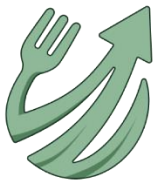
SY43 PROJECT REPORT – CALORIE CREW

Ivann VASIC – Swann GLOTIN – Rémi BERNARD – Mourad KETATA



Outline

1.Project Specification

1.1 Description of the Chosen Application Theme

CalorieCrew is an Android application designed to assist users in managing their health and nutrition by tracking their daily calorie intake. The concept is basically helping individuals to set and achieve personal health goals through simple and intuitive functionalities.

The theme of the application is centered on **well-being and fitness**.

CalorieCrew stands out by emphasizing:

- **Simplicity**: The user interface is streamlined to allow users to log meals, view their calorie consumption, and track progress with minimal effort.
- **User-friendliness**: The onboarding flow (gender, physical activity, goal, height, weight, birthdate) customizes the user profile to provide personalized feedback and calorie targets.
- **Data-driven insights**: The application calculates recommended daily calorie intake based on user information and provides visual feedback on progress through weekly progress charts and streak.

The application also lays the foundation for **community engagement features**, such as adding friends, joining or creating fitness communities, and launching challenges between communities.

In summary, CalorieCrew's theme combines personal health monitoring with social interaction, aiming to promote healthier lifestyle habits through digital tools.

1.2 Target Users and Expected Use Cases

Target Audience:
CalorieCrew is primarily aimed at **young adults and students** who wish to monitor and improve their dietary habits. These users often face constraints such as lack of time, limited nutritional knowledge, or inconsistent motivation, making them ideal candidates for a simple, accessible calorie tracking app.

Typical Users:

- **Fitness Enthusiasts:**
  Individuals who regularly exercise and want to ensure their caloric intake aligns with their fitness goals, such as muscle gain or fat loss. They value accurate tracking and personalized feedback.
- **Users Seeking Social Motivation:**
  Many individuals struggle to maintain healthy habits without support. By integrating social features such as adding friends, joining communities, chatting, and engaging in challenges, CalorieCrew addresses this by providing a **community-driven experience**.

Expected Use Cases:

- **Daily meal logging:**
  Users input their meals to track calories and stay within a recommended range regarding their personal profile.
- **Progress visualization:**
  Users can visualize their improvements over time, with the system of daily streak and weekly progress to evaluate how close they are to reaching their goals.
- **Community engagement:**
  Users can invite friends, create or join communities, and participate in shared challenges to stay motivated.

1.3 Functional and Non-Functional Requirements

Functional Requirements

The main functional requirements include:

- **User Registration and Authentication**
  Users can create an account or sign in using email and password. Firebase Authentication ensures secure access.
- **Personal Profile Setup**
  Upon registration, users define their gender, height, weight, workout habits, and health goal (e.g., weight gain, loss, or maintenance). This data is used to calculate a personalized daily calorie target.
- **Daily Meal Logging**
  Users can manually log meals by entering a food description and its calorie content. Entries are stored and displayed with timestamps.
- **Real-Time Calorie Tracking**
  The application calculates the total calories consumed for the current day and displays progress toward the user's daily goal using a circular progress bar and visual indicators.
- **Community Creation and Joining**
  Users are able to create or join communities to share progress, participate in challenges, and engage socially, by sending messages.
- **Friend System**
  This feature allow users to add friends, in order to motivate each others and achieve their goals more easily.
- **Challenge Creation**
  Community manager are able to create challenges within communities and every users can participate to stay motivated.

Non-Functional Requirements

To ensure a smooth and scalable user experience, several non-functional requirements guide the design and development of CalorieCrew:

- **Fast and Responsive UI**
  Built entirely with **Jetpack Compose**, the app delivers modern, fluid, and reactive user interfaces without relying on traditional XML layouts.
- **Cloud Integration with Firebase**
  Firebase is used for both authentication and data storage (Firestore), enabling real-time syncing of meals, user profiles, and community data across devices.
- **Data Persistence and Multi-User Support**
  Each user's data is isolated and securely stored in the cloud, allowing multiple users to interact with the app simultaneously without interference.
- **Scalability and Maintainability**
  The use of MVVM architecture, modular components, and coroutines ensures that the app remains maintainable and adaptable for future extensions like chat or notifications.

2. Conceptualization of the Application

2.1 System Architecture and Main Components

CalorieCrew follows a **Model-View-ViewModel (MVVM)** architecture, which promotes a clean separation of concerns and simplifies testing, scalability, and maintainability of the app.

Architecture Overview

The system is organized into the following layers:

- **View (UI Layer)**
  Built entirely using **Jetpack Compose**, the View layer displays all UI components and reacts to state changes from the ViewModel. Each screen is a @Composable function, often parameterized via navigation.
- **ViewModel (Logic Layer)**
  The ViewModel contains the application's logic and state. It processes user inputs, handles data transformations, and communicates with repositories.
- **Repository (Data Access Layer)**
  The repository serves as an abstraction between the ViewModel and the Firebase backend. It handles Firestore operations (reading, writing, deleting) and simplifies database access.
- **Firebase Backend**
  Firebase Authentication is used to manage user sessions securely. Firestore is used to store and retrieve user profiles, meals, and future data like messages and community memberships.

Data Flow

The typical flow of data is:

1. **User Action in View** (e.g., logs a meal or signs in)
2. View triggers a **function in ViewModel**
3. ViewModel calls **Repository** to fetch or save data
4. Repository communicates with **Firebase Firestore**
5. Data is returned back through ViewModel and reflected in the View

Navigation Stack

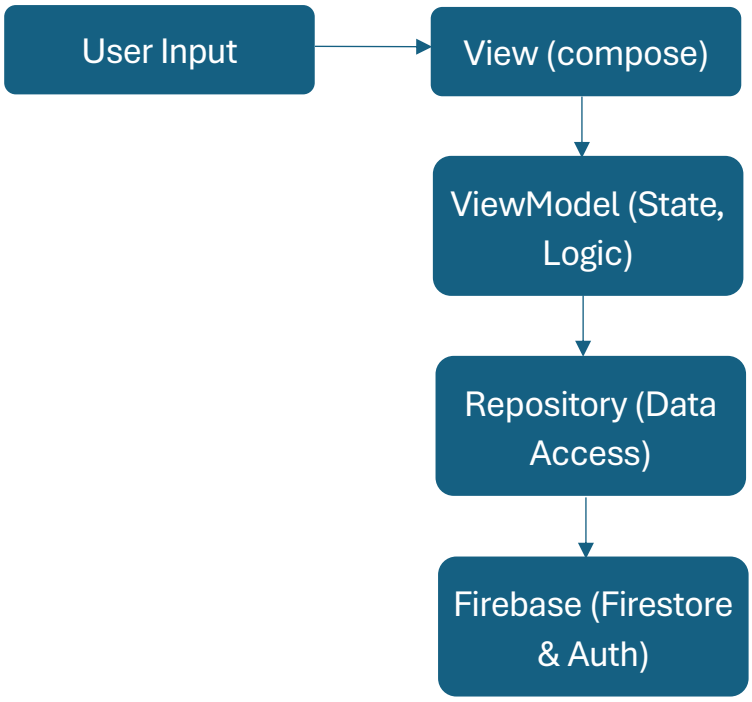
Jetpack Compose Navigation is used to move between screens. Each screen is defined with a route and receives parameters via navigation arguments (e.g., gender, weight, birthdate).

Screens follow a hierarchical flow:

Home → Gender → Workout → HeightWeight → Birthdate → Goal → SignUp → Menu

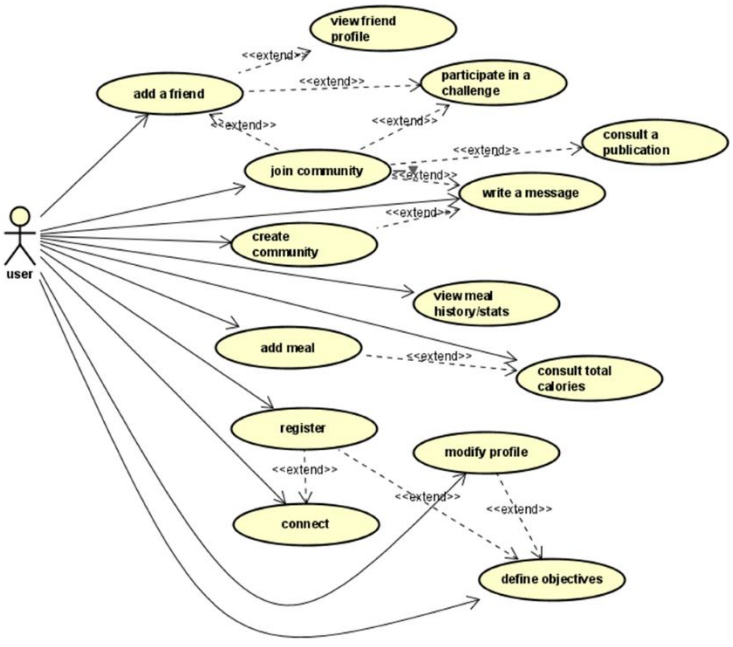
All transitions are handled using NavController, with popBackStack() used to return to previous screens.

System Architecture Diagram



2.2 UML Diagrams

2.2.1 Use case Diagram

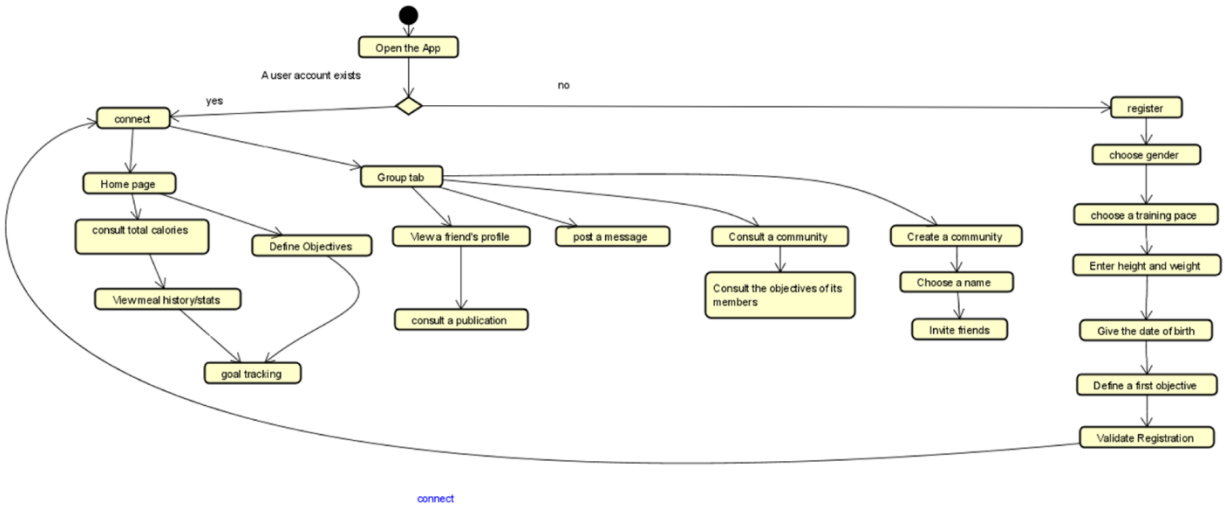


This **use case diagram** illustrates the main interactions between a user and the CalorieCrew application.

It includes three key areas:

- **Account management**: users can register, log in, and edit their profile or objectives.
- **Nutrition tracking**: users can add meals, view their meal history, and monitor total calorie intake.
- **Community features**: users can add friends, create or join communities, chat, consult publications, and participate in challenges.

2.2.2 Activity diagram



This activity diagram shows the overall user flow in the CalorieCrew application. It distinguishes two paths:

- **New users** go through a guided registration process including gender, workout pace, physical data, and goal setting.
- **Returning users** are directed to the home screen, where they can access calorie tracking features and navigate to the group tab.

From the group tab, users can view friend profiles, write messages, create or join communities, and interact with other users' objectives and publications.

This diagram highlights the linear registration process and the modular navigation within the app once connected.

2.3 Database Schema

Our application uses Cloud Firestore to store and synchronize data in real time. The data model is organized into several collections, each with its own structure and purpose. Below are the main collections used in the app:

- **1. users**

Holds profile data for each registered user.

Fields:

- id: user ID (same as Firebase UID)
- email: user's email

- name: display name
- gender, birth, height, weight: personal information
- goal: e.g., "GAIN", "LOSE"
- workoutHabits: e.g., "MED" for medium workout frequency

- **2. mealHistory**

Stores daily food intake per user.
Each document represents a meal consumed, associated with calorie count and metadata.

Fields:

- mealId: unique identifier of the meal
- userId: ID of the user who logged the meal
- mealDescription: description of the meal
- calories: number of calories
- date: date of consumption

Purpose: Enables users to log and track meals for nutritional analysis.

- **3. communities**

Stores all community groups users can join or create.

Fields:

- id: unique ID of the community
- name: community name
- goal: lose weight / gain weight / maintain weight
- joinCode: 6-digit code for joining
- ownerId: ID of the user who created the community
- public: Boolean to define if community is open

Purpose: Enables group interaction, challenge organization, and social motivation.

- **4. friendRequests**

Handles pending friend requests between users.

Fields:

- senderId: user who sends the request
- receiverId: user receiving the request

- status: "pending", "accepted", or "declined"
- createdAt / updatedAt: timestamps for request tracking

Purpose: Supports building a social network within the app.

- **5. friendships**

Represents a confirmed relationship between two users.

Fields:

- user1Id, user2Id: two user IDs involved in the friendship

Purpose: Allows for interaction, such as chat or challenge invitations.

3. Implementation Details

3.1 Tools and Technologies

The development of CalorieCrew relied on a modern Android technology stack composed of Kotlin, Jetpack Compose, and Firebase. From the outset, the team followed a Model-View-ViewModel (MVVM) architecture to maintain a clean separation of concerns and to facilitate future scalability and maintainability. Git & GitHub were used for version control and collaborative development. We followed a Git branching strategy with feature branches and regularly merged updates to ensure continuous integration.

3.2 Technical Challenges

One of the key technical challenges was the implementation of Firebase Authentication. Integrating its token-based session mechanism into a reactive interface required time and multiple refinements. Specifically, coordinating onboarding flows, persisting session state, and ensuring a smooth experience across launches proved complex and involved extensive testing.

Another difficulty involved managing Agile-inspired sprints. Accurately estimating development time for components such as UI transitions, calendar integration, and complex state management was more difficult than anticipated. The team occasionally missed sprint targets but learned to improve their workflow through better coordination and more granular planning.

3.3 Firestore Architecture and Repository Layer

The application stores user and community data using Firebase Firestore. A major improvement during development was the refactoring of the database schema: all community-related data—members, messages, and challenges—was nested under a unified "communities" collection. This restructuring optimized access patterns and reduced redundant queries.

Firestore interactions are abstracted through a dedicated Repository layer. This abstraction improved testability, reusability, and modularity. Real-time updates in features like chat and challenge participation were implemented using Firestore snapshot listeners. These listeners were scoped carefully to avoid memory leaks and minimize unnecessary UI recompositions.

3.4 Key Functional Implementations

Functionality evolved in stages. The user authentication and onboarding flow was prioritized first to ensure the app could provide personalized calorie goals. Next came the meal logging system, including daily total computation and visual feedback through a circular progress bar. A separate weekly overview component provided a clear summary of user consistency.

Community features included public group discovery (filtered by goals), invite-based joining, real-time chat, and challenge creation. Each challenge includes a name, deadline, and an integer reward that contributes to a community's total score once completed. State synchronization

across members was managed through a combination of Compose state management and Firestore snapshots.

Custom UI components were created to ensure visual consistency, especially for displaying progress data and supporting Material 3 guidelines throughout the app.

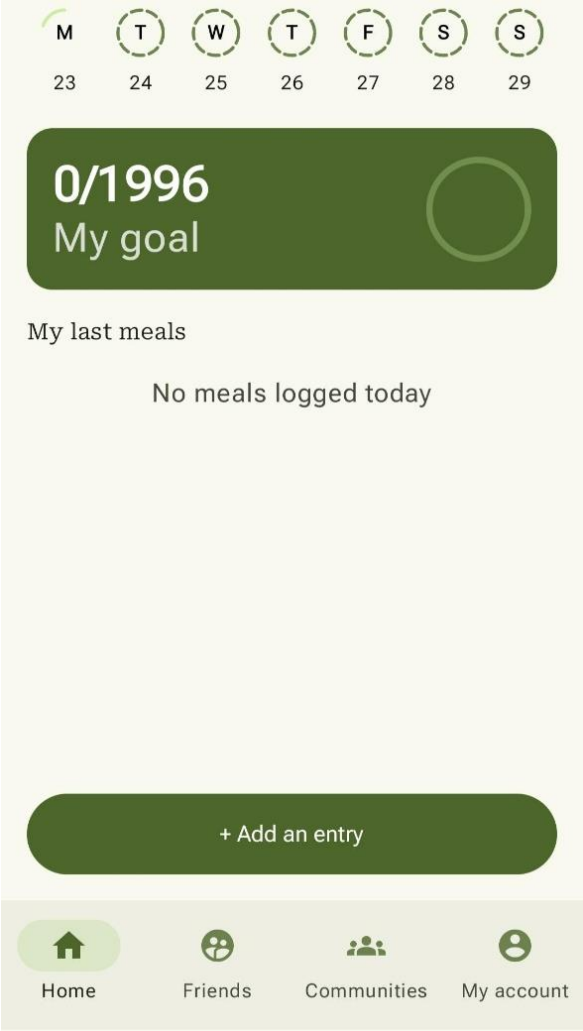
4. User Interface

4.1 Onboarding and Navigation

CalorieCrew's user interface was designed with a focus on clarity, responsiveness, and user guidance. Built exclusively with Jetpack Compose, the application offers a reactive and modern interface that adapts to state changes and user inputs in real time.

Upon launching the app, new users are guided through a multi-step onboarding flow, where they input personal data such as gender, activity level, weight, height, and health goals. After completing registration, users land on the Home screen, the main hub for calorie tracking.

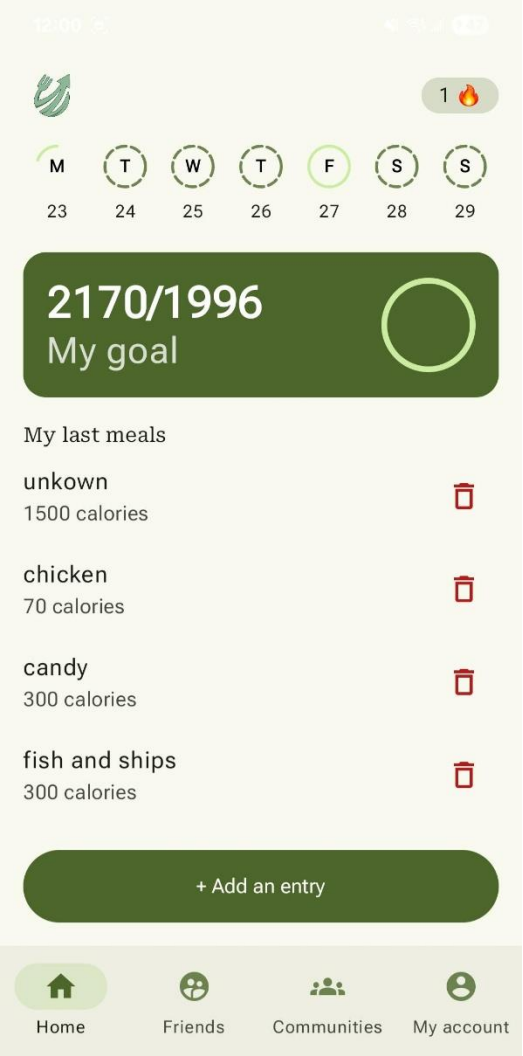
A bottom navigation bar allows seamless access to four core areas: Home, Friends, Communities, and Profile. Navigation is intuitive and transitions between screens are smooth, supporting both casual exploration and direct task execution.



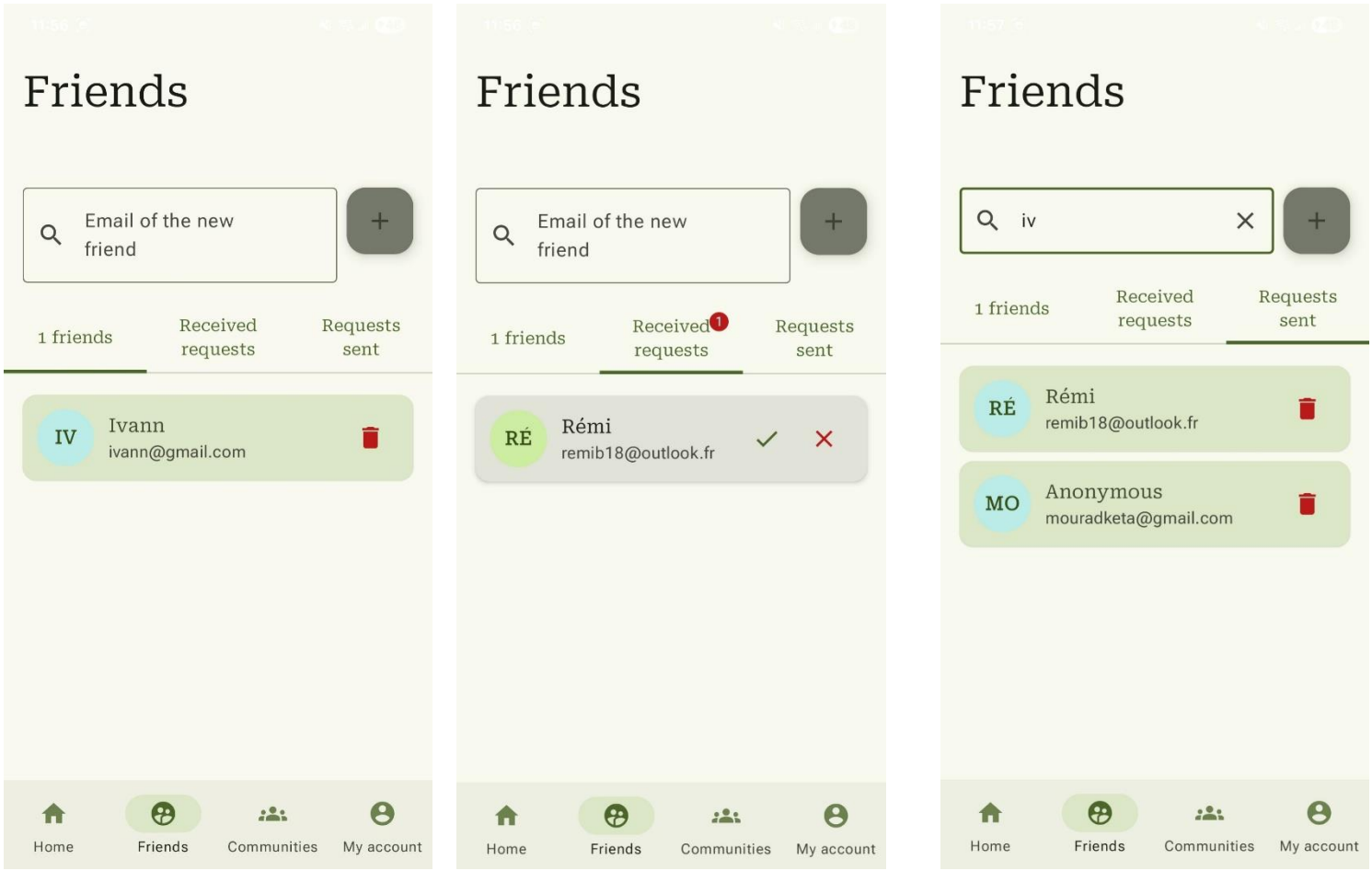
Home screen

4.2 Main Screens and User Flow

On the Home screen, users can view a large circular progress bar indicating their daily calorie intake relative to their target. Below, a horizontal row of smaller circular indicators provides a visual summary of calorie tracking over the week. A streak counter is prominently displayed, motivating users to meet their goals on consecutive days.



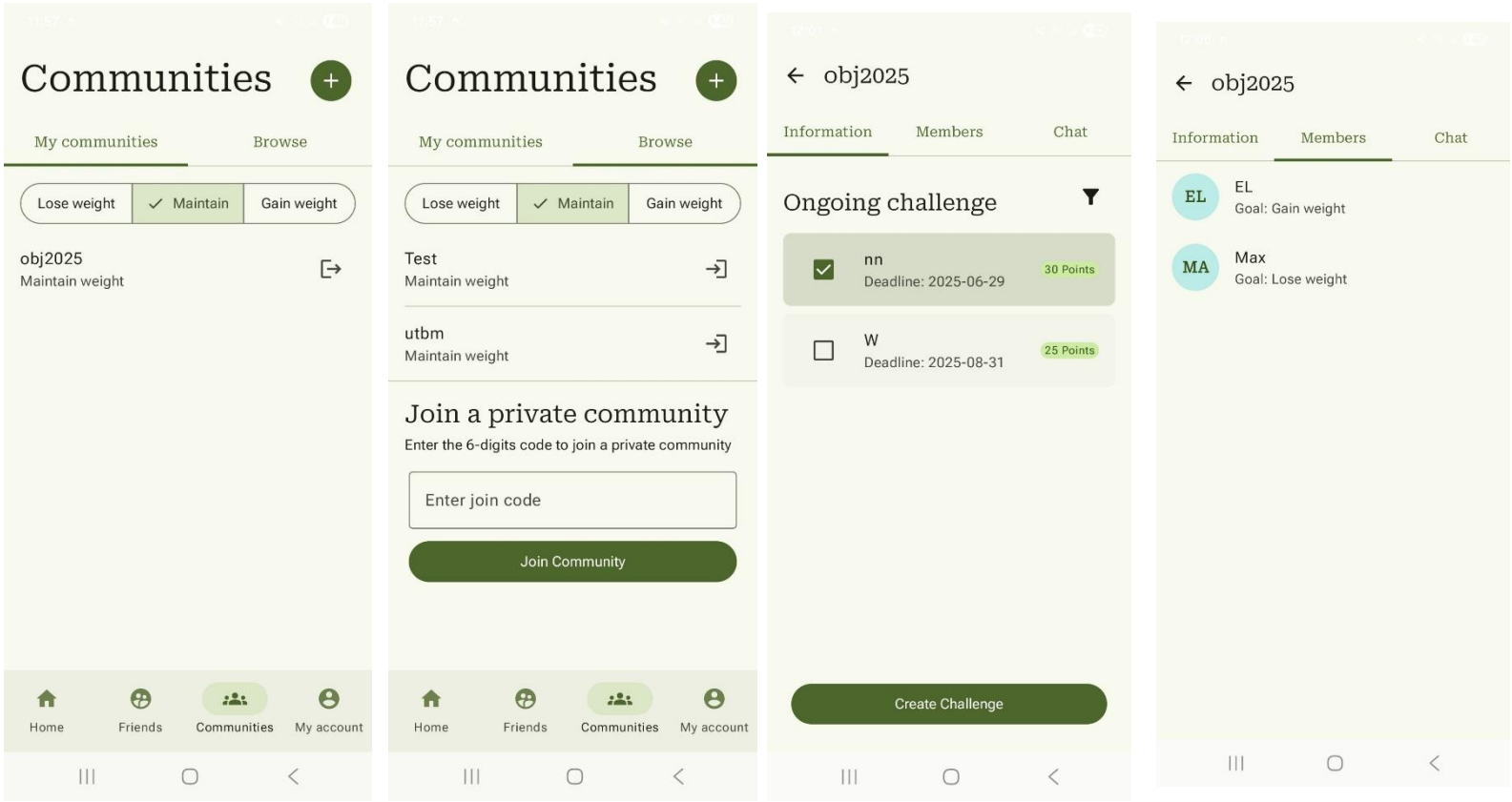
In the Friends tab, users can search for friends, send requests, and monitor their contacts' activity.



Friends screen including show friends, add friends, accept friends

The Community section lists public groups sorted by objectives. Each community card shows its name, member count, and accumulated score. Users can tap a card to access community features like chat, challenges, and member lists.

Within each community, users can chat in real time, view active challenges, or inspect the full member list. Users with admin rights can also create new challenges via a dedicated form. This form allows entry of a challenge name, a numerical reward (which contributes to the community's score), and a deadline selected from a calendar picker.



Some of the Community Screens

4.3 Visual Design and Components

Visually, CalorieCrew follows Material 3 design principles, featuring rounded cards, soft shadows, adaptive layouts, and consistent color themes. Text fields include validation logic, while feedback is provided through snackbars and subtle animations.

Compose's LazyColumn and LazyRow components ensure smooth scrolling and dynamic rendering of lists like meals, friends, and messages. Custom Compose elements such as the circular progress bars reinforce a unified visual language across the app.

Responsiveness was a key design focus. The UI adapts gracefully to different screen sizes and orientations, ensuring a consistent user experience. The combination of state-driven UI and Compose's declarative syntax allowed for fast prototyping and reliable rendering.

Overall, the interface supports CalorieCrew's mission of combining personal wellness tracking with community-driven engagement, providing users with an app that feels both personal and socially connected.

5. Testing and results:

5.1 Test Scenarios and Validation

To make sure that the application was reliable and user-friendly, we defined several test scenarios covering core features. These scenarios were tested manually across emulators and physical devices.

Feature	Test Scenario
User Registration	New user inputs data, create an account, and lands on home screen
Login/Logout	User logs in, session is remembered, then logs out and is redirected properly
Meal adding	User adds, edits, and deletes meals;
Weekly Progress	User sees correct progress through circular progress bar and streak count for the week using flames
Profile Editing	User modifies height, weight, or goal; changes are saved in Firestore
Community Creation	User creates a community; others can join and see its objectives and access the chat
Challenge System	Admin creates challenge; all members see it
Chat Feature	Messages are sent and received in real-time in communities
Friend System	Users search and add friends; friend requests are stored and updated

We also added specific tests after the oral presentation:

- UtilsTestSimple: tested core utility functions such as calorie calculations, activity level and goal multipliers, as well as date conversions.
- ValidationTestSimple: verified registration validation logic, including required fields, age range, height/weight consistency.
- LoginFlowTest: tested the complete login navigation and Firebase response.
- UserProfileEditTest: tested Firestore updates after modifying user data.

5.2 Bugs Encountered and Fixes

Although most features worked as expected, it was normal to have a few bugs and usability issues during testing, and we resolved them during development.

Sample Bugs and Fixes :

Communities not updating after joining via join code
→ After using a join code, the UI wasn't reloading the updated list of communities.
Fix: We added a call to getCommunitiesForUser() immediately after joining

Community creator not seeing their own communities

→ The filtering logic for "My communities" only showed communities where the user was a member, not the creator.
Fix: We updated the Firestore query to also include communities where ownerId == currentUserId

Friend requests not showing immediately

→ Although the data was saved in Firestore, the UI didn't reflect it in real time.
Fix: Triggered an explicit data refresh after sending/accepting a friend request.

Meal entry not updating the screen immediately
→ When adding a new meal, it was not instantly shown in the list on the home screen.
Fix: Called refreshTrigger++ and updated the local meal list (mealsToday.add(...)) to reflect changes immediately.

Navigation crash when quickly clicking back after signup
→ On some devices, navigating too quickly after account creation caused a crash due to ViewModel state not yet updated.
Fix: Added a delay using LaunchedEffect to wait for user data to be fully loaded before navigation.

Incorrect calorie goal calculation for some profiles
→ When users left height or weight empty, the daily calorie goal calculation crashed.
Fix: Added safe parsing and default fallbacks to prevent calculation errors.

6. Conclusion

Working on CalorieCrew was our first real experience creating a mobile app from scratch, from designing the features to writing the code in Kotlin and building the interface with Jetpack Compose. We set out to make something useful and intuitive and that could be appreciated and used in real life if we add a few more features, we're proud of what we accomplished: an app that helps people track their calories and stay motivated through community challenges.

Along the way, we ran into real challenges like managing real-time data with Firestore, and making sure everything flowed smoothly from screen to screen. But each obstacle helped us make the app better. We also learned how to work better as a team planning our sprints, organizing our code together, and testing features in a more structured way.

The final result brings together everything we learned during this semester. There's still room to add features like notifications, a real food database or food scanning, but the app is stable and ready to be extended.

Overall, this project was much more than a coding exercise it was a chance to discover mobile development, challenge ourselves, and build something we're genuinely proud of.