system, AI-powered recommendations, and profile customization to enhance user experience.

Getting Started

Installation Instructions

1. Prerequisites Before running this project, make sure you have:

1. Install Flutter SDK

To run this project, you must install **Flutter SDK** on your machine.

📌 **Follow the official installation guide here:**
🔗 Install Flutter SDK

After installation, verify your setup by running: flutter doctor

2. Android Studio/Xcode

- For device/emulator testing

3. Firebase Account

- Set up Firebase for authentication and Firestore
2. Clone the Repository Use the following command in your terminal:

git clone https://github.com/BeckyJin300374335/W25_4495_S1_BeckyJ.git

cd Implementation/minimates_app

3. Set Up Firebase Go to Firebase Console and create a new project. Enable Authentication → Sign-in method → Enable Email/Password. Enable Firestore Database in test mode. Enable Storage for image uploads. Download the google-

services.json (for Android) and GoogleService-Info.plist (for iOS) and place them in:
Android: android/app/ iOS: ios/Runner/
Run Firebase initialization: **flutterfire configure**

4. Install Dependencies Run the following command to install all required Flutter packages: **flutter pub get**
5. Run the App Start the app on a connected device or emulator: **flutter run**

This project is a starting point for a Flutter application.

10. References

Ai tools

- ChatGPT for the Ui adjustment, part of the coding, debugging and reports.

Websites

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

https://www.xiaohongshu.com/explore

https://www.meetup.com/