



# ActivMatch



Mikael Morales  
Charles Parzy  
Yassin Kammoun

# Code organization - Packages

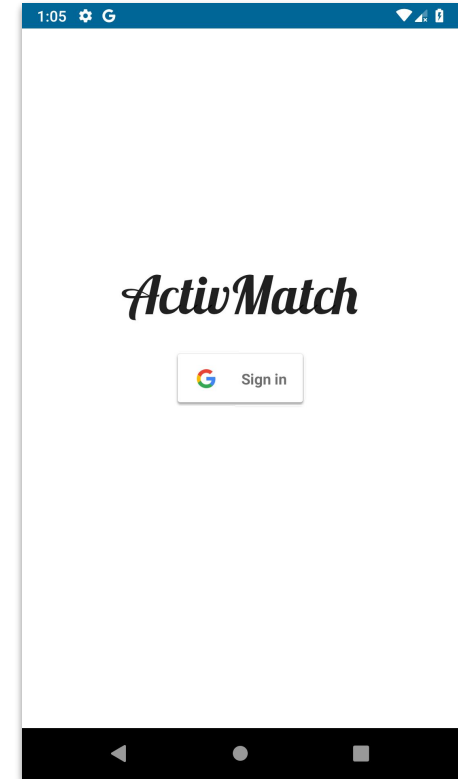
- **activmatch** (*root folder*)
  - **adapter** (*generic adapter used throughout the app*)
  - **io** (*network request and local storage*)
  - **models** (*entity models use in the app*)
  - **notifications** (*logic for the match notification*)
  - **ui** (*user interface tools to avoid copy pasting code*)
  - **utils** (*utility classes for various purposes*)
  - **activities** (*10 activities, including 1 super activity*)

# Network services

- Common interface shared between the mock and the real services
- Mock services
  - store everything in the user phone instead of sending a network request to our server
- Goal: single line change when the server is ready

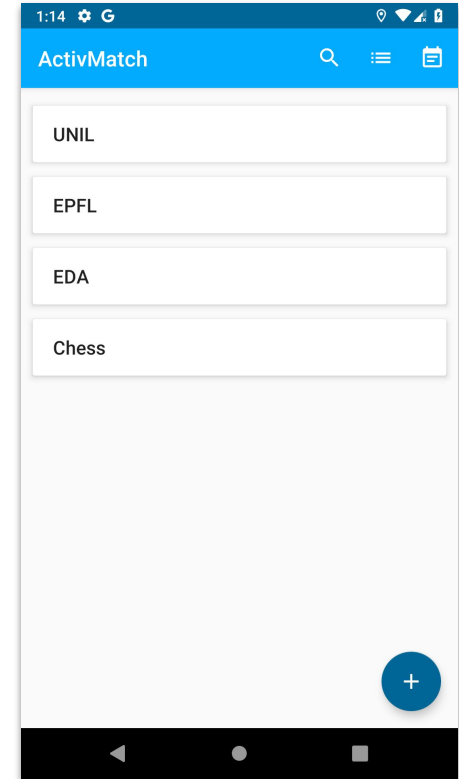
# Sign In

- Visible after a splash screen
  - skipped if the user is already signed in
- Users need to sign-in with their google account.
- Service call made to store user id and name.
- Keep the user id and name in the phone storage
- User id is used for several requests throughout the app



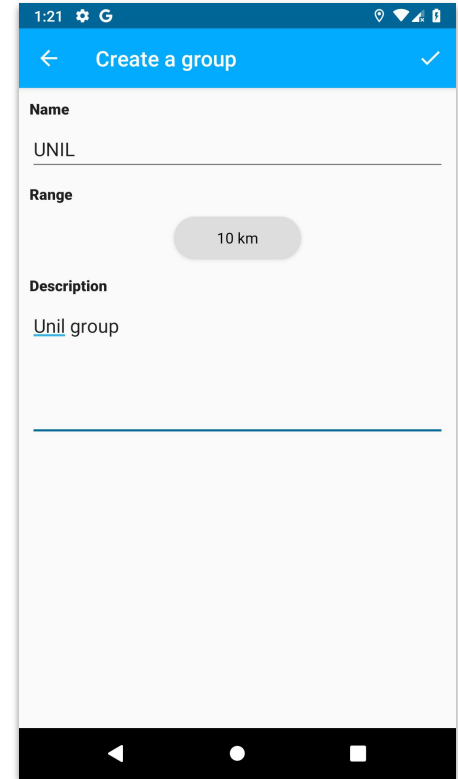
# Main view

- List of the user groups
- The list is return by the service → Only a subset of information is returned (Heading)
- For now it is recovering the ones stored locally
- Button to create a group (publication)
- Three actions in the action bar
  - Subscribe to a topic
  - View matches
  - Update user status (only locally for now)



# Create Group

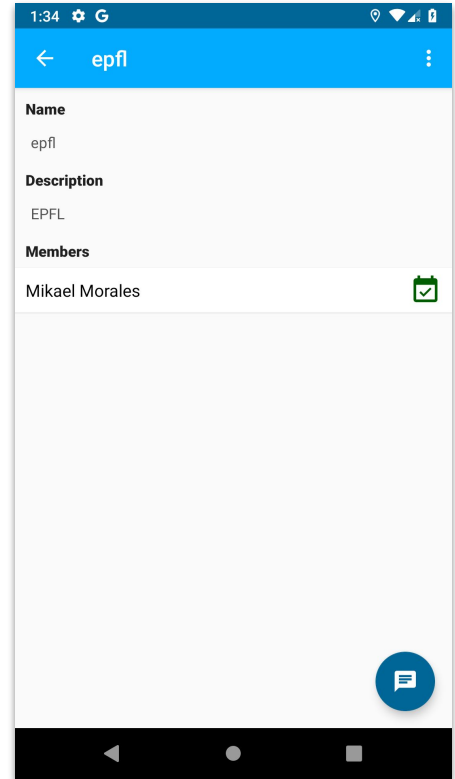
- Requires to define the following properties:
  - Name
  - Range
  - Description
- Creates a publication (MatchMore)
  - Topic: “ActivMatch” (constant value)
  - Duration: 1 year (not modifiable)
- Creates a group (locally) using the UUID returned by Matchmore.
- Will update the content of the main view when going back



The screenshot shows a mobile application interface for creating a group. At the top, there is a blue header bar with a back arrow, the text "Create a group", and a checkmark icon. Below the header, the form is divided into sections: "Name" with the text "UNIL", "Range" with a grey pill-shaped button containing "10 km", and "Description" with the text "Unil group". A horizontal blue line separates the description section from the bottom of the form. The bottom of the screen shows a black navigation bar with three white icons: a back arrow, a circle, and a square.

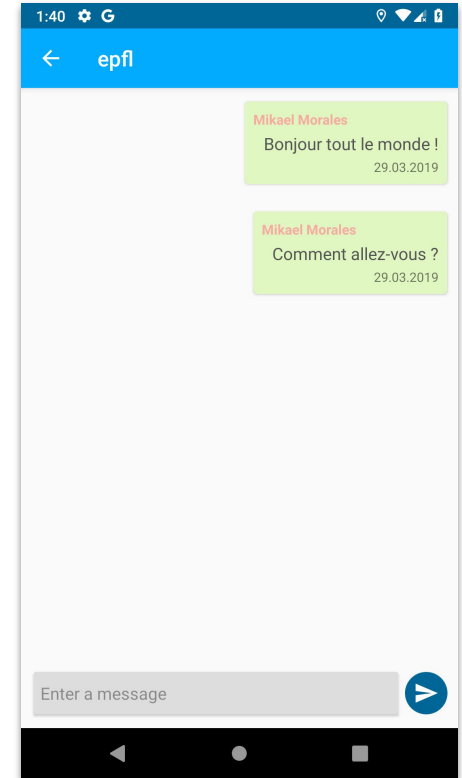
# Group details

- Show the name and description of the group
- List of members with their status (only one member since local)
- Ability to quit the group (top-right)
  - If the user is the admin → deletes the publication (group can't grow anymore)
  - If a simple user → Leaves the group
- Removes the group from user device when leaving
- Button to access the chat of the group



# Group Chat

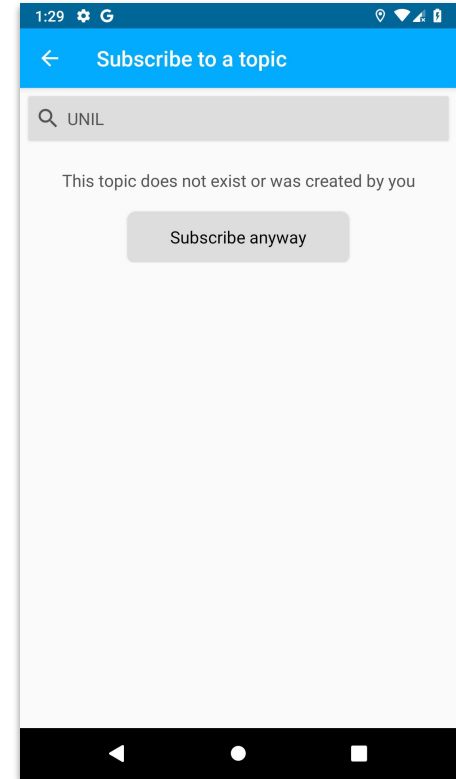
- List of the messages in the group
- Only local message for now (no interactions)
- Two different views:
  - Own messages on the right side
  - Others messages on the left side
- Poll every 5 seconds to check if new messages exists and add them at the end





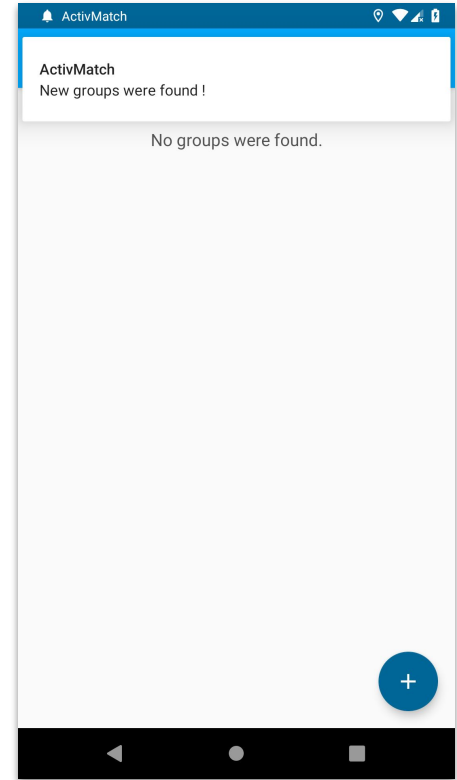
# Search for topics

- Search bar to enter group name (case insensitive)
- Display current groups that matches the name as suggestions.
- If no topic exists, allows the user to subscribe to it anyway
- Subscribing to a topic:
  - Topic: "ActivMatch"
  - Name: name LIKE '%<GROUP\_NAME>%'



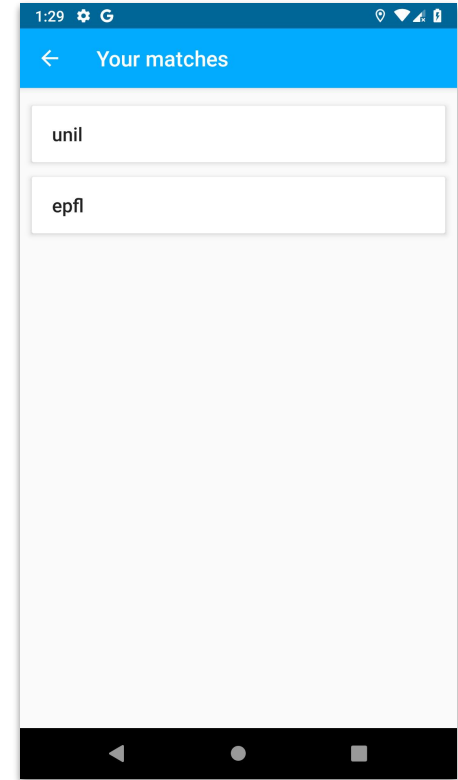
# Your Matches

- Notify the user when a match occurs
  - Create a notification locally using Matchmore match listener.



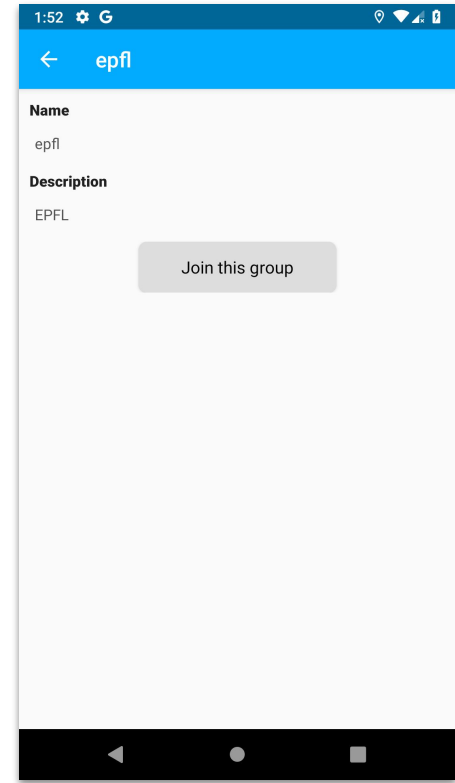
# Your Matches

- Notify the user when a match occurs
- Display the list of matches returned by Matchmore



# Group Previews

- Allows the user to see group details to decide if he wants to join or not
- Can see the members of the group only by joining the group.



# Difficulties

- Integrating MatchMore PUSH notifications using Matchmore pushers.
- All view needs to define a match listener to be able to notify when a match occurs regardless of where he is in the app.
- Architecture of the app: services & local storage needed to be accessible in every activity
- Avoid as possible Android boilerplate code by implementing generic solutions.
- Compelling user experience by providing asynchronous queries and a simple interface design
- Block the UI when needed to avoid wrong behaviors (e.g when creating a group).



Questions?

