Design and Implementation of Mobile Application

Luca Lain, Segio Lupo

Professor Luciano Baresi

AY 2023/2024

POLITECNICO DI MILANO

Design Document

AgorApp

Contents

1. Introduction

1.1. Project description

AgorApp is a versatile social application designed to bring together individuals with common interests across different categories. Users can join or create groups focused on specific topics, facilitating meaningful interactions and exchanges of ideas.
Beyond its chat functionalities, AgorApp offers comprehensive event management features. Users can organize and participate in events, promoting real-world meetups and fostering community building and networking.
Moreover, AgorApp keeps users informed by providing news updates related to their areas of interest. Through a personalized news feed, users can stay up-to-date with the latest developments and trends relevant to the topics they care about.

The application is divided into five main sections:

- **News section**: this area features two primary collections: one for breaking news and another for trending news. Additionally, it includes a curated selection of articles tailored to the user's specific interests, ensuring they stay informed with the most relevant and current updates. Users can also search for specific news items and share articles directly from this section, enhancing their ability to engage with content and share updates with others.

- **Chats section**: in this section, users can handle both group and private chats. Group chats show all the discussions happening in the groups the user belongs to, while private chats allow one-on-one conversations. Users can also create new groups here.

- **Calendar Section**: this section shows all the events related to the user, including both upcoming and past events. It covers events the user has created and those he/she has attended, providing a complete schedule. In this section it is also possible to create events.

- **Search Section**: this area allows users to search for and explore accounts, groups and events within the application. It also helps users navigate to profiles and event pages.

- **Home Section**: this section displays the user's profile along with the events he/she has created and joined.

1.2. Features

1.2.1. Log in and Registration

Users can sign in or sign up using their email address or through Single Sign-On (SSO) with Google as their identity provider. This flexible authentication approach supports both standard email-based login and access via Google credentials.
Additionally, the application features a "Forgot Password" protocol, which allows users to reset their passwords if they encounter issues accessing their accounts.

1.2.2. Navigate and Search news

Registered users can navigate and search for news articles by entering relevant keywords. Each news article is displayed with its title, description, and accompanying image. Users can tap on any news item to read the full article within the app.

1.2.3. Create Groups

Registered users can create groups by providing a name and description, with the option to upload a group profile image and select relevant categories. During the creation process, they can also invite their followers and decide whether the group will be public or private. For private groups, the group admin has the authority to approve or reject membership requests. The group creation page is accessible in the chat section by clicking the add button in the top right corner.

1.2.4. Edit Group

A group admin can update several elements of the group, including its name, description, associated categories, and profile image. Additionally, the admin can adjust the group's visibility settings, switching between private and public, and invite his/her followers to join the group. The group editing page can be accessed by clicking on the tile at the top of the group's message page and then selecting the corresponding option.

1.2.5. Create an Event

Registered users can create and manage events with various customizable options. When creating an event, users provide details, including an event name, description, initial and final dates and times and a location. The event creation process also allows users to upload an event profile image and choose whether the event will be public or private. Users can add multiple dates and locations for recurring events or events with multiple venues. They can also invite their followers and groups, extending invitations to individual followers or specific groups to participate in the event. To create an event, navigate to the group's message page, click the add button positioned in the left side of input bar, and select the corresponding option. Alternatively, an event can be created from the calendar page by clicking the add button positioned in the top-right corner.

1.2.6. Edit Event

The host of an event can edit the event's name, description and profile image. This editing page is accessible by clicking the appropriate button on the top-right of the event page.

1.2.7. Share News and Events

Registered users can share news and events with their groups or followers by selecting the appropriate tile located in the central part of the screen, which appears after tapping the button in the top-right corner of the article or event page.

1.2.8. Follow and Unfollow users

Registered users can follow or unfollow other accounts by clicking the appropriate button located in the central part of the screen that appears when a user visits another user's profile. However, if the visited account is private, the user must request to follow it, and the other user will need to accept the request, that can be found on the page that appears after clicking the button in the top right corner of the home section and then selecting the designated request tile.

1.2.9. Join and Leave groups

Registered users can join or leave groups by clicking the appropriate button located in the central-right part of the screen that appears when a user searches for a group by name in the designated section on the search page. However, if the group is private, the user must request to join and the group's admin will need to approve the request. The admin can view requests on the page that appears by clicking the tile at the top of the group's message page and then selecting the corresponding tile.

1.2.10. Subscribe and Unsubscribe events

Registered users can subscribe to or unsubscribe from events by clicking the appropriate button in the center of the screen that appears when they tap the date on the event page. For private events, users must request to join and the event host must approve the request. The host can manage these requests on the same page by clicking the relevant tile.

1.2.11. Chatting

Registered users have access to the chat section of the app, where they can view both group and private conversations. On this page, users see a list of group chats they have created or joined, as well as a list of private chats. Each chat tile displays the most recent message, the date of that message and the number of unread messages. Tapping on a tile opens the corresponding conversation.

1.2.12. Display news, events, and images in group and private chats

Users can view a list of news, events and images shared in group or private chats by tapping the corresponding button located in the center of the screen that appears after clicking the tile at the top of the chat screen. When tapping on an individual image, users can see detailed information, such as the username of the account that shared it and the timestamp of when it was sent and they can swipe left or right to browse through other images. Additionally, users can navigate directly to an event or news page by tapping their respective buttons.

1.2.13. Message Actions: Delete, Copy and View Read Receipts

The sender of a message can press and hold on a message within the chat to perform several actions. This includes viewing a list of users who have read the message along with the timestamp when it was read, copying the message content, or deleting the message from the chat.

1.2.14. Enable or disable notifications for messages

Registered users can enable or disable message notifications by toggling the switch located in the center of the screen that appears after clicking the tile at the top of the chat messages screen.

1.2.15. Delete groups and private chats

Registered users can delete chats by swiping left along the designated chat tile they wish to remove.

1.2.16. View Calendar

Registered users have access to a calendar section that displays a calendar, which features all events the user has created, as well as events they have been attended. Users can navigate through different dates to see a comprehensive overview of their upcoming, past and current events.

1.2.17. View user profile

Registered users can view their own profile as well as the profiles of other users. On this page, the top section features the user's profile picture, first name, last name and username, accompanied by three buttons. Clicking these buttons reveals lists of the groups the user belongs to, the people they follow and the people who follow them. The central section showcases the categories the user selected during registration, while the bottom section displays a grid of events the user has created or registered for, depending on the selected button.

1.2.18. Manage Profile Settings

Registered users can manage their profile settings. In the upper right corner of the user profile page, there is a button that directs users to a settings page where they can manage various aspects of their account. On this page, users can:

- view and respond to follow requests from other users and invitations from groups they have been invited to join.
- update profile information, including first name, last name, username, categories of interest and toggle the profile visibility between private and public.
- delete their account.
- log out of their account.

1.2.19. Receive Notifications for your preferred categories

Registered users receive notifications, unless disabled, when new articles are published in their preferred categories. These notifications are dispatched every 4 hours, but only if there is new content in a category since the last notification.

1.3. Purpose

This document aims to provide a comprehensive overview of AgorApp, with a specific focus on the architecture.

Additionally, a series of interaction diagrams will illustrate the overall application workflow and the messages exchanged between system components.

Lastly, the document will cover the requirements, user interface design and testing.

1.4. Document Structure

- Section 1: Introduction
  This section provides an overview of the document, including a description of the project, its key features and its objectives.
- Section 2: Architectural Design
  This section delves into the system architecture, offering detailed descriptions of data modeling, data management strategies and interactions with external services. It includes sequence diagrams to illustrate the system's design and workflows.
- Section 3: User Interface Design
  This section presents the main screens of the user application, accompanied by brief annotations highlighting key elements and functionalities.
- Section 4: Requirements
  This section outlines the key requirements that AgorApp must fulfill, along with brief descriptions of each.
- Section 5: Testing Campaign
  This section outlines the procedures employed to test the components of our application, featuring tables that provide details on test coverage and the number of tests conducted.
- Section 6: Future Developments
  This section discusses potential future enhancements and expansions of the system, highlighting areas for improvement and further development.

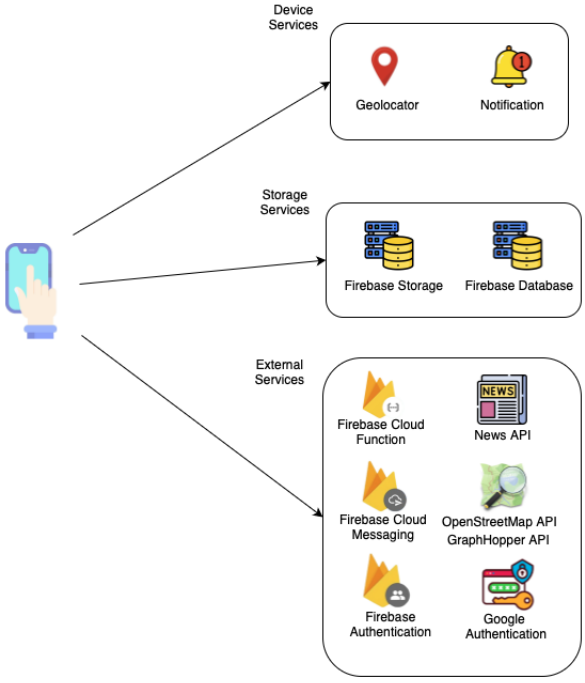
2. Architectural Design

2.1. Overview

We have chosen to develop AgorApp with Flutter, an open-source framework created by Google that allows for the development of natively compiled, cross-platform applications using a single codebase. Flutter is built on Dart, a language specifically optimized for creating high-performance apps across various platforms.

For the development of the application, several key services will be utilized. These services are categorized into three main types:

- Device Services: APIs that interact directly with the device's hardware and sensors, such as the camera, GPS and other built-in functionalities. These services are crucial for enabling features that rely on real-time data from the device itself.
- Storage Services: APIs that manage the storage of application data. These services ensure data persistence and reliability across different usage scenarios.
- External Services: APIs provided by third-party vendors or companies that offer additional functionality or data integration. These services extend the application's capabilities beyond what is natively possible on the device.



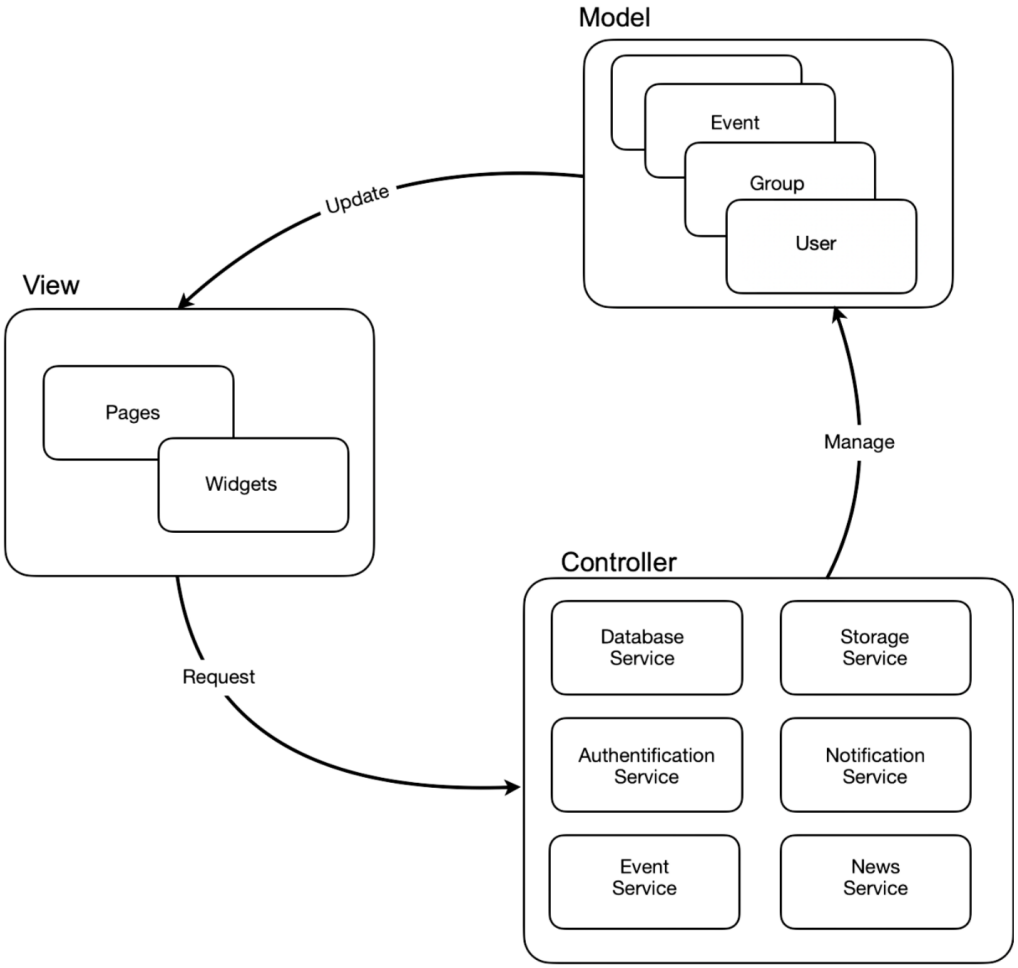
To ensure a smooth and responsive user experience, the integration of these services will leverage asynchronous communication protocols wherever possible. Asynchronous operations, facilitated by FutureBuilders and similar constructs, allow the application to remain responsive even when waiting for data from external services or performing intensive tasks.

To manage the application's state effectively, the Flutter Riverpod provider will be used. Riverpod offers a robust and scalable solution for state management, allowing for clear separation of concerns, easy state manipulation, and efficient dependency injection. This approach not only simplifies state management but also enhances maintainability and testability of the codebase.

2.2. Application Architecture

The application follows a Model-View-Controller (MVC) design pattern, which provides a clear separation of concerns between different components of the application. This pattern is chosen to facilitate modularity and maintainability by distinguishing between the core concepts: Model (data management), View (UI components), and Controller (business logic and user input handling).

This separation allows for each component to be developed, tested and maintained independently, minimizing interdependencies and reducing the complexity of the codebase. Additionally, by isolating the business logic in the Controller, we enable the seamless replacement or modification of external services in a way that is transparent to the rest of the application. This flexibility is critical for supporting future enhancements and integrating new services with minimal disruption.



To align with the MVC pattern and enhance code clarity, the source code is organized in a way that mirrors the logical structure of the application. Each component, whether a service, class, widget, or page, is organized into a directory that corresponds to its role within the MVC architecture. This structured approach to organizing source files ensures that the codebase remains intuitive and easy to navigate, thereby facilitating collaborative development and easing future maintenance and updates.

2.3. Data Modeling and Structuring

The data displayed on the application pages is organized using a series of classes that represent the Model component of the Model-View-Controller (MVC) pattern, as discussed earlier. These model classes are designed to encapsulate the application's data and provide a structured way to manage and interact with this data across different parts of the application.

In alignment with the application requirements outlined in section 4, and based on the design of each page's user interface, the following data model objects have been defined:

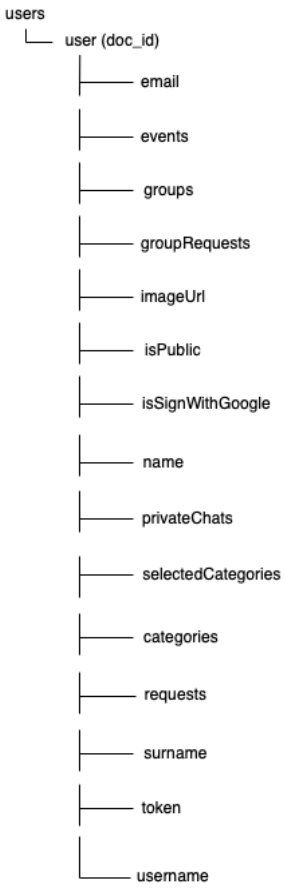
- ArticleModel and CategoryModel: these classes are used to manage the news content displayed within the application. The ArticleModel class includes attributes such as title, description, url and image, representing key elements of a news article. Meanwhile, the CategoryModel class contains the name of the category and an associated image, providing a simple way to categorize news articles.
- EventModel and EventDetails: these classes are designed to manage events within the application. The EventModel class represents the core details of an event, such as its name and description. The EventDetails class, on the other hand, contains specific details for each instance of the event, including location, start and end date and time. This structure supports flexible event management, allowing a single event (defined by EventModel) to have multiple occurrences with different locations or times (defined by multiple EventDetails). For instance, an event might be scheduled at two different locations on the same date, facilitating scenarios where the same event takes place in multiple venues.
- Group: the Group class defines the attributes associated with user groups in the application. It includes key details such as the name, description, and admin information. This class supports various group functionalities, including managing group interactions and facilitating chat capabilities within the group.
- PrivateChats: this model is dedicated to managing private chat interactions between users. It helps facilitate direct communication by structuring chat data appropriately.
- LastMessage, Message and ReadBy: these classes define the attributes and structure for managing messages within both group and private chats. The LastMessage class represents the most recent message in a conversation, while the Message class contains details for each individual message. The ReadBy class records which users have read a particular message, supporting read receipts and message tracking.
- UserData: this class stores user-specific information, including attributes such as name, surname and username. It serves as the primary model for managing user profiles and related data throughout the application.

2.4. Data Management

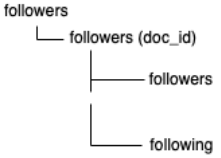
2.4.1. Cloud Firestore

This section provides an overview of the various collections utilized with Cloud Firestore in the application. Cloud Firestore is a scalable, document-based database designed to store and manage the application's data efficiently. It organizes data into collections and documents, offering a flexible and powerful way to handle complex data structures.

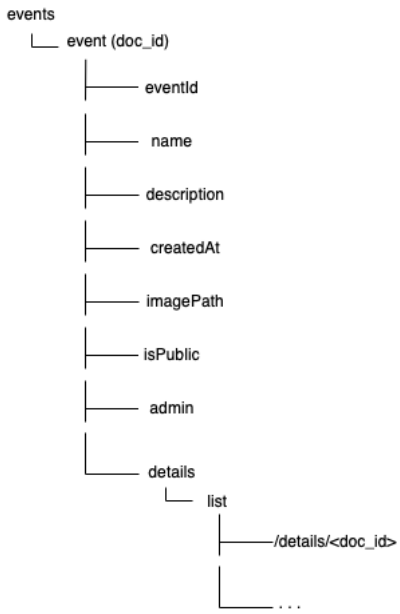
In our application, the internal structure of the Firestore collections is meticulously designed to align with the application's data requirements and functionality. The collections represent different categories of data and are structured to facilitate efficient data retrieval, updates and management. Here's a detailed breakdown of the internal structure of the collections:



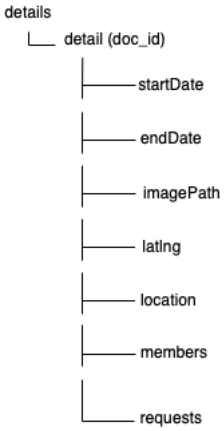
users: this collection manages and stores comprehensive profile information for each user. This includes the user's email address for authentication, as well as their chosen username, first name and last name. Additionally, it stores a URL for the user's profile image. The collection also features a field indicating whether the user's profile is public and another that specifies if the user signed in using Google. Furthermore, the collection includes an array of preferred categories, which reflects the user's interests and lists the IDs of groups, events and private chats the user is involved in. It also contains a notification token for sending push notifications and IDs for any group membership requests the user has received. This structured approach ensures efficient management of user profiles, supports personalized content delivery and facilitates seamless integration with various application features.



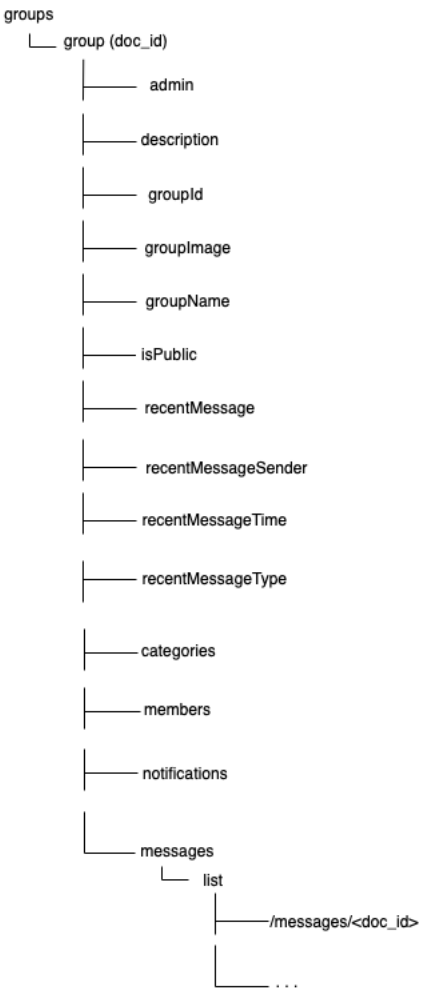
followers: this collection is designed to handle and store information related to user relationships, specifically focusing on who follows whom within the application. This collection is integral to managing social interactions and connections among users. Each document in the followers collection is identified by the user's UID (User ID), which serves as the document's key, ensuring precise and efficient association with the corresponding user. Within each document, there are two key arrays: followers and followings. The followers array contains UIDs of users who follow the specific individual, while the followings array lists the UIDs of users that the individual follows.



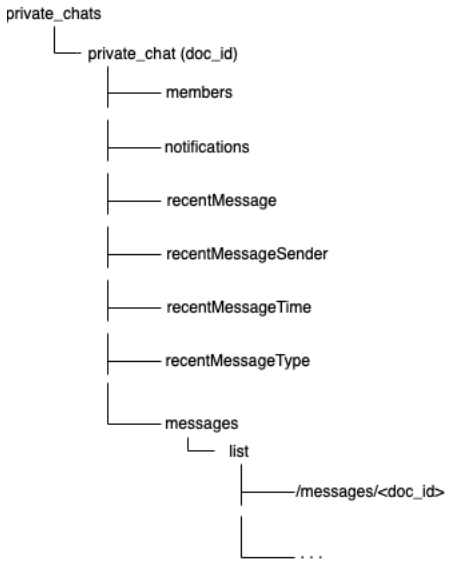
events: this collection is meticulously designed to store and manage comprehensive details about events within the application. Each document in this collection provides a complete profile of an individual event, including its name, a detailed description and the date it was created. Furthermore, each event document includes the identity of the creator and also features an image associated and a field that indicates whether the event is public or private, which helps in managing access and visibility. This collection also includes a sub-collection named details, which will be described later.



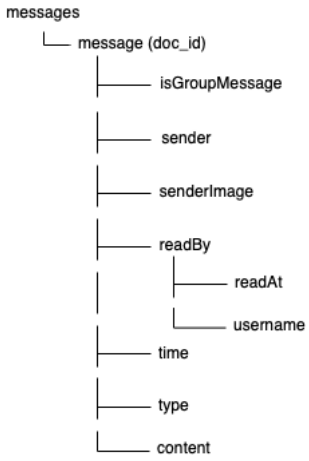
details: The details sub-collection, nested within the events collection, is designed to provide comprehensive information about the specifics of an event. This sub-collection is essential for managing the logistical elements of an event, detailing when and where it will take place. Each document in the details sub-collection includes information such as the geographical coordinates of the event location, which are recorded under latlng. It also specifies the physical location of the event, including the address or venue name. The startDate field indicates the exact date and time when the event begins, while the endDate field marks the conclusion of the event. In addition to these temporal and locational details, the details sub-collection tracks members involved in the event and it also stores the requests if the event is private.



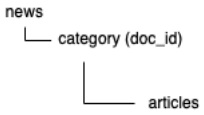
groups: this collection is designed to manage and store detailed information about each group within the application. It includes the group's name and a description that provides an overview of the group's purpose and activities. Additionally, the collection contains information about the group's administrator and a list of members who are part of the group. The collection also features a field indicating whether the group is public or private. Public groups can be accessed by any user, while private groups are restricted. Each group is categorized by some categories, which helps in organizing and classifying groups based on their interests or themes. Moreover, the collection includes details about membership requests. It also provides information about the most recent message sent within the group, offering a quick reference to recent communications. In addition, the groups collection contains fields related to notifications, ensuring that group-related updates and messages are effectively communicated to members. Nested within the groups collection is a messages sub-collection, which stores comprehensive details about the messages exchanged within the group. This includes message content, timestamps, and sender information, supporting efficient management and review of group communications.



private_chat: this collection is dedicated to managing and storing information about private conversations between users. Each document within the private_chat collection represents a distinct private chat session between two users, serving as a container for the chat history between the participants. These documents include details about the chat members, recent message information and notifications. Each document also contains a nested messages sub-collection, which is responsible for storing the actual messages exchanged during the conversation.



messages: This collection functions as a sub-collection within both the group and private_chat collections, serving to manage and store detailed information about individual messages exchanged in these contexts. Within the messages collection, each document represents a specific message and includes essential details such as the message content, the sender and the timestamp of when the message was sent. Additionally, it specifies whether the message was sent in a group chat or a private conversation, providing context for the message's delivery. The collection also tracks who has read the message and it includes information on the type of message, which may indicate whether it is a text message, image, news or event.



news: this collection is designed to store and organize the latest news updates, categorized by category. To ensure that users are notified of significant updates, a cloud function is triggered every four hours. This function monitors changes within the news categories and, when updates occur, it sends push notifications to users. This mechanism ensures that users stay up-to-date with the most recent and relevant news in their areas of interest.

2.4.2. Cloud Storage

Cloud Storage is an essential component for applications that need scalable storage solutions for user-generated content. It is highly effective at managing various types of data, including photos, videos and other media files. In this application, Cloud Storage is primarily used to store and deliver user-generated images, including profile pictures, event photos, group profile images and chat images. Each user's profile picture, a critical aspect of their identity, is securely stored in Cloud Storage and displayed consistently throughout the application's interface. Similarly, event and group images are stored to provide visual context and enhance user engagement with event-related or group-specific content. For chat functionality, Cloud Storage manages images shared between users, whether in group chats or private conversations. By leveraging Cloud Storage, the application ensures efficient storage management, guaranteeing seamless access to images and optimizing performance, thereby enhancing the overall user experience.

2.5. External Services

2.5.1. Firebase Authentication + Google Sign-In

Firebase Authentication offers robust backend services for user authentication within our application. It provides secure and scalable methods to verify user identities, ensuring a smooth and reliable login experience. In our application, Firebase Authentication is utilized to support authentication via email and password. During the registration process, users provide their email address and password, which are securely stored. Additionally, each user is assigned a unique identifier, and their profile is further customized with their name, surname, a chosen username, and a default profile picture. Preferences, such as preferred categories, are also recorded during this registration phase. For users who prefer a quicker authentication method, Google Sign-In is available. This feature allows users to authenticate using their Google account, simplifying the login process and providing an alternative to traditional email and password authentication. By leveraging Firebase Authentication, our application ensures a secure, user-friendly authentication process while supporting multiple login options to accommodate diverse user preferences.

2.5.2. Firebase Cloud Function

In our application, Firebase Cloud Functions play a crucial role in enhancing user engagement by delivering timely push notifications. These functions are specifically designed to notify users whenever latest news updates are available in their preferred categories. To ensure users are consistently updated, the function is executed every four hours. It monitors for latest news items and triggers notifications based on the users' selected interests. This approach guarantees that users are promptly informed about the latest developments in their preferred categories, keeping them engaged with the most relevant and recent content.

2.5.3. News API

The News API is integral to our application, serving multiple functions to enhance the news experience. It offers access to a carefully curated selection of trending news stories, organized by category, allowing users to effortlessly explore popular topics and discover content tailored to their interests. Additionally, the News API is utilized by Firebase Cloud Functions to deliver push notifications to users, keeping them informed about the latest news in their preferred categories. The API also supports a search feature, allowing users to find specific news articles quickly and efficiently. This combination of features ensures a comprehensive and dynamic news experience, catering to diverse user preferences and enhancing content discovery.

2.5.4. OpenStreetMaps API + GraphHopper API

OpenStreetMaps (OSM) is a comprehensive project providing free geographic data globally. Within the application, OpenStreetMaps is utilized to offer detailed and interactive map functionality.
The integration is achieved using a specialized Flutter package designed for mapping purposes. This package supports various features, including templates, markers, and polylines, and integrates seamlessly with the OSM API. The map is centered on the event location selected by the user, with the initial view set to the user's current location if localization permissions are enabled. This setup facilitates easier event location selection.
Additionally, GraphHopper API is used to convert latitude and longitude coordinates into human-readable place names. This service ensures that users receive accurate and understandable location information, enhancing the overall mapping experience and making navigation more intuitive.

2.5.5. Firebase Cloud Messaging

Firebase Cloud Messaging (FCM) is a cross-platform messaging service that enables reliable and cost-free delivery of messages. In this application, FCM is employed to send push notifications to users who have opted in to receive them. The process involves making an HTTP POST request to the FCM API, which handles the delivery of notifications to the specified devices. This setup ensures that users receive timely and relevant updates directly on their devices, enhancing engagement and keeping them informed of important events or information.

2.6. Push Notifications

AgorApp provides registered users with the option to enable or disable push notifications for various types of updates, including new messages in group chats, private chats, and modifications to joined events. Notifications are delivered regardless of whether the app is in the foreground, background or terminated.

When the app is in the background or closed, tapping on a notification will reopen the application, ensuring users can quickly access the latest information. Notifications displayed while the app is in the foreground are managed by a dedicated Flutter package, whereas Firebase Cloud Messaging (FCM) is responsible for handling and displaying notifications that arrive when the app is in the background.

2.7. Dependencies

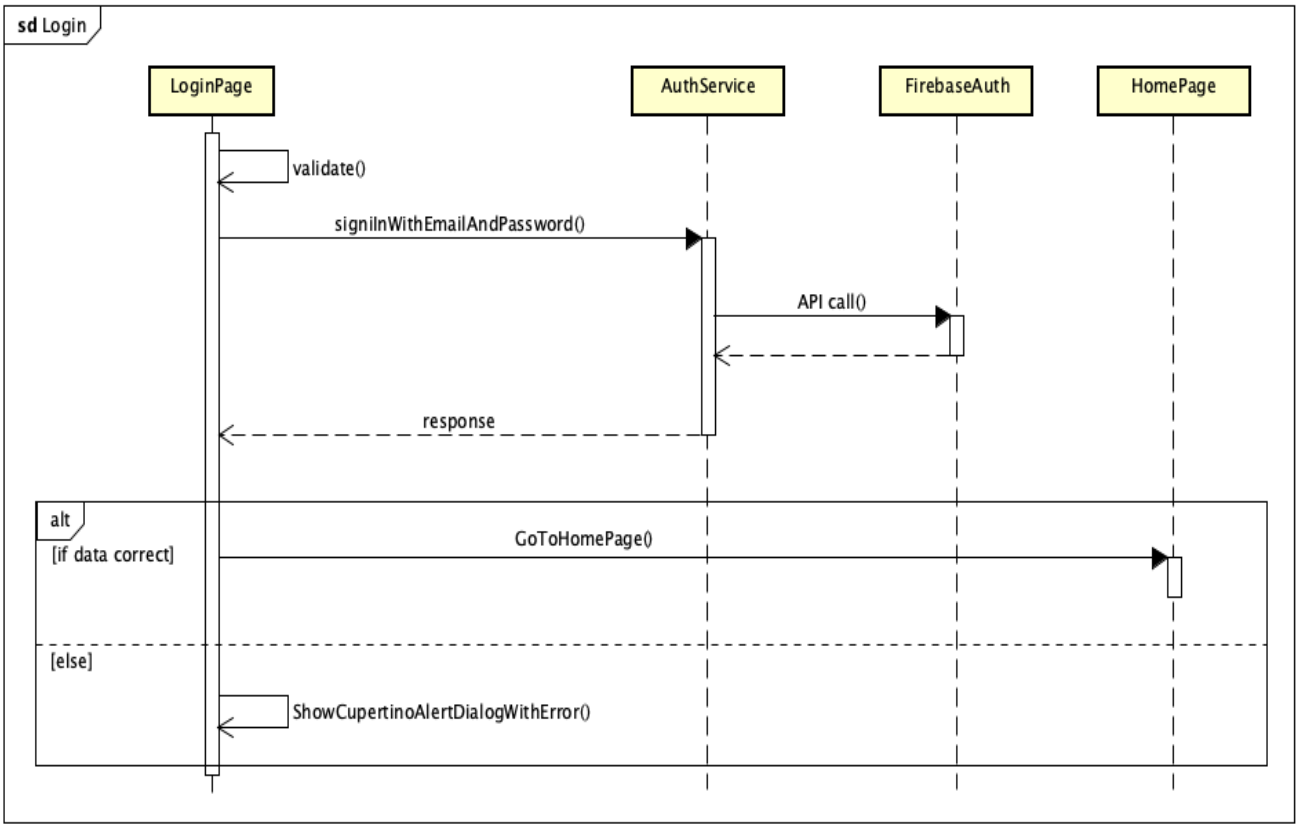
The key Flutter packages integrated into the application are listed below.

Package Name	**Description**
`go_router`	Routing package for managing navigation
`firebase_auth`	Firebase package for user authentication
`firebase_core`	Core Firebase package for integration
`google_sign_in`	Google Sign-In integration for authentication
`firebase_messaging`	Firebase package for push notifications
`firebase_storage`	Firebase package for cloud storage
`cloud_firestore`	Cloud Firestore database access
`flutter_map`	Mapping package for integrating maps
`geolocator`	Geolocation package for retrieving location data
`flutter_riverpod`	Flutter integration for Riverpod
`add_2_calendar`	Allows users to add events to their calendar
`table_calendar`	Used to provide the calendar widget
`map_launcher`	Package for launching maps with coordinates
`flutter_local_notifications`	Local notifications package
`mockito`	Mocking framework for tests
`network_image_mock`	Mocking images from network for tests
`build_runner`	Code generation tool for tests
`mocktail`	Package for creating mocks and stubs
`nock`	HTTP request mocking library

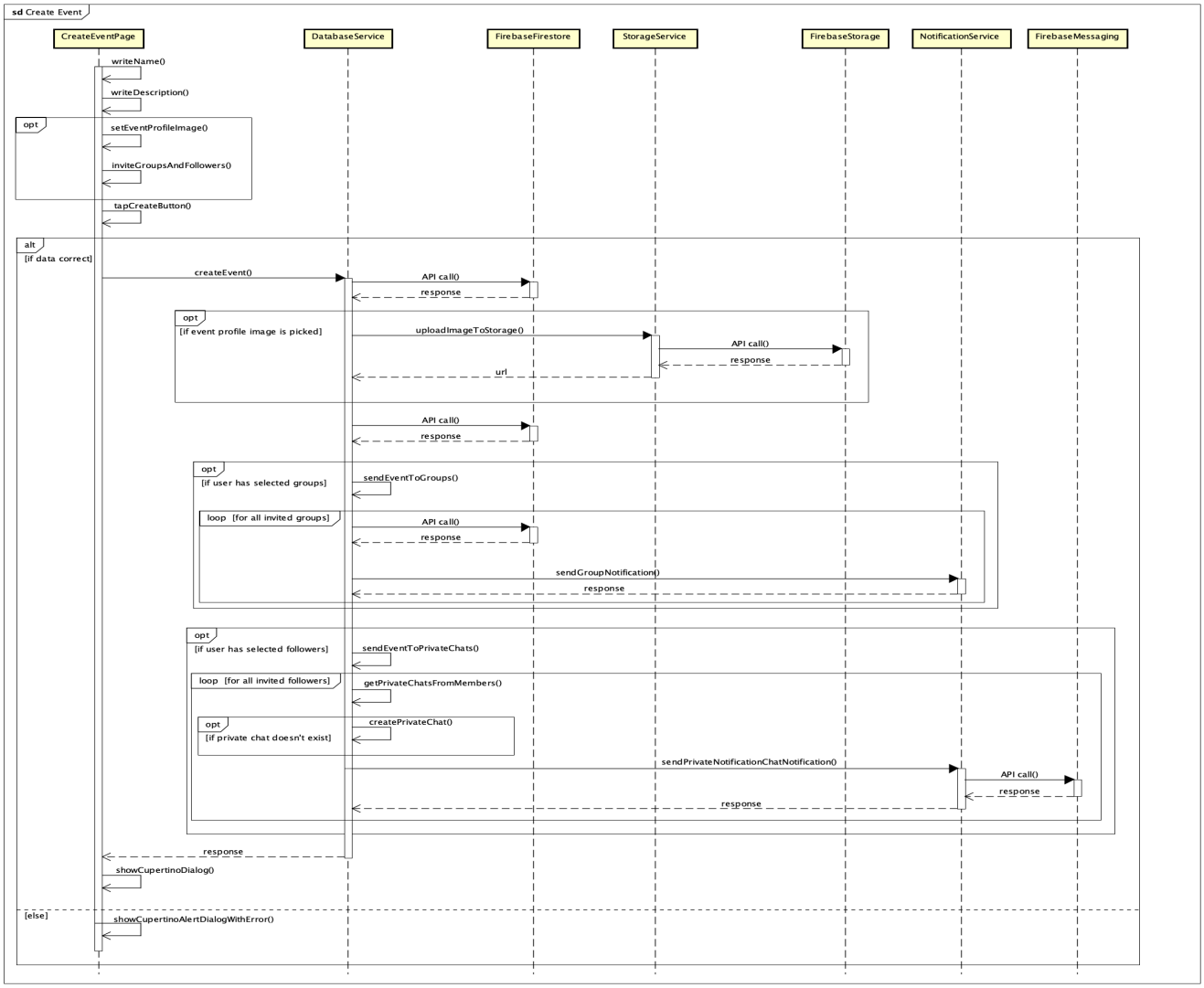
2.8. Sequence Diagrams

Sequence diagrams provide a visual representation of the interactions and collaborations between different components or objects within our system. They illustrate how these elements work together by highlighting communication flows and the specific responsibilities of each component. To offer a clear view of these interactions, a sequence diagram depicting the key actions within our application is presented. For brevity, only a selection of these interactions is shown.

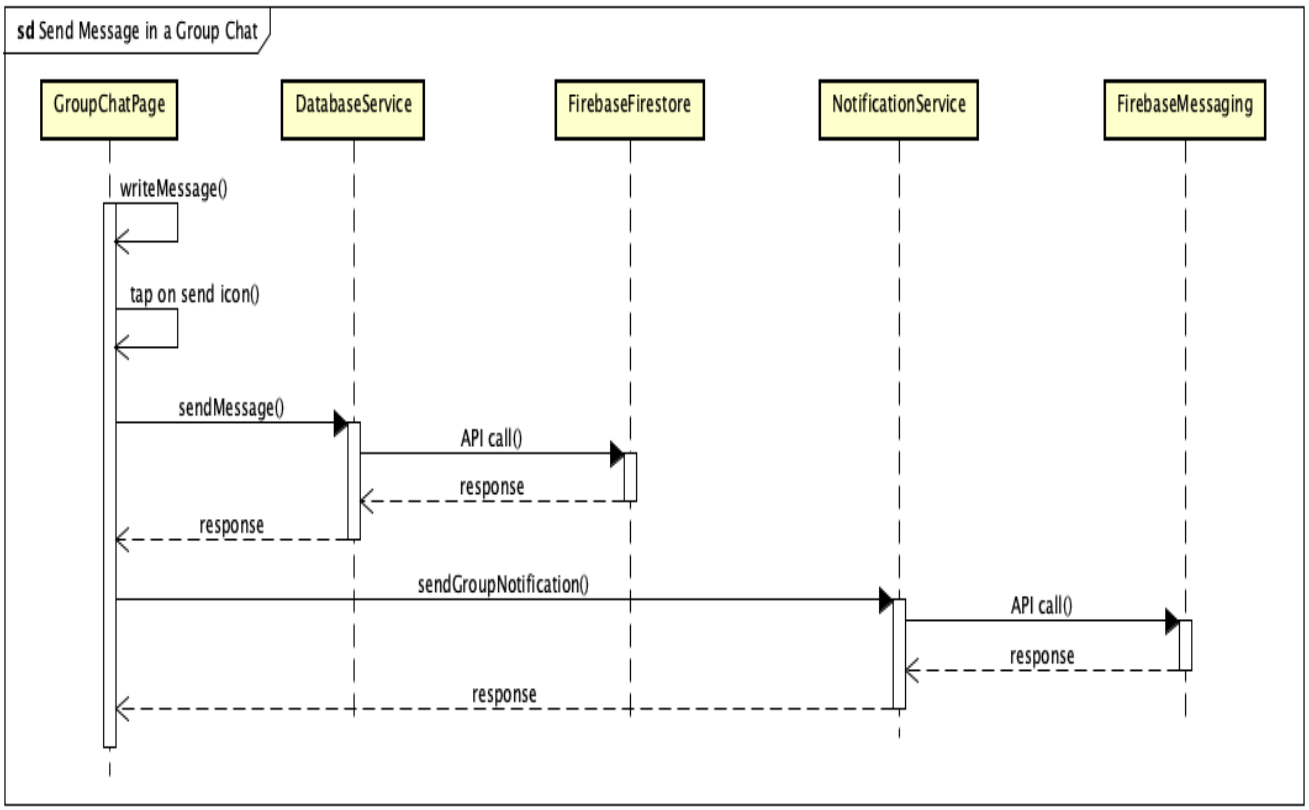
2.8.1. Login



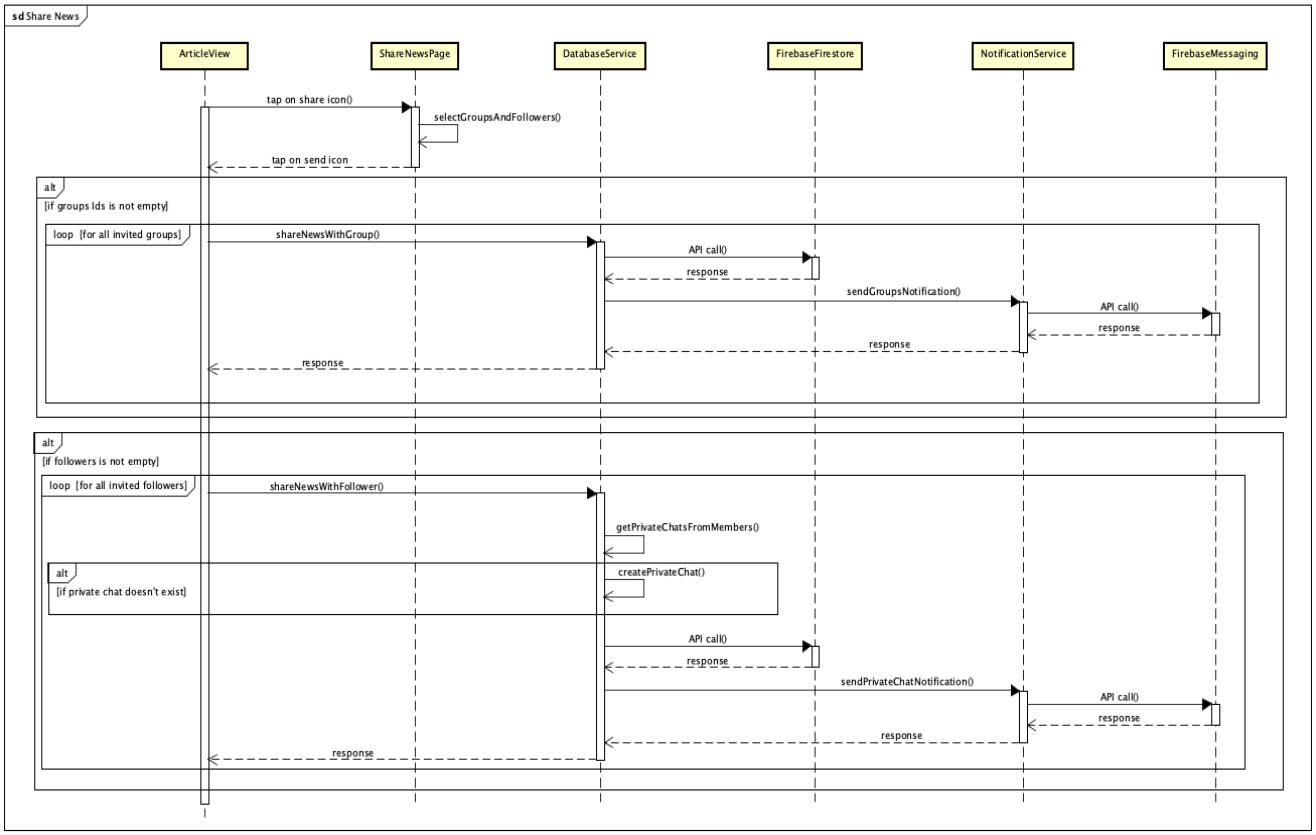
2.8.2. **Create Event**



2.8.3. **Send Message in a Group Chat**



2.8.4. **Share News**



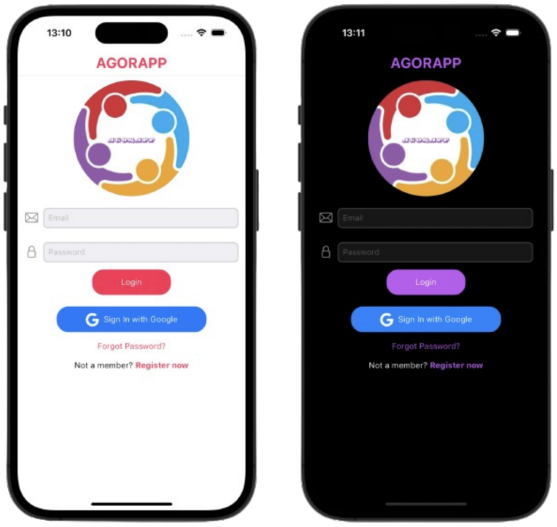
3. User Interface Design

3.1. Introduction

This section showcases the design of the main screens for the user application, some screens are omitted for brevity.
Page layouts are adjusted based on the device to optimize screen dimensions.

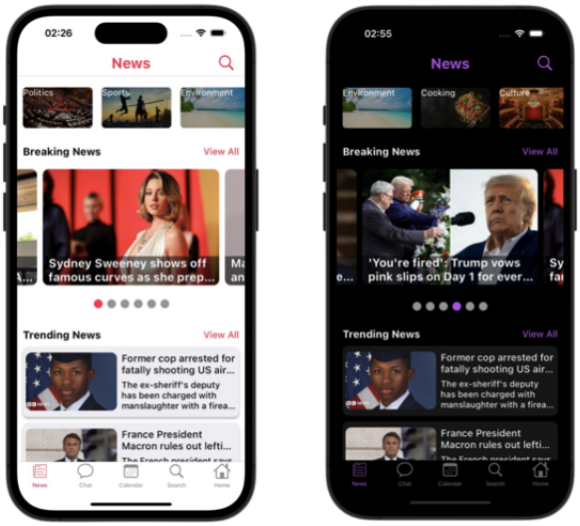
3.2. Smartphone UI

3.2.1. Login



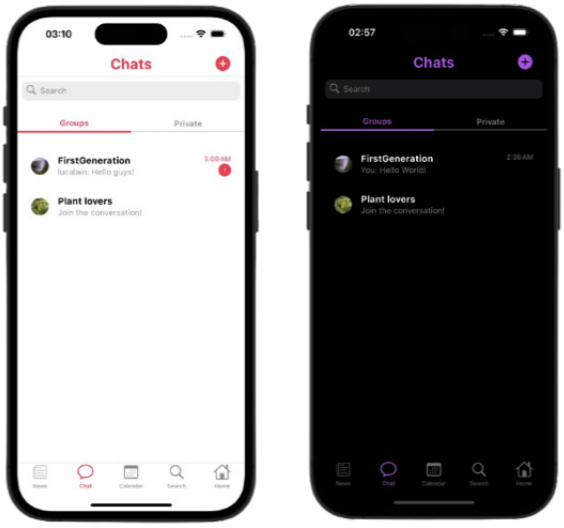
The first screen displayed when the application is opened for the first time or after a logout is the login page. A text button is used to switch from the login to the registration form and vice versa. Google Authentication is also available.

3.2.2. News Page



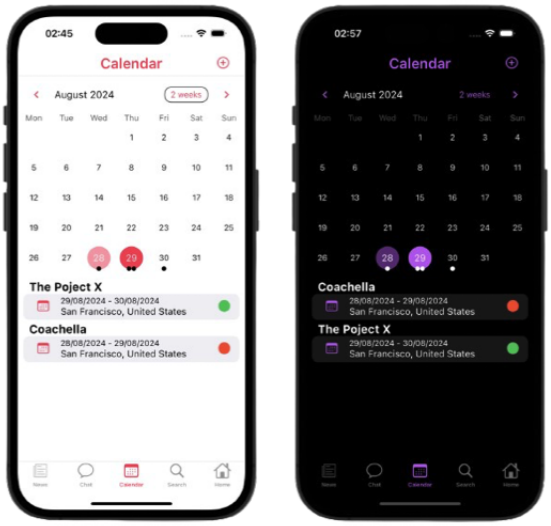
This screen is the first displayed to a logged-in user and features the most recent news. At the top, users will find the categories they have selected. Clicking on a category displays news relevant to that category. Additionally, users can search for news by clicking the search button on this page.

3.2.3. Chat Page



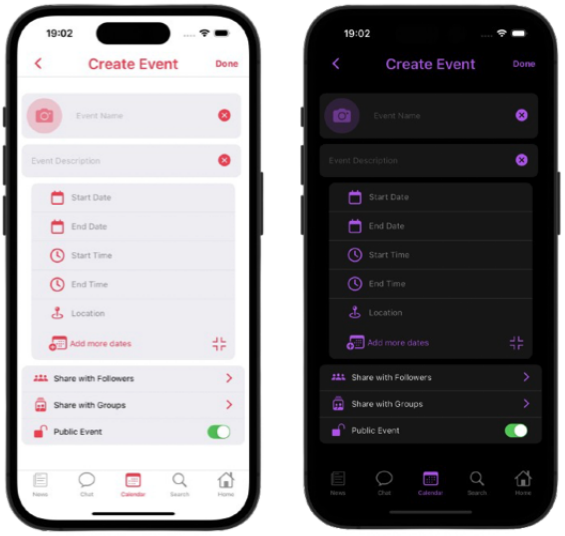
This page displays the chat interface, featuring a list of group and private chats. A button in the center allows users to switch between group chats and private chats. The page also includes a search bar for finding specific chats and an add button in the top corner that directs users to the group creation page.

3.2.4. Calendar Page



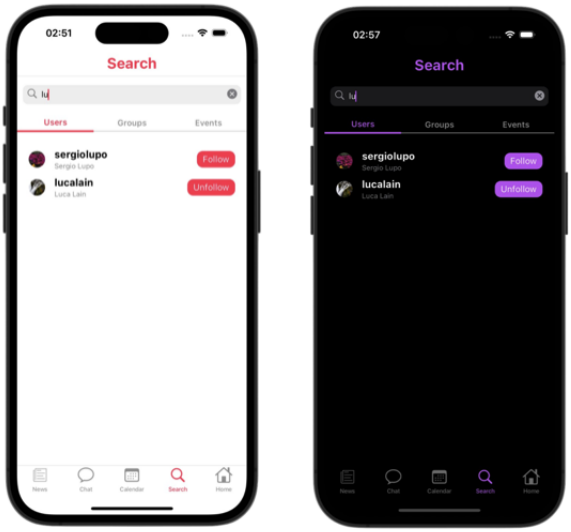
This page features a calendar that allows users to select two dates and view events created or joined between those dates. Clicking on an event tile located below the calendar displays the details for that specific event. Additionally, clicking the plus button in the top right corner takes users to the event creation page.

3.2.5. Create Event Page



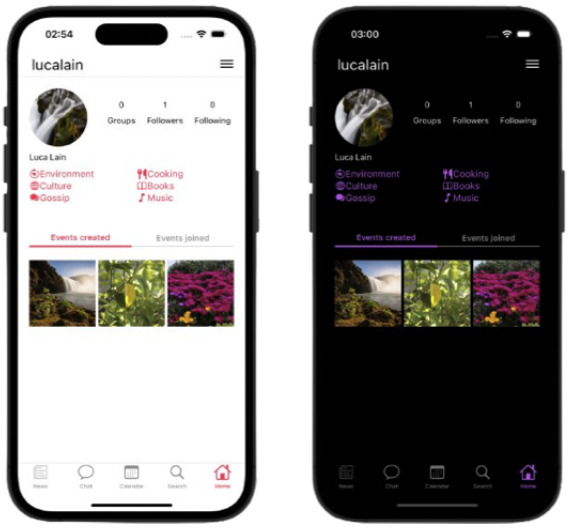
This page displays the Create Event screen, where users must provide the event name, description, at least one start date and time, one end date and time, and one location. The event profile image and invitations to followers and groups are optional.

3.2.6. Search Page



This page features a search bar that allows users to find other users, groups, and events. Users can follow or unfollow other users and send connection requests to those with private accounts. Additionally, users can join or leave groups and request access to private groups. Clicking on a user tile will take the user to the profile of the selected individual. If the user is already a member of a group, tapping on the group tile will lead to the group chat page. Selecting an event tile will navigate to the event page.

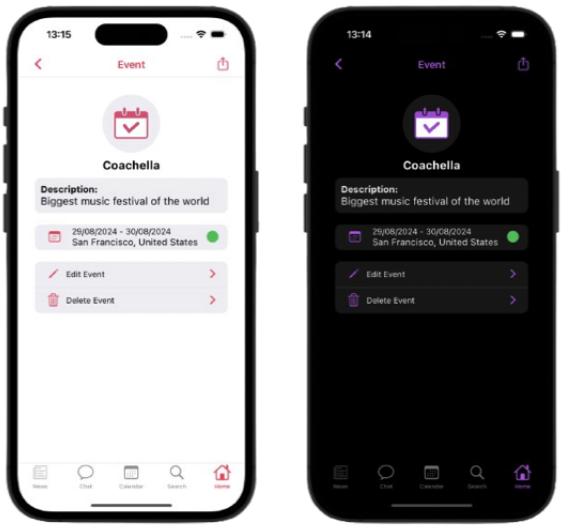
3.2.7. User Profile Page



This page displays the user profile, including the username, full name and preferred categories. It also shows the number of groups the user has joined, as well as their number of followers and followings. If the account is private and the current user does not follow the profile owner, he/she will not be able to view the user's events. In other cases, the profile displays a list of events created or joined by the user. If the displayed profile belongs to a different user, options to follow, unfollow, or request access (if the

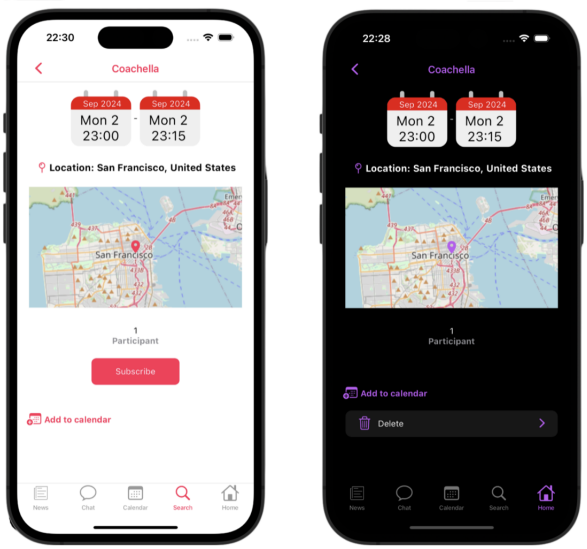
profile is private) are available. If the profile belongs to the current user, a button in the top right corner provides access to manage settings.

3.2.8. Event Page



This page displays the event details, including the name, description, list of dates and locations where the event will take place. It also shows a button in the top right corner to share the event with other groups or followers. If the logged-in user is the host, the page will shows two tiles that allow editing or deleting the event.

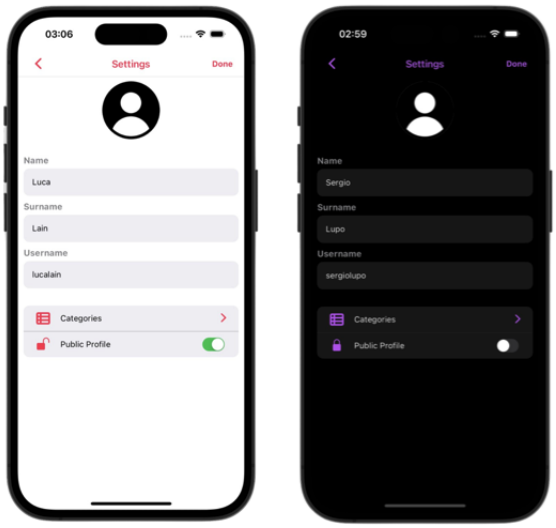
3.2.9. Detail Page



This page provides detailed information about an event, including the start and end dates and times, the location and a map showing the event venue.
If the current user is the event host, they will not see the subscribe button but will have the option to delete the event details. For other users, the page offers options

to subscribe, unsubscribe, or request access if the event is private. Additionally, users can add the event to their device's calendar directly from this page.

3.2.10. Setting Page

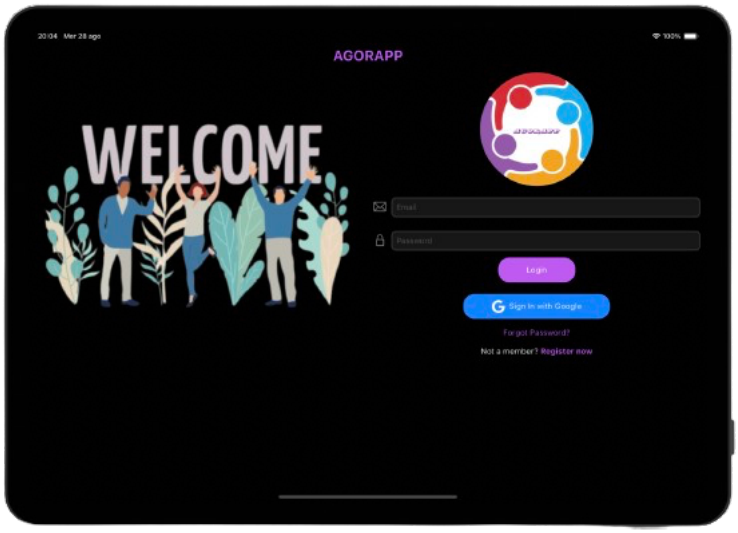


This page displays the user's image, name, surname and username in clickable tiles that allow them to be modified. It also includes two tiles to change the preferred categories and adjust the account's privacy settings.

3.3. Tablet UI

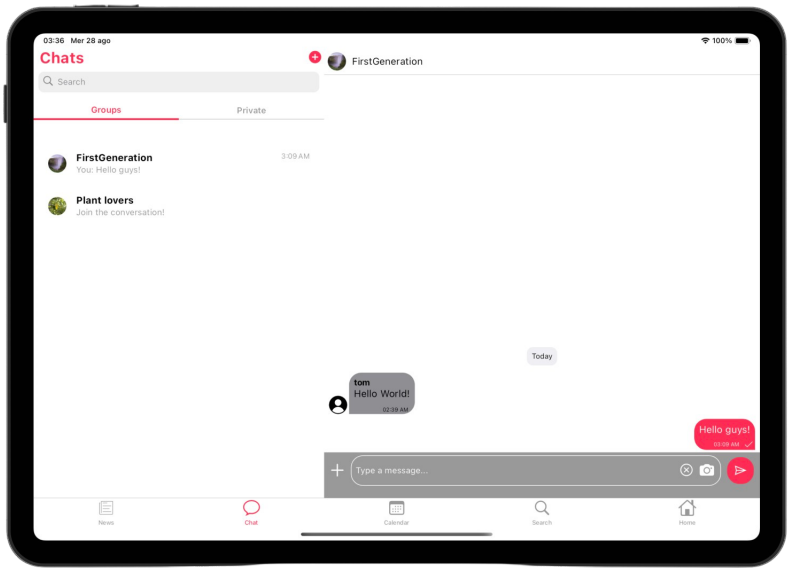
The user interface adapts to different screen sizes to enhance usability and improve the overall look and feel of the application. In this paragraph some of the most relevant page modifications w.r.t. the smartphone design is shown.

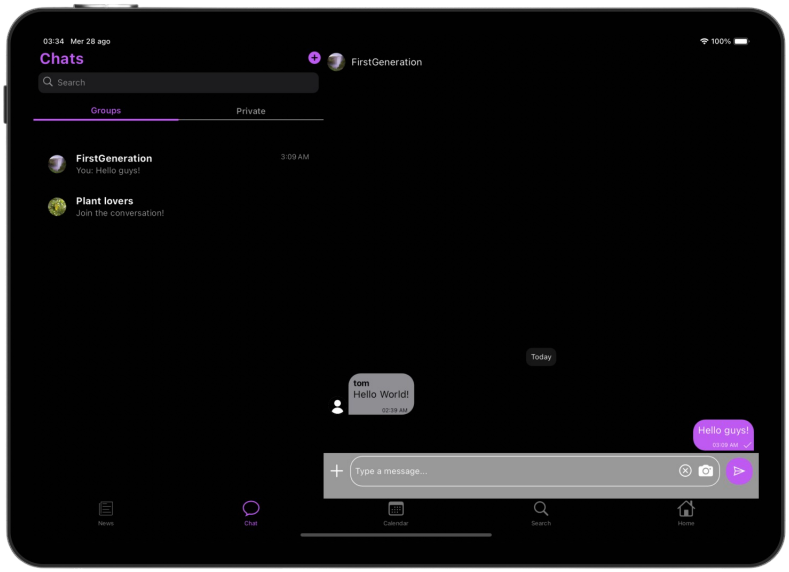
3.3.1. Login



Some of the new pages designed for larger screens do not introduce additional functionalities but enhance the overall design by making better use of the extra space. The design remains consistent with the smartphone UI to ensure a cohesive user experience.

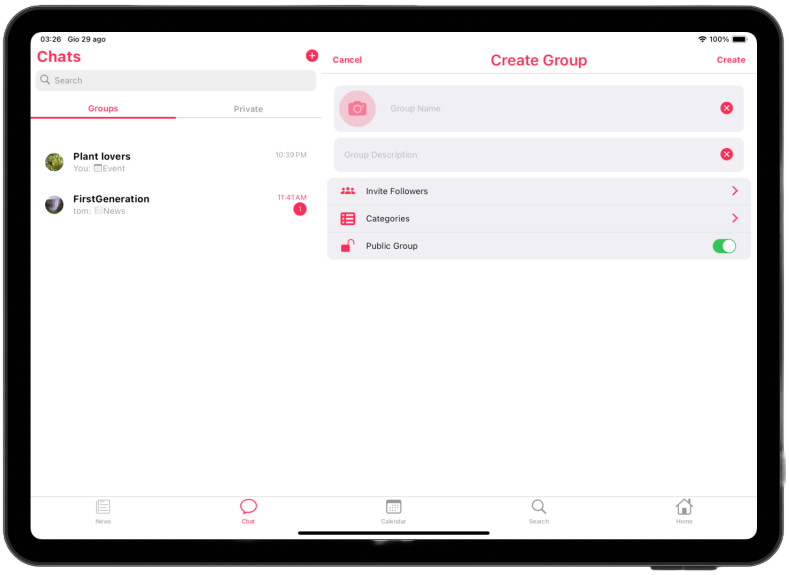
3.3.2. Chats

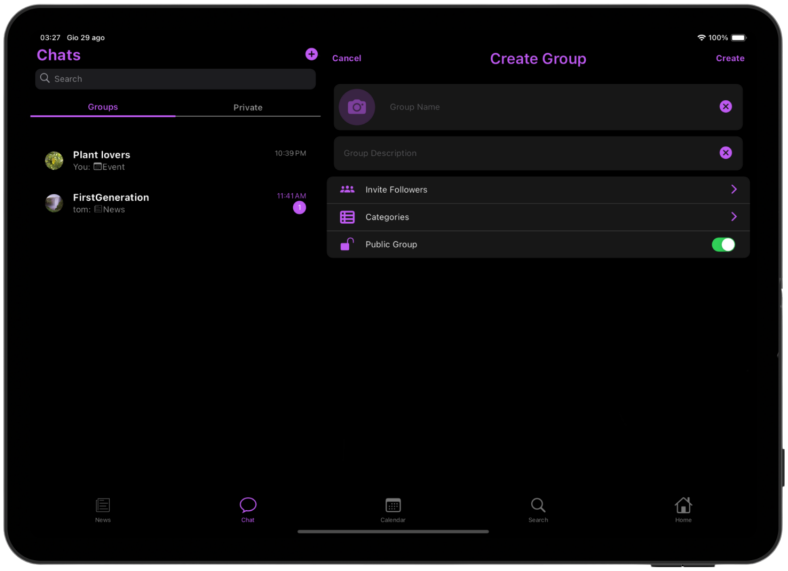




The new width size on the Chats Page allows for the implementation of a side-by-side layout, featuring the list of chats alongside the selected chat. This design choice enhances navigation between different chats.

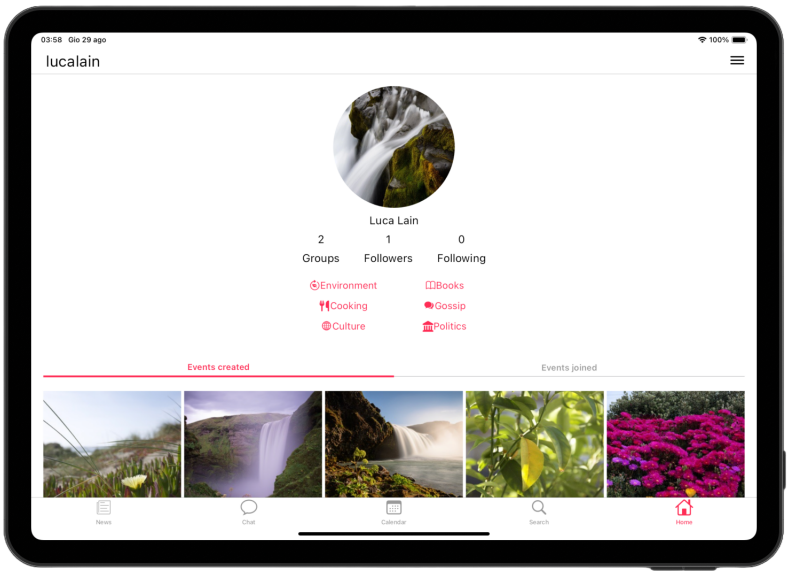
3.3.3. Create Group

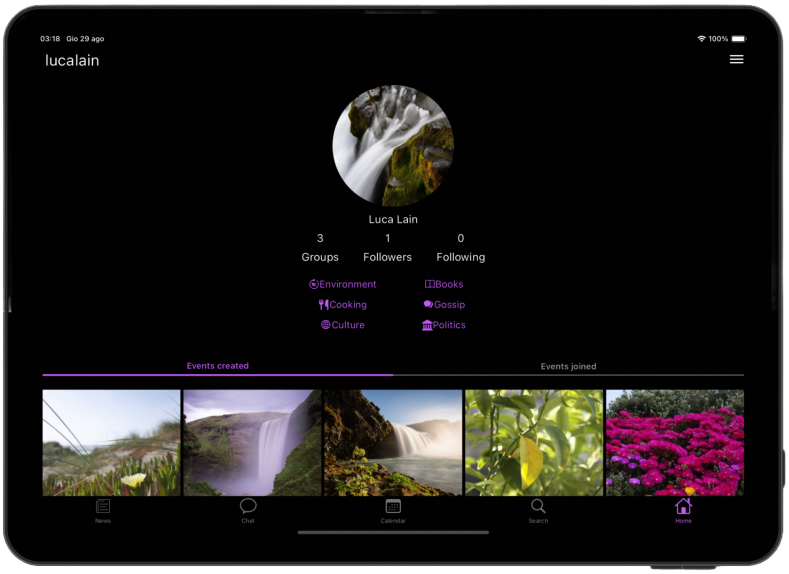




This page presents the "Create Group" interface, where users can set up a new group by specifying its name, description and optionally inviting followers. Users can also categorize the group and choose whether it should be private or public.
To optimize screen space, the "Create Group" section is positioned on the right side of the screen, while the list of chats is displayed on the left.

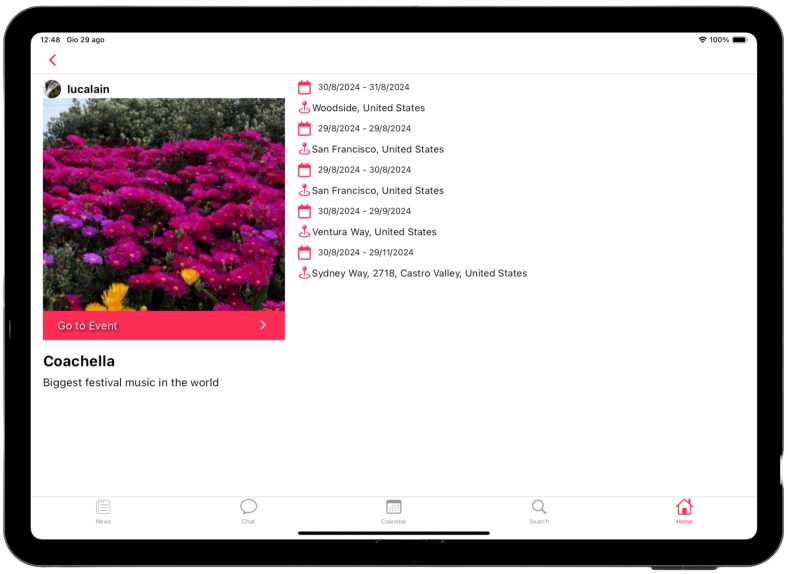
3.3.4. User Profile

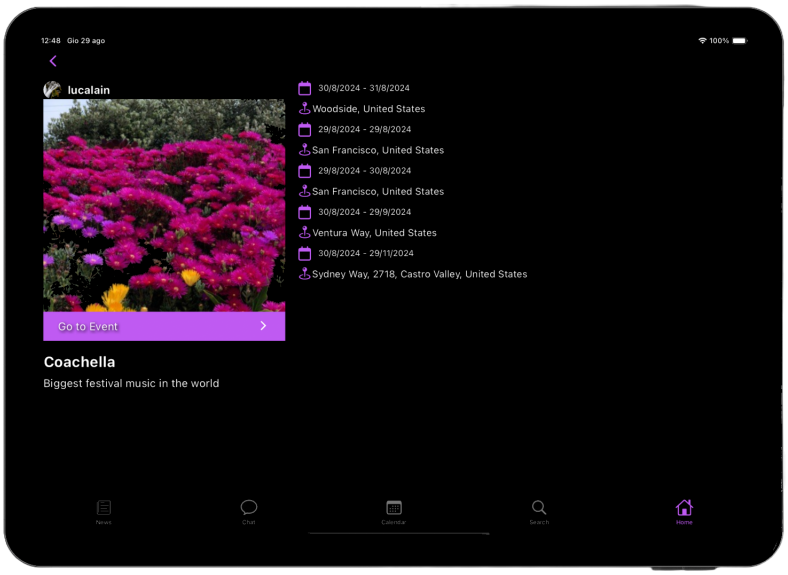




On the User Profile Page, the new width size has enabled the repositioning of the user's profile image, name and surname to the center, above the number of groups, followers and following. Additionally, the dimensions of the created and joined events have been increased and the maximum number of events per row in the grid has been expanded from 3 to 5 to better utilize the available space.

3.3.5. **Show Event Page**





On the event details page, you can view the event's name, description, dates and location. By clicking the button above the image, you can navigate to the corresponding event page for more information.

4. Requirements

The following list contains the requirements that the application should satisfy, together with a brief explanation of them.

- R1: The system shall support user login and registration.
- R2: The system shall allow registered users to edit their personal information, including their categories of interest.
- R3: The system shall allow registered users to send messages in both group chats and private chats.
- R4: The system shall send push notifications for chat messages if enabled by the user.
- R5: The system shall allow users to create a group by providing a name, description, and optional picture and categories.
- R6: The system shall allow registered users to create and delete groups.
- R7: The system shall enable users to invite followers to join a group.
- R8: The system shall allow registered users to create events within a group chat.
- R9: The system shall allow registered users to join and leave groups.
- R10: The system shall allow registered users to create and delete events.
- R11: The system shall allow registered users to view a list of joined and created events.
- R12: The system shall allow registered users to view the location of an event on a map.
- R13: The system shall allow registered users to subscribe to and unsubscribe from events.
- R14: The system shall allow registered users to follow and unfollow other registered users.
- R15: The system shall allow registered users to add and remove multiple dates and locations within the same event.
- R16: The system shall send push notifications to registered users for new news items related to their categories of interest.
- R17: The system shall allow users to log out from the application.
- R18: The system shall allow users to permanently delete their accounts.

5. Testing Campaign

5.1. Overview

The testing strategy for the application comprises three distinct types:

- Unit Testing: verifies that individual software components function according to the specific requirements defined in the model. This ensures that each unit operates correctly in isolation, determining its readiness for deployment.
- Widget Testing: focuses on the user interface and interaction of widgets within the application. It confirms that each widget appears and behaves as expected, requiring an environment that accurately simulates the widget's lifecycle and context.
- Integration Testing: ensures that the combined components of the system work together as intended. This testing confirms that the integrated system meets the specified requirements and functions correctly as a cohesive unit.
  For both Widget and Integration Testing, we utilized a mock environment to simulate real-world scenarios. This approach allowed us to replicate various conditions and interactions without relying on actual production data or live components. By using a mock environment, we were able to isolate specific functionalities, thoroughly test different modules and ensure that all components interact seamlessly within the system.

5.2. Unit Testing

Unit Testing helps verify that each unit of code performs as expected before it is integrated with other parts of the system. For this type of testing, we focused on the "models" folder, achieving 96.5% coverage with 22 tests.

5.3. Widget Testing

The goal of Widget Testing is to ensure that each widget is correctly loaded and populated by providing it with data and verifying its correct display through assertions.

As mentioned earlier, we employed a mock environment for this type of testing, where all APIs were replaced with mock versions. This approach isolates the behavior of external services from the functions being tested, allowing us to bypass the limitations of the Flutter test environment, where external services do not operate. The application's architecture facilitates this process by enabling the easy substitution of service class implementations with static data.

For Widget Testing, we focused on the "widgets" folder, achieving 90.1% coverage with 109 tests.

5.4. Integration Testing

Integration Testing builds on Widget Testing by focusing on how different widgets interact with each other, rather than on individual instances.

In practice, this involves starting from a specific page and, through various taps, text inputs and gestures, verifying that the execution flow proceeds as expected.

For Integration Testing, we focused on the "pages" folder, achieving 90% coverage with 119 tests. Additionally, this testing covered 75.3% of the "widgets" folder.

5.5. Testing Results

The following table summarizes the coverage achieved by running each type of test separately:

Test Type	**Coverage**	**Number of Tests**
`Unit Testing`	96.5%	22
`Widget Testing`	90.1%	109
`Integration Testing`	90%	119

Running all tests together yields the following results:

Test Type	**Coverage**
`Unit Tests (Model folder)`	97%
`Widget Tests (Widget folder)`	93%
`Integration Tests (Pages folder)`	90.2%

6. Future developments

AgorApp has driven the creation of several features across various aspects of mobile application design and development. While the source code from the initial release covers most of our envisioned features, some secondary tasks have been deferred due to time constraints.

Thanks to the modular architecture of the application, future enhancements can be implemented without modifying the existing source code. Here are some planned improvements:

- implement a Recommender System: develop a recommendation engine to suggest relevant news to users based on their preferences and behavior.
- custom backend: replace the current Firebase-based backend with a custom solution to enable more flexible backend operations. Although this feature was not prioritized initially, as the choice of backend does not directly impact the core implementation of the application, it was deferred to focus on front-end development.
- expand Android support: leverage Flutter's cross-platform capabilities to extend AgorApp's compatibility with Android devices. Due to hardware and budget constraints, we have not yet tested deployment on Android devices.
- add moderation bots: integrate bots into group chats to act as moderators. These bots would help manage large groups by detecting and addressing inappropriate language or behavior.

7. Final Notes

7.1. Effort Spent

Student	Backend	Mobile App	Documentation	Presentation	Total
Luca Lain	35 h	250 h	14 h	1 h	300 h
Sergio Lupo	5 h	130 h	21 h	4 h	160 h