App Dev 2

Project Final Report - Team Coffee

Mahsa, Ali and Guillaume

FriendMap™

Final Report

Due date: Friday January 16 2026

Project Aim and Description

FriendMap is a Flutter mobile application that helps users discover and share places with friends, plan small events called Moments, and coordinate socially through lightweight, location first messaging. The app combines a map-based home experience (implemented via an embedded WebView map UI) with Firebase-backed social features: authentication, friend discovery, friend requests, chat, event planning, and notifications.

FriendMap supports these core flows:

Authentication & onboarding: Users can sign in with email/password, Google, or phone/SMS, then access the app's main navigation (Home, Friends, Moments, Chat).

Location discovery & saving: The Home map displays locations loaded from Firestore and allows users to save/unsave locations to their profile.

Sharing Scenes (locations): Users can share a specific location to one or more friends via chat. Chat is intentionally not made for sending text messages. Conversations consist of shared Scenes and Moments.

Friends system: Users can search for other users, send friend requests, accept/decline requests, view a friend's profile, and unfriend.

Moments (events): Users can create events called Moments tied to a location (title, optional description, date/time), invite friends, share Moments through chat, and collect RSVP responses.

Notifications: When users receive friend requests, shared scenes, or moment invites, the app stores notifications in Firestore (and can also show local

notifications), giving users a notification inbox with read/unread status.

Technology stack / architecture overview

Frontend: Flutter/Dart UI.

Backend: Firebase (Auth, Firestore, and image Storage).

Map experience: WebView custom map UI (loading bundled assets and injecting Firestore-provided location + friend marker data).

Data model (high-level):

users/{userId} (profile + friends list + optional location)

users/{userId}/notifications/{notificationId}

friendRequests/{requestId}

conversations/{conversationId} + messages/{messageId}

locations/{locationId} (publicly readable places dataset)

moments/{momentId} (events + invites + responses + optional shareCode + guest responses)

Functional Requirements

Authentication

Users can sign up / sign in using email & password.

Users can sign in with Google.

Users can sign in with phone number + SMS code.

Users can reset password via email reset flow.

Home / Map

The app displays a map UI on the Home screen.

The map loads a list of locations from Firestore and renders them in the map UI.

Users can open a location and view details on the location modal pop up.

Users can save and unsave locations to their account.

The map can display friend markers/pins (friend locations) where available.

Friends

Users can search users by email.

Users can send a friend request to another user.

Users can view incoming friend requests.

Users can accept or decline friend requests.

Users can view a friend profile including that friend’s saved locations.

Users can pin/unpin friends to highlight them on the home map overlay (between 0 and 3).

Users can unfriend an existing friend.

Chat / Sharing

Users can view a list of conversations they participate in.

Users can open a conversation detail view and see shared items.

Users can share a location/Scene to one or more friends, creating messages with a locationId.

Users can share a Moment to friends, creating messages with a momentId.

Messages received in a conversation can be marked as read inside Firestore document when the conversation is opened.

Moments (Event)

Users can create a moment with title, optional description, date/time, and selected location.

Users can view My Moments and Invited moments lists.

Users can invite friends to a moment (adds to invited list and sends notifications).

Users can RSVP to a moment with responses like going/maybe/can’t go, they may withdraw a vote as well.

Moment creators can edit or delete their moments.

Notifications

The app stores notifications for events such as friend requests, new shared scenes/messages, and moment invites.

Users can view notifications, tap to navigate (to Friends or a specific conversation), mark read, mark all as read, and delete notifications.

Non-Functional Requirements

Security & privacy

Only authenticated users can perform protected operations (create friend requests, create conversations, update their profile).

Users can update only their own user profile.

Chat data is only accessible to conversation participants.

Personal data exposure is minimized; location sharing happens explicitly through chat and/or profile settings.

Reliability

The app should handle intermittent connectivity gracefully (Firestore offline persistence is enabled).

Network failures should not crash the app; UI should show appropriate error states/snackbars.

Performance

Lists should load efficiently via streams and pagination/limits where appropriate.

Firestore query constraints (like batching for whereIn limits) must be handled.

Usability

Navigation is consistent through a bottom navigation bar (Home/Friends/Moments/Chat).

Key actions are discoverable: friend search, create moment, share to friends, open notifications.

Maintainability

Clear separation of concerns: pages/widgets for UI, services for Firestore/Auth logic, models for serialization.

Compatibility

Target modern Flutter runtime, should run on Android with Firebase configuration.

User Stories

User Story 1 - Discover and save places I want to try

I use FriendMap to browse a map-based list of interesting locations and quickly save the ones I want to visit later. Instead of taking screenshots or writing notes, I can build a personal ‘saved places’ list inside the app so I always have ideas ready when I’m planning a hangout or looking for somewhere new.

User Story 2 - Share a specific place with friends without back-and-forth texting

I use FriendMap to send a Scene/location directly to one or more friends so we can coordinate on where to go. It saves time because I’m not typing long explanations, copying addresses, or sending multiple links, my friend receives the location card in chat and can open the details immediately.

User Story 3 - Add friends safely before sharing anything

I use FriendMap because it has a friend request system that lets me control who I connect with. I can search for someone by email, send a request, and

only start sharing locations and events after they accept, so the app feels more private and intentional than posting publicly.

User Story 4 - Plan an event/Moment tied to a real location

I use FriendMap to create a Moment for something like coffee tomorrow or game night, attach it to a chosen location, and include a date/time. This helps me turn ‘we should hang out sometime’ into a concrete plan that my friends can view, respond to, and keep track of in one place.

User Story 5 - Keep up with requests, invites, and shares through notifications

I use FriendMap because notifications keep me updated when someone sends a friend request, shares a scene, or invites me to a moment. I don’t have to constantly check every tab, when I tap a notification it takes me to the right place (friends, chat, or moments).

Test Cases

Authentication

Test Case 1: Email sign-up

Steps: Open app > Sign Up > enter valid email/password > submit.

Expected: Account created; user proceeds into app (Home); user profile created/available in Firestore.

Test Case 2: Email sign-in

Steps: Open app > Sign In > enter valid credentials > submit.

Expected: User signed in; navigates to Home.

Test Case 3: Password reset

Steps: Sign In screen > “Forgot Password?” > enter email > send reset.

Expected: Confirmation message shown; reset email sent (Firebase).

Test Case 4: Google sign-in

Steps: Sign In screen > “Sign in with Google” > complete OAuth.

Expected: User signed in; navigates to Home; profile usable.

Test Case 5: Phone sign-in

Steps: Sign In screen > “Sign in with Phone” > enter phone > enter SMS code.

Expected: User signed in successfully.

Friends

Test Case 1: Search users by email

Steps: Friends tab > enter partial/full email > Search.

Expected: Matching users displayed; no crash on empty/no results.

Test Case 2: Send friend request

Steps: Search results > tap add friend icon on a user.

Expected: Friend request created in friendRequests; recipient gets a notification; sender sees “Friend request sent”.

Test Case 3: Prevent duplicate friend request

Steps: Send friend request to same user again.

Expected: Error shown like “Friend request already sent” (no duplicate request stored).

Test Case 4: Accept friend request

Steps: Receiver opens Friends > Friend Requests tab > Accept.

Expected: Request status becomes accepted; both users’ friends arrays updated; request no longer appears as pending.

Test Case 5: Decline friend request

Steps: Receiver declines a pending request.

Expected: Request status becomes declined; users are not added as friends.

Test Case 6: Pin/unpin friend

Steps: Friends list > open menu > pin; repeat to unpin.

Expected: Pin state persists; pinned list updates; pinned friends appear on home overlay where applicable.

Test Case 7: Unfriend

Steps: Friends list > unfriend > confirm.

Expected: Both users removed from each other’s friends arrays; friend disappears from list.

Locations / Map / Saved Locations

Test Case 1: Load locations on Home

Steps: Sign in > open Home.

Expected: Map UI loads; locations fetched from Firestore; markers/entries visible.

Test Case 2: Save location

Steps: Open a location > choose save (via map UI action).

Expected: Location ID added to user’s saved list in Firestore; saved state reflects in UI.

Test Case 3: Unsave location

Steps: For a saved location > unsave.

Expected: Location removed from saved list; UI updates accordingly.

Test Case 4: Friend profile saved locations

Steps: Friends > open friend profile > view saved locations list > tap a location.

Expected: Saved locations load; tapping opens the location detail sheet.

Chat / Sharing

Test Case 1: Create/open conversation

Steps: Friends > Start chat OR open friend profile > Open chat.

Expected: Conversation created if needed; conversation opens and displays message history.

Test Case 2: Share a scene (location) to a friend

Steps: Home map > share/send scene > select friend(s).

Expected: Message created with locationId; recipient sees it in conversation; notification stored for recipient.

Test Case 3: Open shared location in chat

Steps: Conversation > tap location card.

Expected: Location detail sheet opens with correct details.

Test Case 4: Mark messages as read

Steps: Receive new shared content > open conversation.

Expected: Unread messages become read; read status stored (where supported).

Moments

Test Case 1: Create moment

Steps: Moments tab > New Moment > enter title/optional description > pick location > pick date/time > create.

Expected: Moment stored in Firestore with creator ID and (optionally) shareCode; appears in “My Moments”.

Test Case 2: Invite friends to moment

Steps: Open moment details > Send Moment to Friend > select friends > Send.

Expected: Friends added to invited list; moment shared in chat; invite notifications created.

Test Case 3: RSVP to moment

Steps: Open invited moment > choose Going/Maybe/Can’t go; tap same option again to clear.

Expected: Response stored/updated/cleared; counts update in UI.

Test Case 4: Edit moment (creator only)

Steps: Creator opens moment > edit > change title/description/date/time > save.

Expected: Changes persist; non-creator cannot edit.

Test Case 5: Delete moment (creator only)

Steps: Creator opens moment > delete > confirm.

Expected: Moment removed from Firestore and disappears from lists.

Notifications

Test Case 1: Notification created on friend request

Steps: User A sends friend request to User B.

Expected: User B has new notification stored and visible in Notifications page.

Test Case 2: Tap notification navigation

Steps: Tap friend_request notification; tap message notification with conversationId.

Expected: Navigates to Friends page or directly into the relevant conversation when possible.

Test Case 3: Mark all read + clear all

Steps: Notifications page menu > mark all read; then clear all.

Expected: Unread markers removed; notifications deleted after confirmation.

Individual’s role and responsibilities

Mahsa:

Integrating Firebase logic for a sizeable part of the app.

Nav Bar feature.

Added the 35 locations for the map into Firebase and set up the logic to connect them.

Local notification for friend requests.

Geo-location on settings page and profile.

Created settings page.

Added the logic for chats in app.

Various debugging.

Overall: Mahsa has been able to demonstrate strong learning capabilities. She took charge of implementing valuable components into the project.

Ali:

Notif Center page.

Moments/Event page and its app + Firebase code and logic.

Developing a separate app to quickly test ideas before spending time putting it inside the official codebase. This part is extremely helpful for us since we did not use branches on Github.

Bringing high-level guidance due to more industry experience in mobile app dev.

Login page with all of its logic, and the core UI for it.

Friend profile.

Fixing troublesome bugs across the whole project.

Overall: Ali has been a strong support role for the team. He is the only one in the group who had existing experience which helped us move forward when things got stressful.

Guillaume:

Project manager role.

Taking the project ideas we had and formulating an actionable plan for us to follow reliably with tiers and micro-steps.

Coordinating the team productivity sessions and task assignment to each member.

Documentation and presentation material.

Logo’s and branding.

Coordinating the overall guidance from Ali into the project to make adjustments where needed and plan ahead.

Managing the Firebase account, connecting app logic set up by team to the firestore rules/index/database.

Setup email, phone and google login/signup logic with Firebase. As well as wiring the email templates for user registration and forgot email.

Tightening security level .gitignore stuff.

Connecting the notification center to local notifications.

UI design for nav bar, user icons, pinned friends, splash screen, app icon, notif center button, search bar, and various other visual elements accross the app. (Design and debugging).

Managed all the top-level firestore rules for each part of the app.

Creating a separate app for our google maps service used in homepage. Then merging it into our app and debugging it until it worked. (Reason why Guillaume has 250k+ lines of code pushed to the Github. The map files metadata was very large for the UI and actual data like hours, coords, names, descriptions, etc..)

Original map placeholder (swapped out for new map).

Modals for seeing locations when tapping on them.

Edit name for user feature (error handling, char limits, logic).

Profile design.

Location cards for profile and chats.

Pinned friends feature.

Profile picture feature (add, delete, and upload). Connected with Firebase storage. Also making the user icons match their profile pictures across the app each time theres an icon.

Connected the Firebase logic needed for the teams implementation of chats feature.

Sending Scenes and Moments logic in chat.

Connecting the Moments page ‘invited’ tab to the chats sent to user. Displays all the events that have been sent to user directly in the list view for that tab.

Unfriending a friend.

Finalizing the attendance logic for the Moments/events (Going, Not going, Maybe, withdraw vote).

Overall: Guillaume handled his own deliverables while also helping pull everything together across the team. He took what others built and helped refine it. Fixing bugs, connecting Firebase, and adding detail where things needed to be clearer or more complete. He kept communication steady so tasks got started quickly and workloads stayed balanced. He also helped break ideas into clear steps so execution stayed organized and trackable. He spent considerable hands-on time coding with Mahsa to keep everyone aligned and moving at a consistent pace. He supported the branding, presentation structure, and productivity materials to keep the project polished and running smoothly. The role of project management was handled across the board.

CRUD Operations (Firestore + Storage)

Overview

This app’s core data is stored in Cloud Firestore (documents/collections) with user images stored in Firebase Storage. Most CRUD is implemented in lib/services/*_service.dart.

Data Model (Collections)

users/{uid}

Typical fields: uid, email (lowercased), name, photoURL, createdAt, friends[], savedLocations[], pinnedFriends[], location{ … }, lastLocationUpdate

friendRequests/{requestId}

Fields: fromUid, toUid, status (pending|accepted|declined), createdAt

locations/{locationId}

Document ID is the location id; the document stores locationData (shape depends on what the app uploads)

moments/{momentId}

Fields: title, description, locationId, locationName, locationAddress, dateTime, createdBy, createdAt, shareCode, invitedFriends[], responses{ uid: "going|maybe|not_going" }, guestResponses[]

conversations/{conversationId}

Fields: participants[], createdAt, updatedAt, lastMessage, lastMessageTime, lastMessageSenderId

Subcollection: conversations/{conversationId}/messages/{messageId}

Fields: senderId, text, timestamp, read, readAt, optional locationId, optional momentId

users/{uid}/notifications/{notificationId}

Fields: title, body, type (ex: friend_request|message|moment_invite), createdAt, isRead, optional data{ ... }

Users

Implemented in lib/services/user_profile_service.dart.

Create

ensureUserDocumentExists(): creates users/{uid} if missing with defaults (friends/savedLocations/pinnedFriends empty arrays).

Read

getUserByUid(uid), getCurrentUserProfile()

getCurrentUserProfileStream() (real-time updates)

searchUsersByEmail(emailQuery) (prefix query; fallback in-memory filtering)

Update

updateUserName(newName)

updateProfilePicture(photoURL) (writes URL after upload)

updateUserLocation(locationMap) sets location + lastLocationUpdate

Pinning:

pinFriend(friendUid) (enforces max 3)

unpinFriend(friendUid)

getPinnedFriends(), getPinnedFriendsStream()

Delete

No “delete user profile” method is included in the services.

Friend Requests + Friends (friendRequests, users.friends)

Implemented in lib/services/friends_service.dart.

Create (Friend Request)

sendFriendRequest(toUid) creates a friendRequests doc with status: pending.

Also attempts to create a recipient notification via NotificationService.storeNotificationForUser(…).

Read

getPendingFriendRequests() streams incoming pending requests for the current user.

Friends list:

getFriendsListOnce() (one-time)

getFriendsList() (stream)

getFriendProfiles(friendUids) / getFriendProfilesStream() (batched due to Firestore whereIn limit of 10)

Update

acceptFriendRequest(requestId) (batch): sets request status=accepted and adds each user to the other’s users.friends[].

declineFriendRequest(requestId) sets request status=declined.

Delete (Relationship)

unfriend(friendUid) (batch): removes each user from the other’s users.friends[].

Note: requests are not physically deleted here; they’re status-updated.

Locations (locations) + Saved Locations (users.savedLocations)

Implemented in lib/services/locations_service.dart and lib/services/saved_locations_service.dart.

Create / Update (Locations)

uploadLocation(locationData): set(… , merge: true) into locations/{id}.

uploadLocations(list): batch set(… , merge: true).

Read (Locations)

getAllLocations(), getLocationsStream(), getLocationById(locationId)

Delete (Locations)

No delete method is provided.

Saved locations are “CRUD on an array field” in users/{uid}:

Create

saveLocation(locationId) → FieldValue.arrayUnion([locationId])

Read

getSavedLocations(), getSavedLocationsStream(), getSavedLocationsStreamForUser(uid)

Delete

unsaveLocation(locationId) → FieldValue.arrayRemove([locationId])

Moments / Events (moments)

Implemented in lib/services/moments_service.dart.

Create

createMoment(…) creates a doc in moments via .add(…) using MomentModel.toFirestore().

Optionally generates shareCode.

Read

getMyMomentsStream() (where createdBy == currentUser.uid)

getInvitedMomentsStream() combines:

moments where invitedFriends contains user

moments referenced in chat messages (moment IDs shared to the user)

getMomentById(momentId)

getMomentByShareCode(shareCode) (for web access)

Update

RSVP:

updateResponse(momentId, response) sets responses.{uid} = response

clearResponse(momentId) deletes responses.{uid}

Guests (web form):

addGuestResponse(...) appends to guestResponses[]

Invites:

inviteFriends(momentId, friendIds) array-union into invitedFriends[] + creates notifications for invitees

Share code:

regenerateShareCode(momentId) updates shareCode

Edit details:

updateMoment(momentId, {title, description, dateTime})

Delete

deleteMoment(momentId) deletes the document (only if createdBy == currentUser.uid).

Chat (conversations, messages)

Implemented in lib/services/chat_service.dart.

Create

Conversation: getOrCreateConversation(otherUserId) creates a conversations doc if none exists between the two participants.

Message: sendMessage(conversationId, {text, locationId, momentId}) batch-writes:

new doc in conversations/{id}/messages

updates conversation metadata (lastMessage*, updatedAt)

Read

getConversations() streams conversations for current user.

getMessages(conversationId) streams messages; sorts client-side.

Update

markMessagesAsRead(conversationId) batch-updates messages not sent by current user: sets read=true, readAt=now.

Delete

No delete-message or delete-conversation method is included.

Notifications (users/{uid}/notifications)

Implemented in lib/services/notification_service.dart (plus listeners like FriendRequestManager).

Create

storeNotification(title, body, ...) writes to current user's notifications.

storeNotificationForUser(recipientUserId, ...) writes to another user (used for friend requests, messages, moment invites).

showNotification(...) displays local notification + stores in Firestore.

Read

getNotificationsStream() (latest 50, ordered by createdAt)

getUnreadCountStream()

Update

markAsRead(notificationId)

markAllAsRead() (batch)

Delete

deleteNotification(notificationId)

deleteAllNotifications() (batch)

Profile Pictures (Firebase Storage)

Implemented in lib/services/storage_service.dart.

Create/Update

uploadProfilePicture(imageFile) uploads to profile_pictures/{uid}.jpg and returns a download URL (typically then saved into users/{uid}.photoURL via UserProfileService.updateProfilePicture(…)).

Read

App reads via the stored photoURL (HTTP download URL).

Delete

deleteProfilePicture() deletes profile_pictures/{uid}.jpg (best-effort; ignores missing object).

FriendMap™