POLITECNICO
MILANO 1863

Design and Implementation of Mobile Applications

Design Document

CommUNIty

Authors:
Erika
Giuseppe

Professor:
Prof. Luciano Baresi

A.Y. 2022-2023

1. Introduction **4**
1.1. Purpose 4
1.2. Requirements 4
1.3. Features Implemented 5
1.3.1. Login and Registration 5
1.3.2. Subscription and Unsubscription for Courses 5
1.3.3. Add\Remove a Note or a Review for a Course 5
1.3.4. View a Note or a Review for a Course 5
1.3.5. Edit a personal Note 5
1.3.6. Search and View Book information 6
1.3.7. View list of user positions 6
1.3.8. Share your position 6
1.3.9. Chatting 6
1.3.10. Change Preferences 6
2. Application architecture **7**
2.1. Overview 7
2.2. Data Model 9
2.3. Data Management 9
2.3.1. Cloud Firestore 9
2.3.2. Cloud Storage 12
2.3.3. Shared Preferences 12
2.3.4. Local storage 12
2.4. External Services 13
2.4.1. Firebase Authentication + Google Sign-In 13
2.4.2. Google Books API 13
2.4.3. Giphy API 13
2.4.4. OpenStreetMaps API + GraphHopper API 13
2.4.5. Firebase Cloud Messaging 13
2.5. Notification Implementation 14
2.6. Dependencies 14
2.7. Widget Architecture 15
2.8. Sequence Diagrams 16
2.8.1. Login 16
2.8.2. Add new note 17
2.8.3. Open user position 18
2.8.4. Send a message in a group chat 19
3. User Interface (UI) **20**
3.1. UI Design Choices 20
3.1.1. User Interaction 20
3.1.2. Main Design 20
3.2. Smartphone UI 21
3.2.1. Login and Registration 21
3.2.2. Course Page 22

3.2.3. Network Page 22
3.2.4. BookShelf Page 23
3.2.5. Group Chat Page 24
3.2.6. Settings Page 24
3.2.8. Course Reviews 25
3.2.9. Add Course Review 26
3.2.10. Course Notes 27
3.2.11. Add Course Note 27
3.2.7. Drawer 28
3.3. Tablet UI 29
3.3.1. Login Page and Minor Changes 29
3.3.2. Course Page: Note and Review 30
3.3.3. BookShelf Page 31
3.3.4. Group Chat Page 32
3.3.5. Network Page 33
3.3.6. Settings Page 34
4. Testing Campaign 35
4.1. Testing Environment 35
4.2. Unit Test 35
4.3. Widget Test 35
4.3.1. Coverage Analysis 37
4.4. Integration Test 38
4.5. User Test 38
5. Future developments 39

1. Introduction

1.1. Purpose

CommUNIty is a multi-platform application with the aim of gathering together different aspects of the study life of students.

The idea was born when we, as students, noticed the high number of services that we use to manage our university life: we had to use multiple platforms to take or share notes, organize study sessions with other students, read or write course feedbacks and opinions, choose the right book for studying and so on… but most of the services used do not fulfill these needs, so we decided to create an application born to do exactly what a student needs in a single optimized place.

This document explains the most important design choices we made and the motivations behind them.

1.2. Requirements

The following list contains the requirements that the application should satisfy.

ID	Functional Requirements
FR1	Users should be able to sign in/sign up in the app
FR2	Users should be able to logout from the app
FR3	Registered users should be able to edit their personal information
FR4	Registered users should be able to add and remove course subscription
FR5	Registered users should be able to add\remove reviews of subscripted courses
FR6	Registered users should be able to read reviews of subscripted courses
FR7	Registered users should be able to add\remove notes for subscripted courses
FR8	Registered users should be able to modify notes for subscripted courses
FR9	Registered users should be able to upload attachments into a note
FR10	Registered users should be able to download attachments from a note
FR11	Registered users should be able to check the list of provided books
FR12	Registered users should be able to read the details of a selected book
FR13	Registered users should be able to share their position
FR14	Registered users should be able to check the list of shared positions

FR15	Registered users should be able to check on the map where a user is located
FR16	Registered users receive push notifications of a new shared position, if enabled
FR17	Registered users should be able to check the list of subscripted course chats
FR18	Registered users should be able to read\send messages in a course chat
FR19	Registered users receive push notifications from chat messages, if enabled

1.3. Features Implemented

The functionalities implemented in the app, trying to follow a possible order of interaction, are:

1.3.1. Login and Registration

Users can sign in (or sign up) by using the email address or the Single-Sign-On (SSO) functionality with Google identity providers. A “forgot password” protocol is implemented to reset the password when needed.

1.3.2. Subscription and Unsubscription for Courses

Registered users can select the courses to enroll from a list that is displayed by tapping the appropriate button from the course page. The registration enables the user to access the course note or review, and write his own pages. Also, he can access the course chat and find all the people that want to study together.

1.3.3. Add\Remove a Note or a Review for a Course

Registered users, if subscribed to the relative course, can add reviews and notes related to it. The reviews are shared among all the enrolled students and are composed by a name, description and a rank from 1 to 5. Multiple reviews of the same course are possible, to differentiate them according to the topic. Notes have different privacy granularity: the user can choose if sharing his notes with the community or leaving them private. These files are composed by a name, a privacy setting, a text and a list of attachments, that can be of any type and are stored in the application cloud.

1.3.4. View a Note or a Review for a Course

Registered users, if subscribed to the relative course, can consult reviews and notes related to it. In the course page both lists are present, and ad-hoc filters are implemented for faster research of the wanted files. in note that includes attachments, a button is present to download them in the local memory for comfortable access.

1.3.5. Edit a personal Note

The author of a note can edit his script any time and in any form: From the edit page,

accessible from the note viewer by the author, he can modify the title, append other text below, remove or add attachments and modify the privacy setting of the note.

1.3.6. Search and View Book information

Registered users have access to an extensive list of book information accessible from the book page. A search button is available, where text search is enabled. A first look on book description is readable from the list card, meanwhile more information such as number of pages, author, language are accessible by tapping the book tile, which leads to the book information page.

1.3.7. View list of user positions

Registered users have access to the list of shared positions of the CommUNIty app network page. This list contains information of all the students who recently shared their position to study together with others. The list tile shows information about the course the user is studying, a small description and the time the post was published. Tapping on the tile, a map is shown indicating the position of the user, your position (if available) and shortest route by foot or by car with the relative distance in meter. A filter is present in the network page to search user positions by course.

1.3.8. Share your position

Registered users can share their position from the network page using the dedicated button. This process needed to work the user geolocalization permission, that is asked the first time the user accesses this part of the app. To share his position an alert dialog is shown, where the user can choose the course he intends to study from the list of his subscripted ones and add a little note. After the publishing procedure, the user position tile is visible on top of the position list and can be canceled with a swipe gesture from right to left.

1.3.9. Chatting

Registered users can access the chat section of the app. In this page the list of the group chats related to the courses selected by the user are shown. The list tiles show for each group the last message with the relative date and, by tapping on it, the group conversation is opened. In this page (in addition to the messages) course and users icons can be tapped to show some additional information like profile information or number of members that joined the chat. Gif button in the left part of the chat provides a database of gifs that can be used in the chat to brighten the *hard* days of students!

1.3.10. Change Preferences

Registered users can modify their profile information in the settings page of the app. In this section it is possible to edit the username, first and last name. Also, by tapping on the user profile picture, a menu appears to modify it: the user can choose from a list of amazing predefined avatars, take a picture with the camera or load it from the device memory. Removing the profile pic is possible from the same menu, which will restore the default one. Notification preferences for chats and position updates are available on this screen.
It is also possible to change the theme of the entire application (dark mode is a must for Computer Science students!)

2. Application architecture

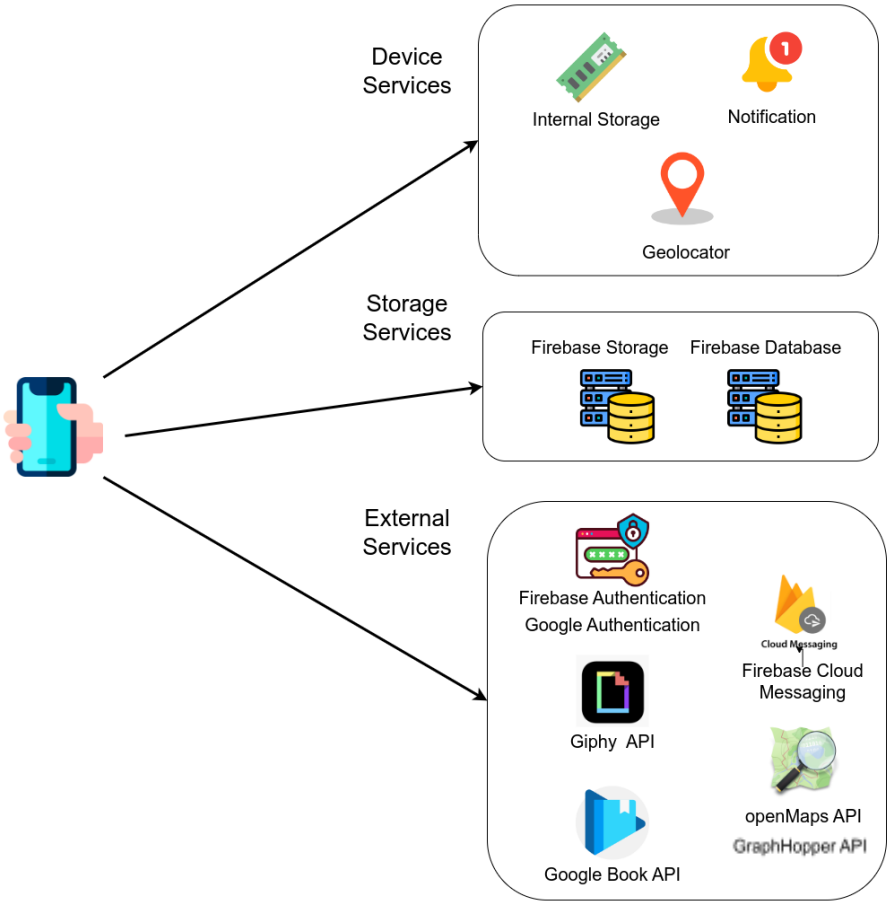
2.1. Overview

We have decided to build CommUNIty using Flutter. Flutter is an open source framework developed by Google for building natively compiled and multiplatform applications from a single codebase. Dart is the language on which Flutter is based, optimized for fast apps on any platform.

For the development of the application different services are used, which can be divided in 3 categories:

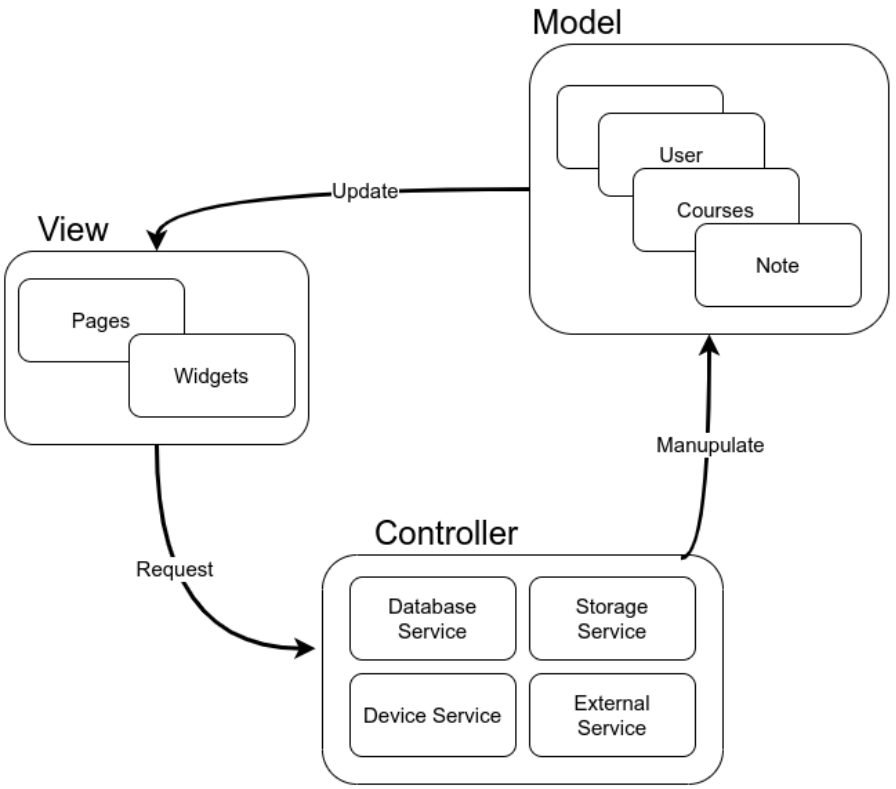
- Device Services: APIs related to the device sensors and internal functionalities.
- Storage Services: APIs related to the storage of application data.
- External Services: APIs related to functionalities and data provided by external companies.

The integration of these services is implemented by exploiting asynchronous communication protocols when possible that, with the help of future builders [1], allows the application to be as responsive and smooth as possible regardless of the performance of the external services.



[1] https://api.flutter.dev/flutter/widgets/FutureBuilder-class.html

The internal structure of the app is divided into services, classes, widgets and pages following a Model, View, Controller (MVC) pattern.



This pattern allows a clear separation between different concepts and allows to replace the external services in a transparent way with respect to the other parts of the application.

For device services, flutter libraries help us to keep the application device-independent by exploiting wrappers that permit the use of Application Programming Interfaces (API) regardless of the device used.

For the sake of clarity and to better match the pattern, the same structure is kept for the source code file structure.

2.2. Data Model

Data shown in the application pages has been structured using classes which represent the *Model* part of the MVC pattern previously illustrated.
By following the app requirements (1.2) and designing the view of each page, we ended up defining the following objects:

- *UserModel* and *PositionModel*, which include info of registered users and details of their shared position.
- *CourseModel*, which includes data of a specific course.
- *NoteModel*, *ReviewModel* and *FileModel*, which define the structure of course note/review and the related attachments.
- *BookModel*, which defines the structure of a book.
- *GroupChatModel* and *MessageModel*, which define attributes for group chats and messages details.

Data is fetched from different sources and it is parsed using some utility functions in order to obtain the desired structure.
More details about data sources and data attributes will be presented in the next two sections, together with other useful data for which it was not necessary to define a model.

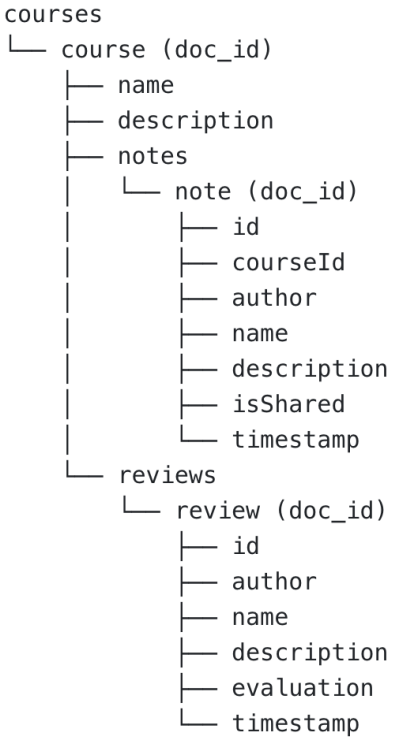
2.3. Data Management

2.3.1. Cloud Firestore

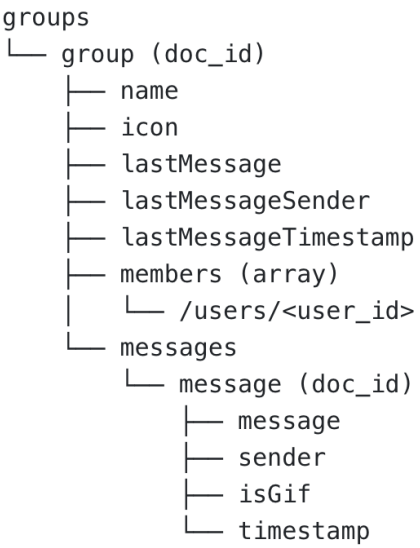
Cloud Firestore is a document-based database that stores all the relevant data displayed in the application. In particular, the internal structure of the collections is the following:

```
users
└── user (doc_id)
    ├── email
    ├── username
    ├── imageUrl
    ├── first_name
    ├── last_name
    ├── position
    │   ├── coordinates
    │   │   ├── latitude
    │   │   └── longitude
    │   ├── course
    │   ├── description
    │   └── timestamp
    ├── courses
    │   └── list
    │       ├── /courses/<course_id>
    │       └── ...
    └── groups
        └── list
            ├── /groups/<group_id>
            └── ...
```

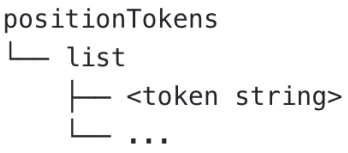
users: this collection stores information about registered users, i.e. email, username and profile picture (file is stored in firebase storage, while its url is stored here). When the current position is shared through the dedicated feature, a map field is added to the document, including geographical coordinates and other details. First name and last name fields can be added in the settings page. *Courses* and *groups* are two sub-collections that store the list of course references (added when a user selects a course) and group references, respectively (a reference is a document's path inside Firestore).
The main advantage of sub-collections is the 'shallow query' mechanism according to which, when a document is retrieved from the database, all data contained in a sub-collection is not automatically retrieved, but further requests have to be performed in order to get and use data. This allowed us to avoid downloading useless loads of data when not necessary, improving the overall application performances.



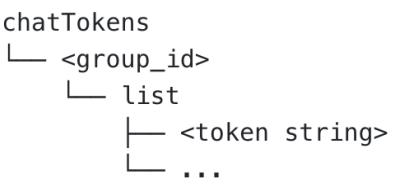
courses: this collection stores information about university courses, i.e. name and description. Two sub-collections include the list of reviews and notes related to the course. *Note* structure includes, among all the attributes, a name, an author, a description field and a boolean field that specifies if the note is private or shared with all CommUNIty users. Notes can also include attachments that are collected in the storage service (it will be discussed later). *Review* structure includes an evaluation parameter and a description.



groups: this collection stores information about group chats, i.e. name, icon, an array of members (each of which is a reference to a user in the database), info about the last message sent (if present) and a sub-collection of messages where each message includes the sender (user reference) and a boolean field indicating if it is of type gif.



positionTokens: this collection stores the devices' tokens of users that have enabled the position push notifications, in order to get one of them when a user shares his/her current position.



chatTokens: this collection stores the devices' tokens of users that have enabled the chat push notifications, in order to get one of them when a user sends a new message in a group chat. Lists of tokens are divided by group id.

2.3.2. Cloud Storage

Cloud Storage is built for apps that need to store and serve user-generated content, such as photos or videos.
In this scenario, it has been used mainly to store users' profile pictures, as well as attachments uploaded together with a course note. The structure of the main folders is the following:

```
notedoc
└── <course_id>
    └── <note_id>
        ├── <attachment_name>
        └── ...
```

```
profilepic
├── <user_id>_profilepic.png
└── ...
```

Attachments are divided by course and note, both identified by their ids generated in Firestore. Each user has only one profile picture associated with him/her.
Two other folders, *avatar* and *groupicon,* store avatar images (which can be selected as profile picture) and group chat icons, respectively.

2.3.3. Shared Preferences

Shared preferences is a plugin that provides persistent storage for simple data represented in key-value format in device's memory.
In this application it has been useful for storing users' preferences:

- choice of theme mode between light and dark (key: *darkMode*)
- activation/deactivation of chat and position notifications (keys: *notificationChat* and *notificationPosition*)

When a user updates one of these options in the dedicated page, a boolean value is assigned to the specific key and stored in memory so that, when the user closes and reopens the application, the chosen settings are always restored.

2.3.4. Local storage

One of the main features of CommUNIty app is the creation/view of course notes where, on one hand, users can upload personal contents from their device's local storage, for example pdf/csv files or images containing relevant course arguments; on the other hand, public notes can be seen by all registered users who can download the related attachments and go through them on their devices whenever they need. The uploading/downloading of contents is possible thanks to some helpful flutter packages that allow the application to communicate with the supporting device.

2.4. External Services

2.4.1. Firebase Authentication + Google Sign-In

Firebase Authentication provides backend services to authenticate users to our app. It has been used to support authentication using email and password which are then stored when a user completes the registration phase, together with a unique generated id. A custom username and a default profile picture are also generated and stored during this phase.
Google Sign-In, instead, supports authentication using a Google account.

2.4.2. Google Books API

Google Books API is a public service used to show books information that can be useful for users who are often searching for specific topics of study during their university career.
Data is retrieved through an HTTP GET call towards an API endpoint; the following information is fetched: book title and subtitle, description, image, language, number of pages, publication date and authors.

2.4.3. Giphy API

Giphy is a public API that provides free access to the largest library of GIFs and stickers in the world.
It has been integrated in our application by using a specific flutter package and an API key provided after the registration to the developers' platform, and it is used to send funny gif messages in a group chat. Gif messages are stored in the database as urls and they are shown as network contents in the chat page.

2.4.4. OpenStreetMaps API + GraphHopper API

OpenStreetMaps is a project that creates and distributes free geographic data for the world.
Flutter provides a specific package for building maps (with templates, markers, polylines, etc..) which can be easily integrated with the OSM API.
In this scenario, the map is built around the saved position of a selected user. The current user of the app can also see his/her position on the map if the localization permissions are enabled. It is worth noting that no real-time location tracking functionality is provided for the current user since the map wasn't designed to be used as navigator, but just to show approximately the distance from another user.

GraphHopper, instead, is an API used to compute the route between two geographic points (the current user position and the other user position), by following a path on foot or by car. The distance in meters is retrieved and shown too.

2.4.5. Firebase Cloud Messaging

Firebase Cloud Messaging is a cross-platform messaging solution that allows reliably to send messages at no cost. In this application, it is the service used to send push notifications to users that have enabled them. This happens thanks to an HTTP POST call towards the Firebase Cloud Messaging (FCM) API that will take care of delivering the built message to the devices specified in the body of the request (fields are title and body of message and list of devices' tokens).

2.5. Notification Implementation

CommUNIty allows registered users to enable/disable push notifications for new incoming messages in a group chat, as well as push notifications when a user shares his/her position through the dedicated feature.
When a user tries to switch on notifications in the settings page, a pop-up asking for permissions is displayed: if the user agrees, then he/she will receive the aforementioned messages, otherwise this feature will not be activated and it will be required to update the device settings in order to enable notifications at a later time.

Notifications arrive both when the app is in foreground and when it is in background or terminated. In the latter case, tapping on the notification will reopen the application.
The visualization of foreground notifications is handled by a specific flutter package, while for messages arriving when the app is in background, FCM takes care of displaying them (2.4.5).

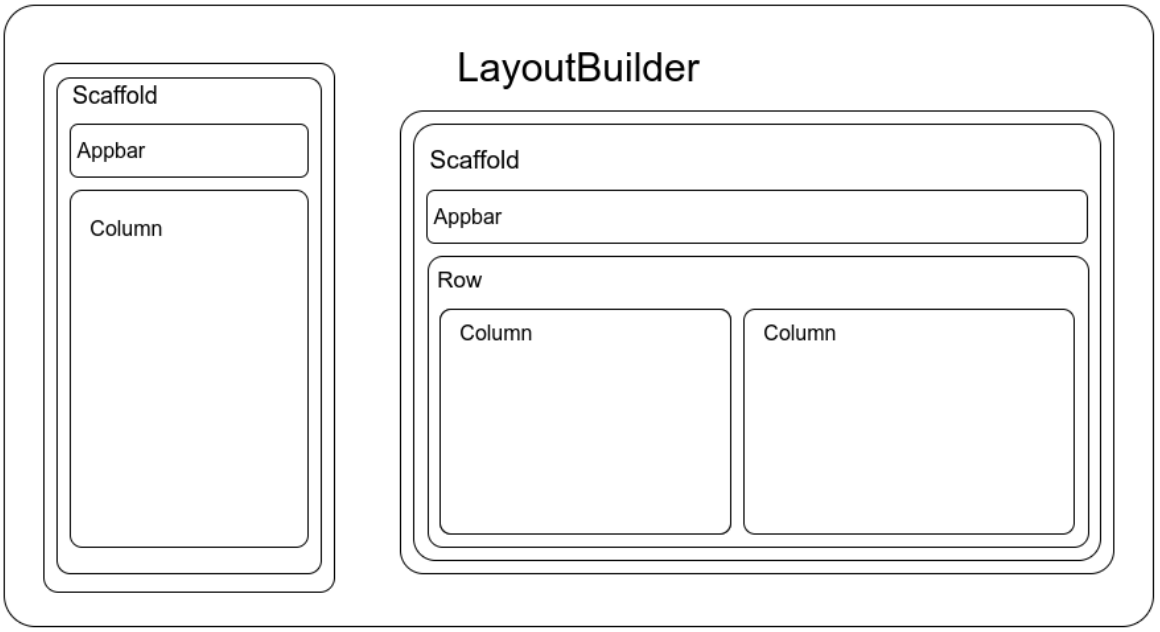
2.6. Dependencies

The most significant flutter packages included in the application are listed below.

firebase_core	Used to initialize Firebase services
firebase_auth	Used for authentication with email and password
google_sign_in	Used for authentication with Google
cloud_firestore	Necessary to use the Cloud Firestore API
firebase_storage	Necessary to use the Firebase Cloud Storage API
shared_preferences	Used to provide persistent storage for simple data
go_router	Necessary to use the Router API
firebase_messaging	Necessary to use the Firebase Cloud Messaging API
flutter_local_notifications	Used for displaying local notifications
geolocator geocoding	Used to access location services Used to compute the address from coordinates
flutter_map	Used for the configuration of a map
image_picker file_picker giphy_picker	Used to load images when updating profile pic Used to load files when writing a new note Used to load gifs to send in a group chat
flutter_file_downloader	Used to download attachments from notes
flutter_test integration_test	Used to perform widget tests Used to perform integration tests

2.7. Widget Architecture

In this section the general widget organization of the pages is shown. The use of this widget framework is used to better generalize on different kinds of screen size and help with code structure for better reuse of it.



As can be seen in the User Interface Paragraph (3.), most of the tablet screens are composed of different smartphone pages positioned side by side. To implement this design we have chosen the above widget framework: a scaffold with an app bar that remains constant in most of the pages, and a column that includes all the main widgets used in the page; this column is often wrapped within a general widget class to be reused in the tablet implementation.

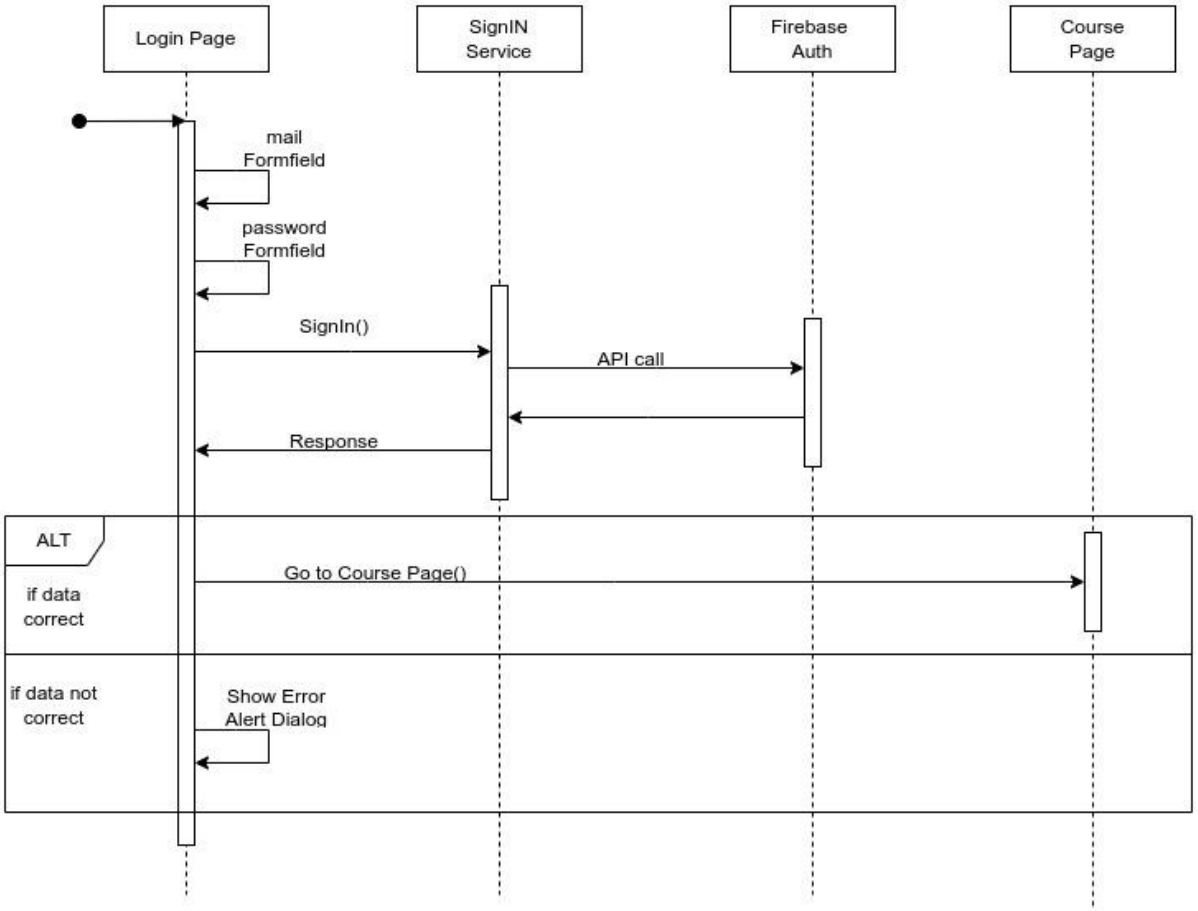
Both smartphone and tablet pages are then wrapped in a LayoutBuilder widget that allows us to select the right design dynamically according to the screen size and in a transparent way with respect to the design of the page.

The choice of wrapping the whole page and not only the different widgets was made in order to achieve an optimal trade-off between reuse of the code and design flexibility.

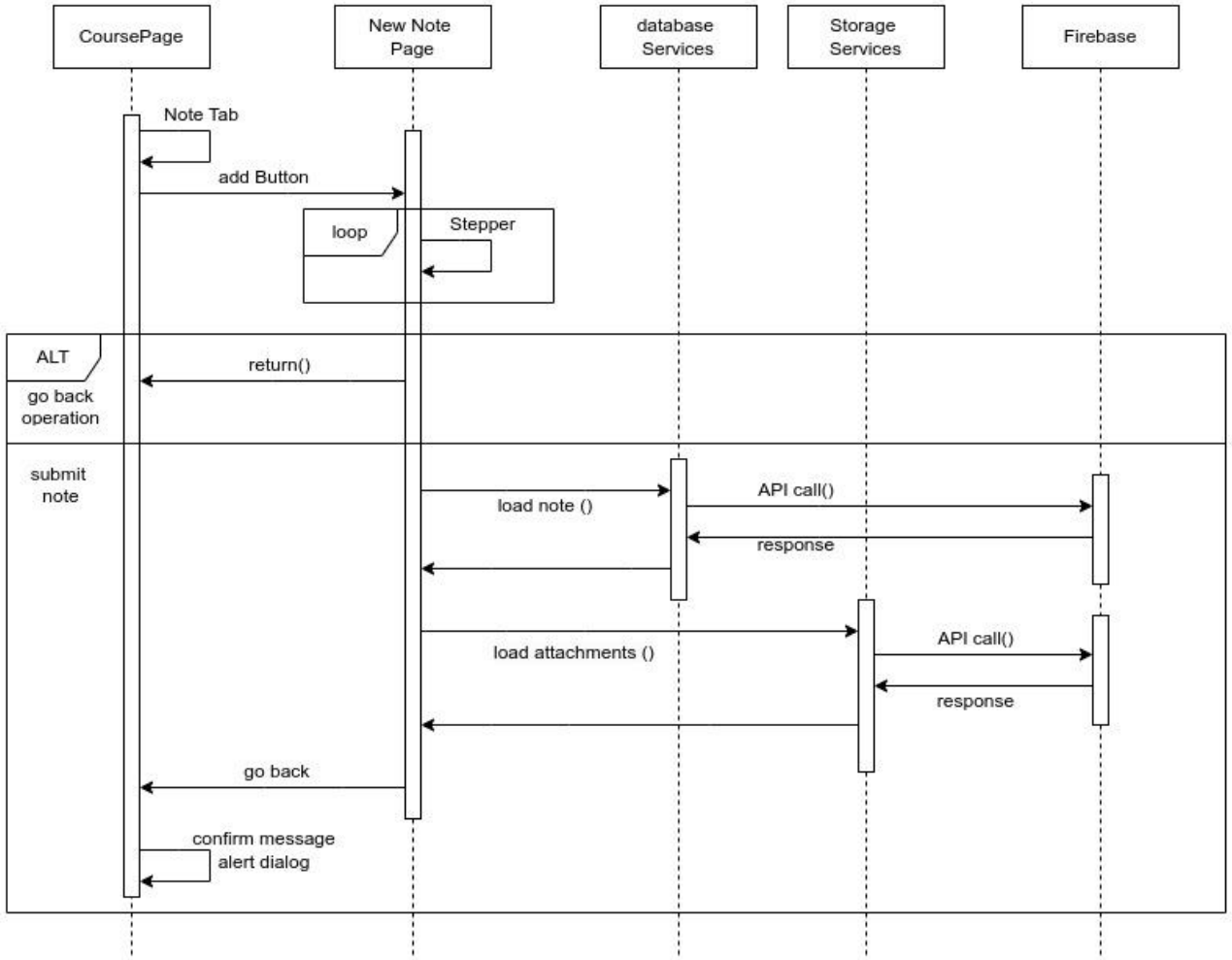
2.8. Sequence Diagrams

Sequence diagrams offer a visual representation of how different components or objects interact and cooperate within our system; these diagrams provide a clear understanding of the system's behavior, communication patterns, and the responsibilities of each element. For these reasons, the sequence diagram of the most relevant actions of our application is shown.

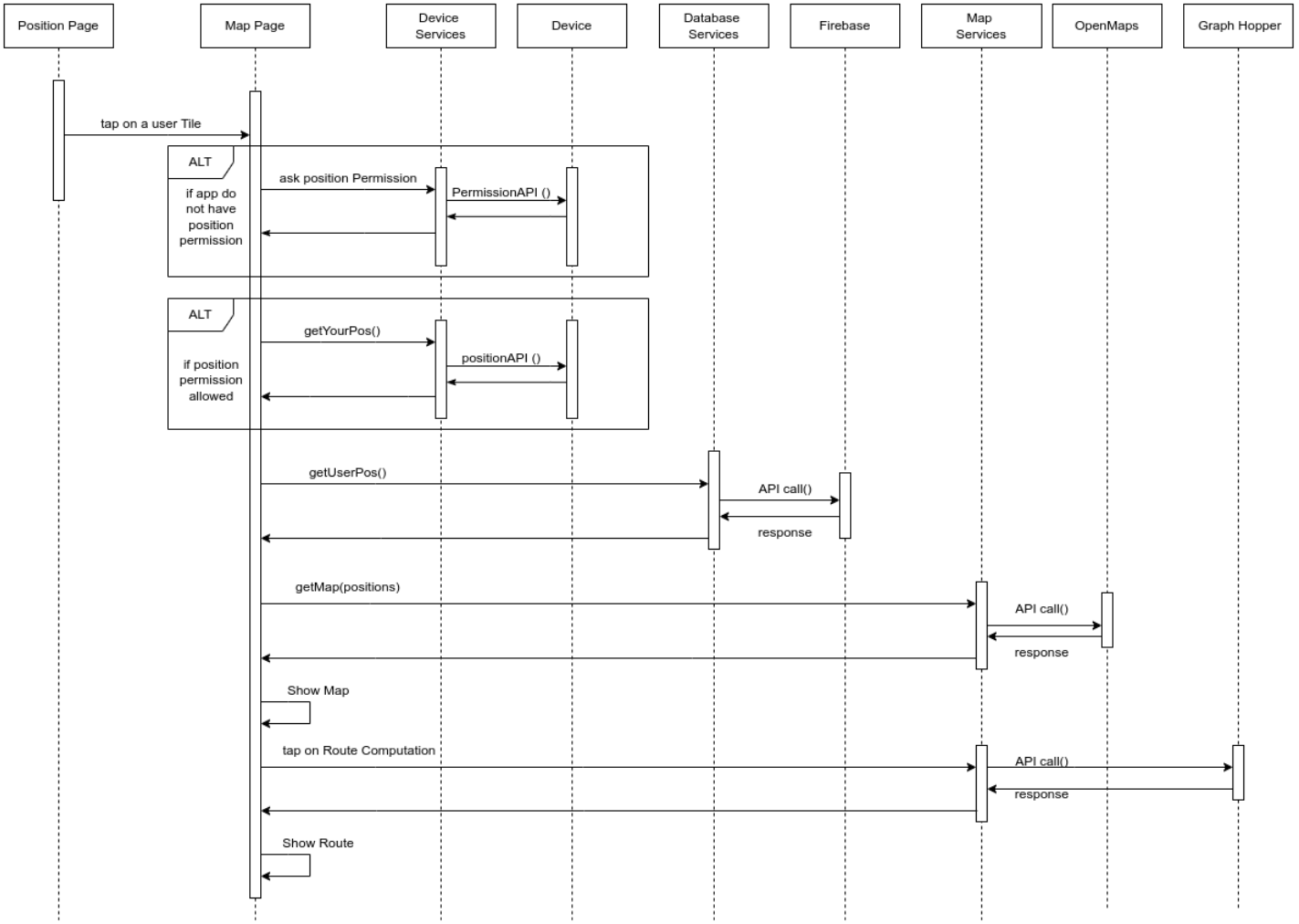
2.8.1. Login



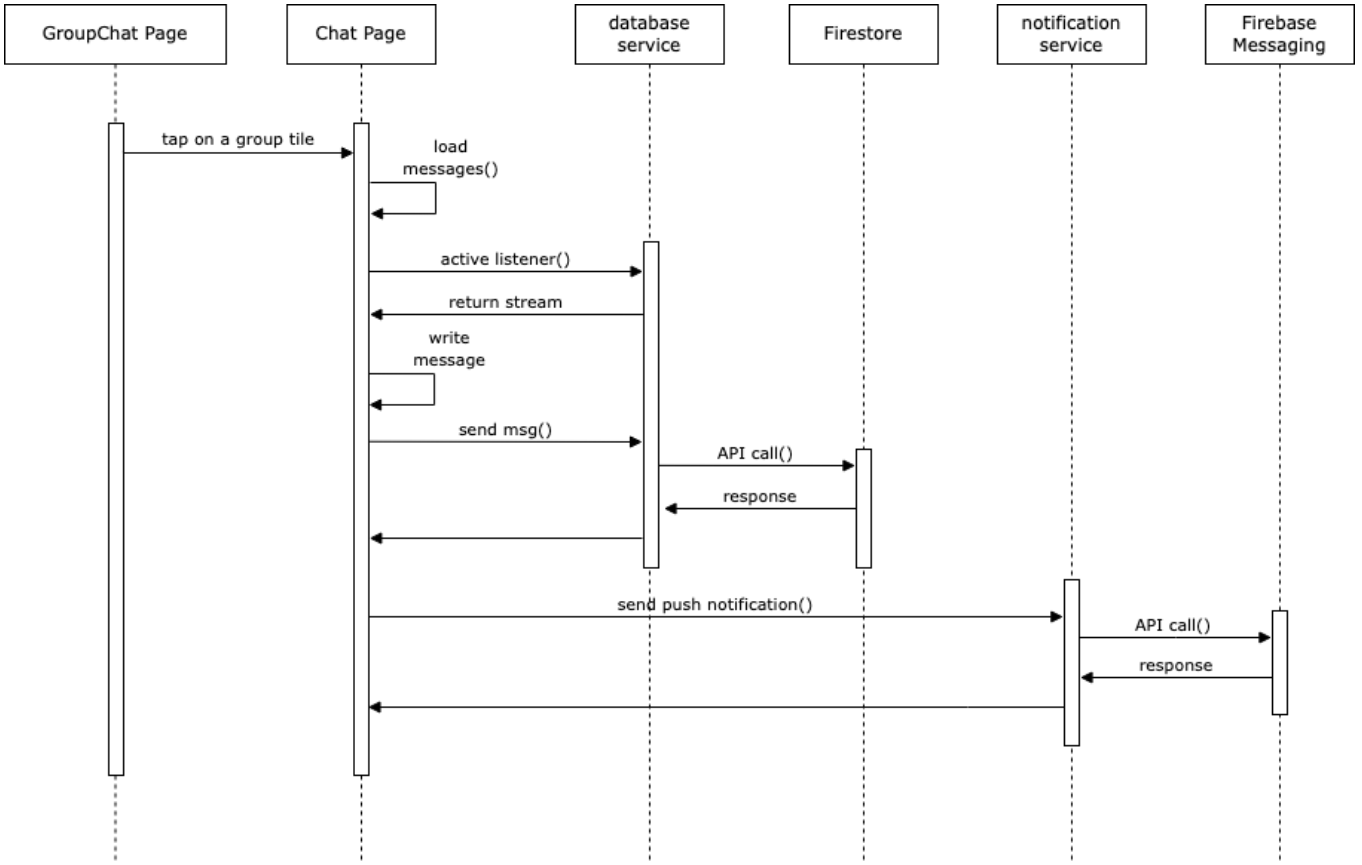
2.8.2. Add new note



2.8.3. Open user position



2.8.4. Send a message in a group chat



3. User Interface (UI)

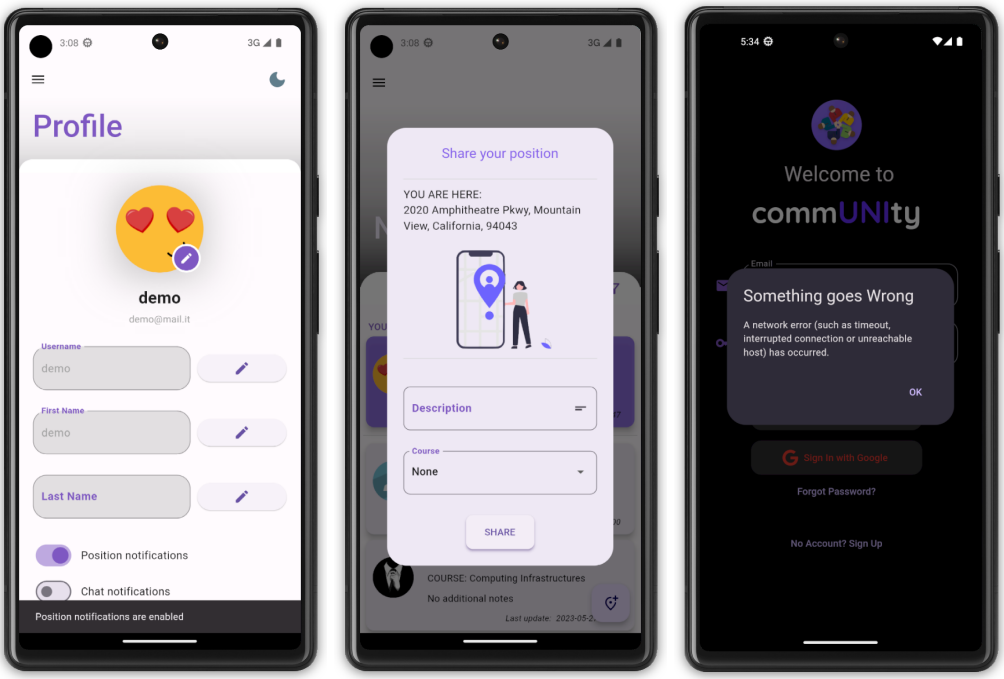
This section is used to show the design of the main screens of the user application. Page layouts change according to the device used in order to exploit the dimensions of the screen. Most of the screens are shown in both theme modes, light and dark; others, instead, are shown in only one of them for the sake of brevity.

3.1. UI Design Choices

3.1.1. User Interaction

We have decided to use alert dialogs for quick interactions with the user (i.e. simple form or showing short information); this widget allows us to simplify the navigation of the app and reduces the steps needed for an action. For secondary information, snackbars are used not to interrupt the flow of actions of the user.

Complex interactions (e.g. add a note) are managed using a stepper, which splits the form in different parts that can be easily managed with a touchscreen interface.



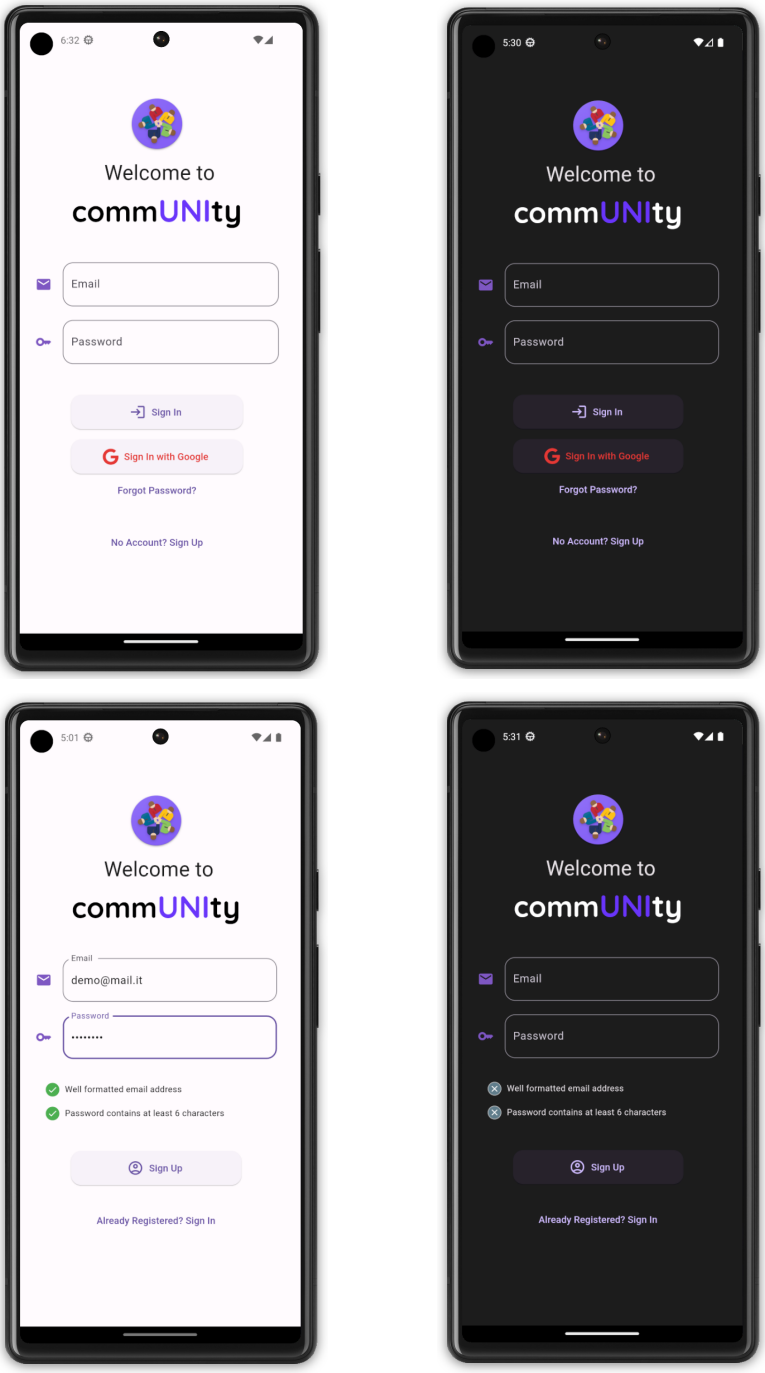
3.1.2. Main Design

For the design of the main pages (accessible from the drawer) we decided to take inspiration from the "*Reachability*" [1] function designed by Apple, making only the lower part of the screen interactive to let the user perform simple actions comfortably (e.g. choose an element from a list).

[1] https://support.apple.com/guide/iphone/reachability-iph145eba8e9/ios

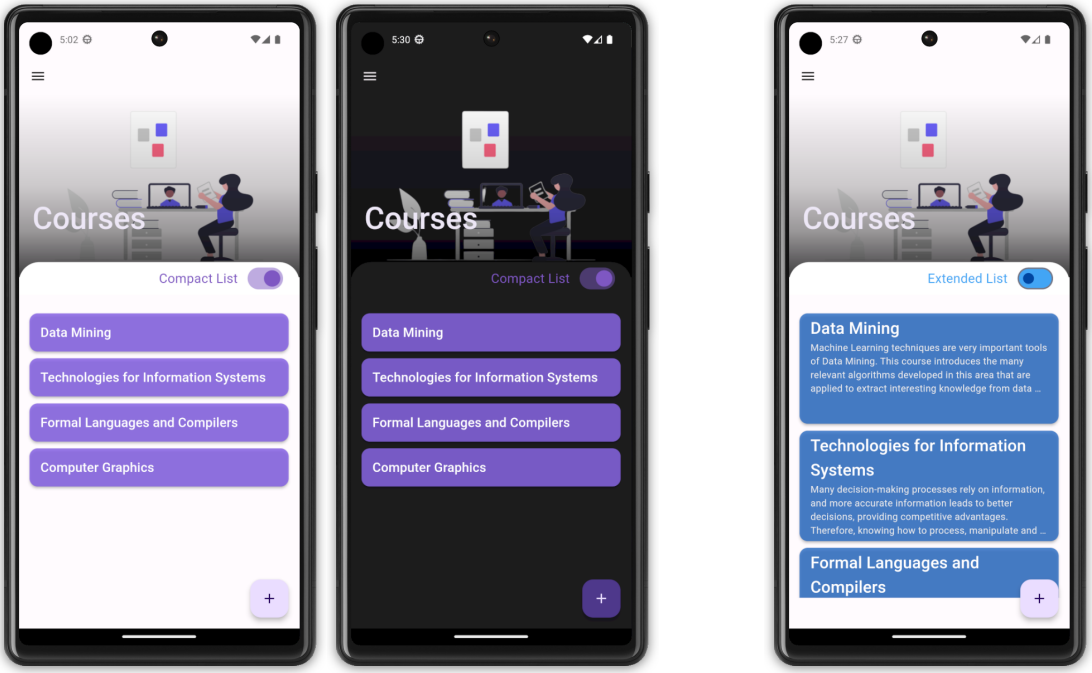
3.2. Smartphone UI

3.2.1. Login and Registration



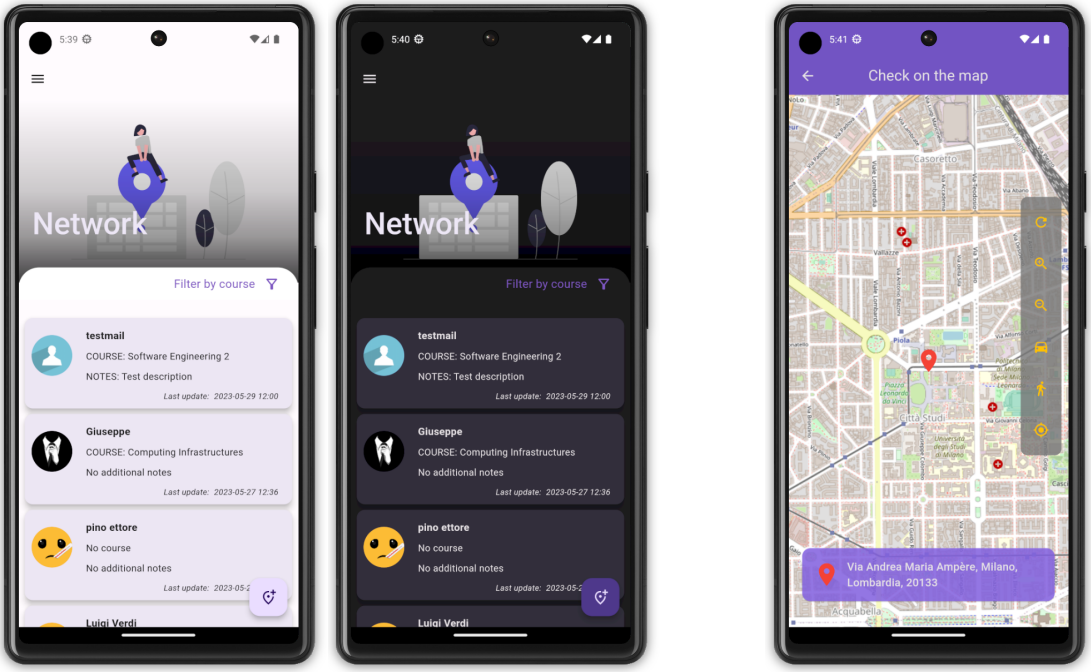
The first screen displayed when the application is opened for the first time or after a logout is the login page. A text button is used to switch from the login to the registration form and vice versa. Google Authentication is also available.

3.2.2. Course Page



This screen is the first displayed to a logged user, it contains the list of the courses which the user is subscribed to. A button is present to add a new course, while a left swipe is needed to delete an already present entry. Two possible views of the same list are available (compact and extended).

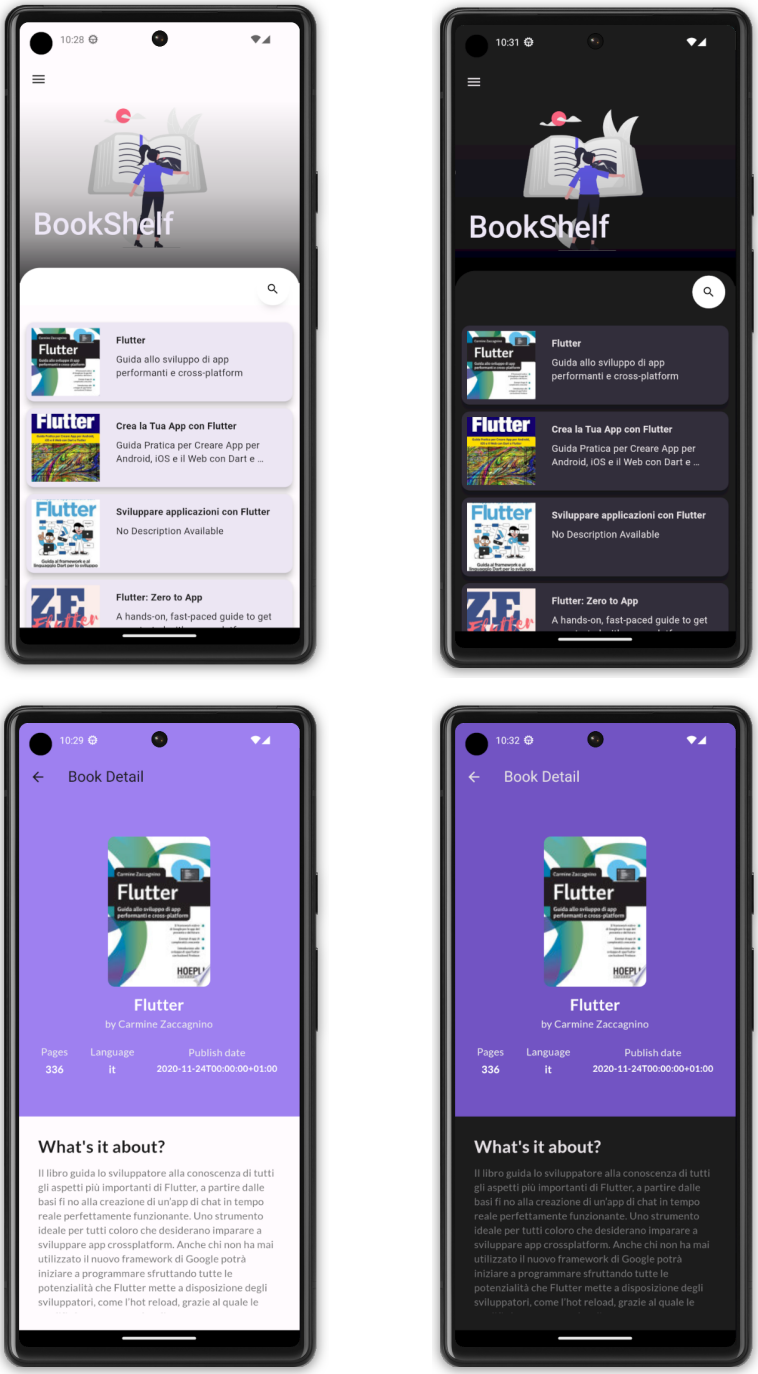
3.2.3. Network Page



This screen is dedicated to the network functionalities: you can share your position and the course you are studying with a small description, in order to allow other students to reach you. By tapping on a card you can see the position of a user on the map and, if the location

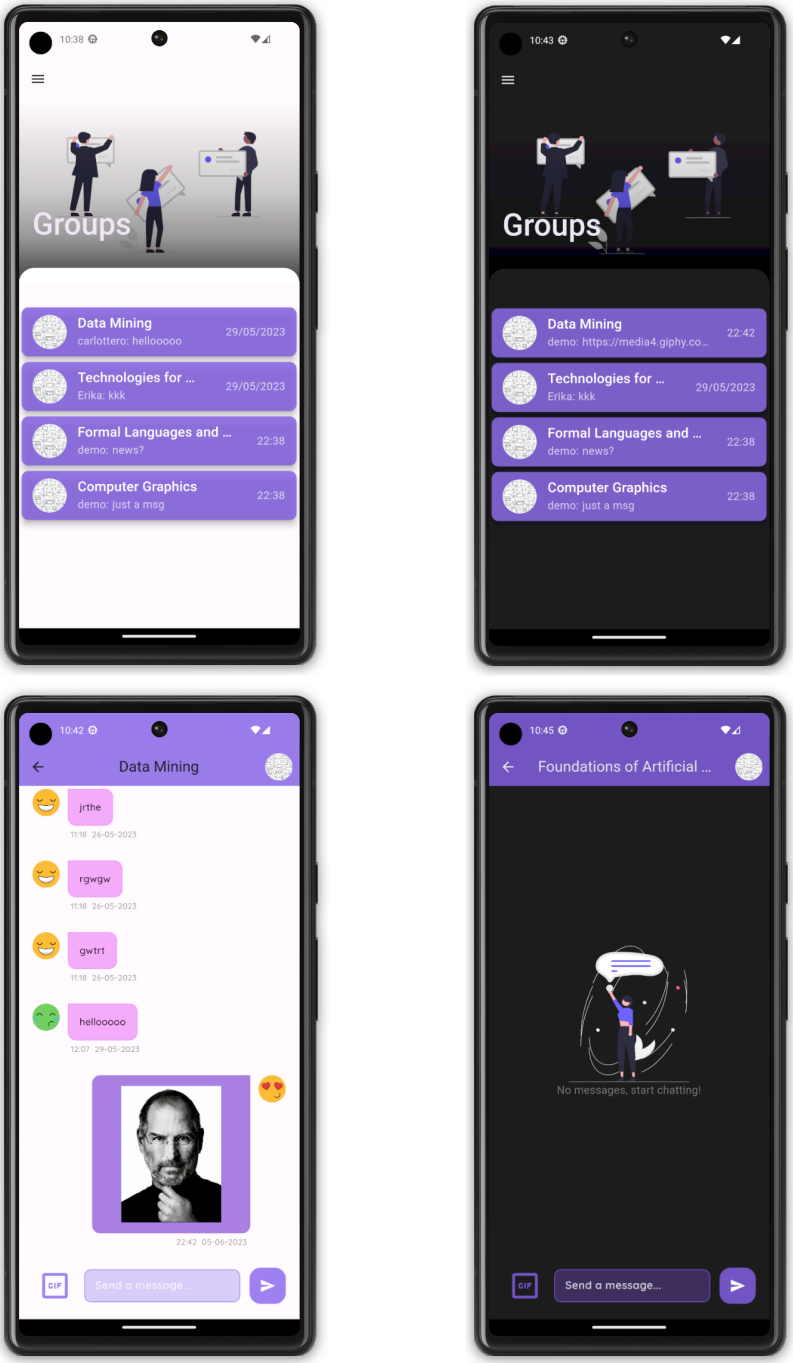
services are enabled, follow a route to reach him, by car or on foot. Not enabling location services will disable this functionality. Your shared position is highlighted on the top (if present) and it can be deleted with a left swipe. Pulling down will refresh the list.

3.2.4. BookShelf Page



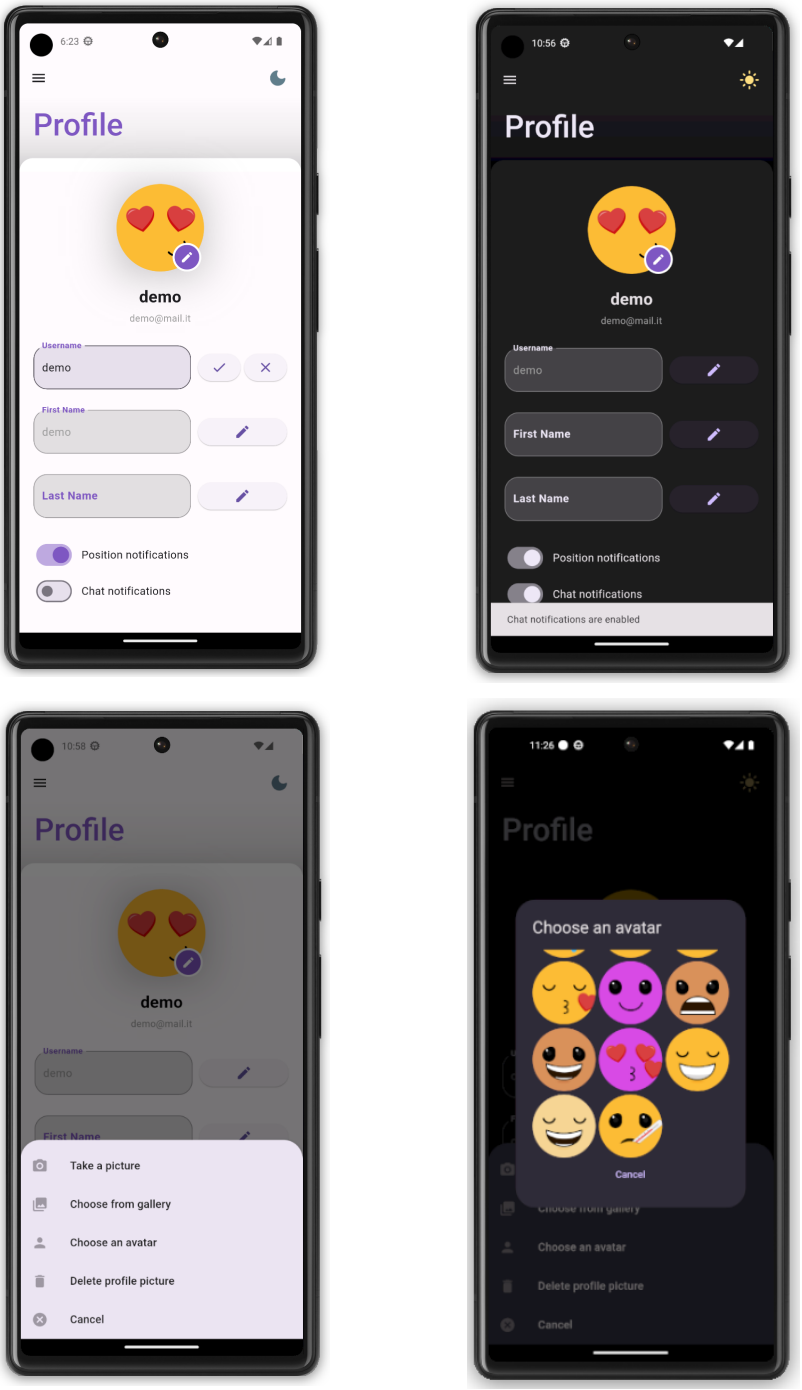
This screen is the one dedicated to the book functionality: in this page Google Books API is used to retrieve all available books information. A search button is present and allows you to search for a specific book. Tapping on a book tile moves the current screen to the book detail page, where more information about the selected publication is shown.

3.2.5. Group Chat Page



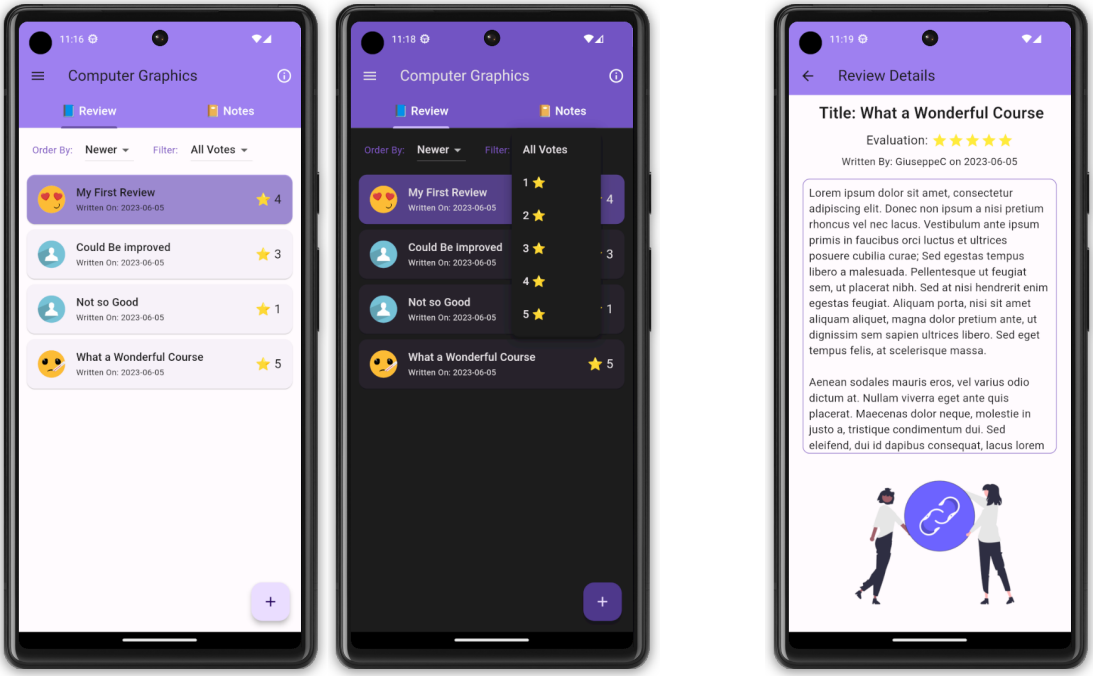
This screen is the one dedicated to the chat functionality: a group chat appears when the course is added to your list; by tapping on a tile you enter the chat where you can send messages, use Giphy API to forward fancy gifs to your colleagues, or tap on a user avatar or group icon to show some details related to it.

3.2.6. Settings Page



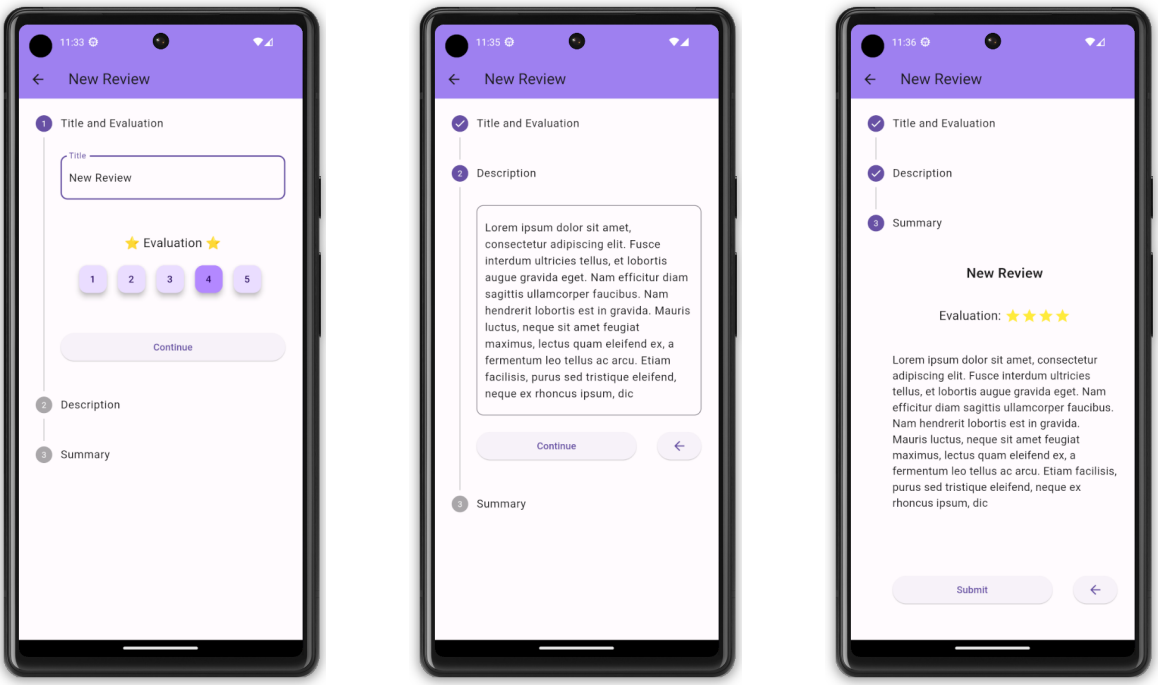
The Settings page is dedicated to the customization of your profile: here you can change your username, first and last name, change the preferences about notifications and, by tapping on the profile image, a pop-up menu appears showing the possible ways to change your profile picture. Last but not least, by tapping on the top right-hand corner you can also change the overall theme of the application (light or dark).

3.2.8. Course Reviews



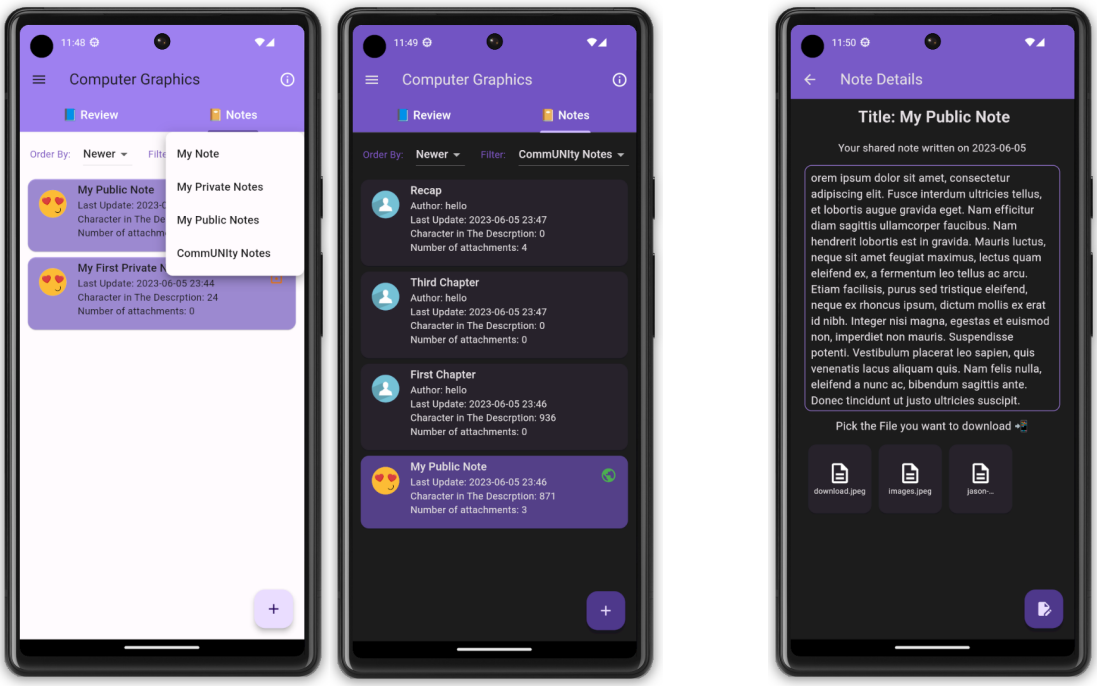
This page is the first shown when you tap on a course tile in the course page (3.1.2). The list of all the reviews is present, 2 filters are available: one to retrieve only the reviews with a specific vote, the other one is used to sort the list according to the timestamp. Clicking on a review opens a new page, where you can comfortably read all the details of it. A different color is used to highlight your reviews, each of which can be deleted with a swipe left.

3.2.9. Add Course Review



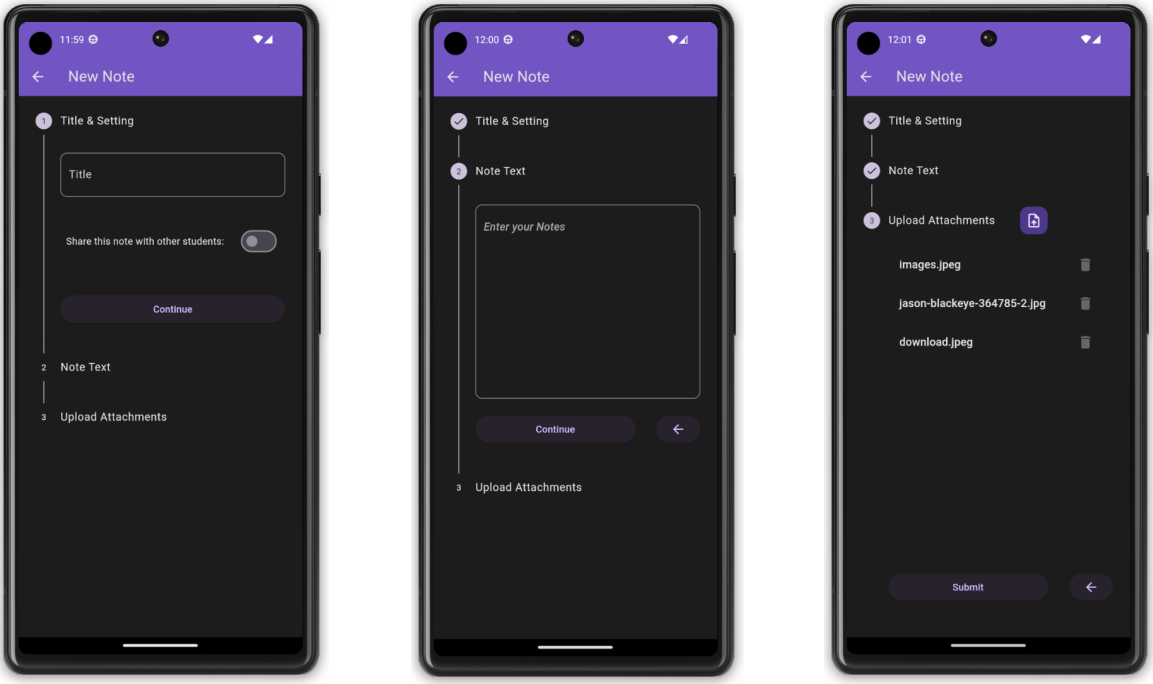
The form used to add a review is implemented using a stepper: this choice is useful to split all the different steps of the form for a more comfortable and safe immission of data. At the end of the form a summary step is shown to check the data before the submission.

3.2.10. Course Notes



The other tab of the course page is the Notes page, which is implemented with the same design of the review tab inherits the same functionalities. A different filter is implemented to select between your notes and the community ones. By tapping on a note, all the details are shown, including all the attachments that can be downloaded and stored in local memory. A modify button is shown in your personal note to let you change its detail.

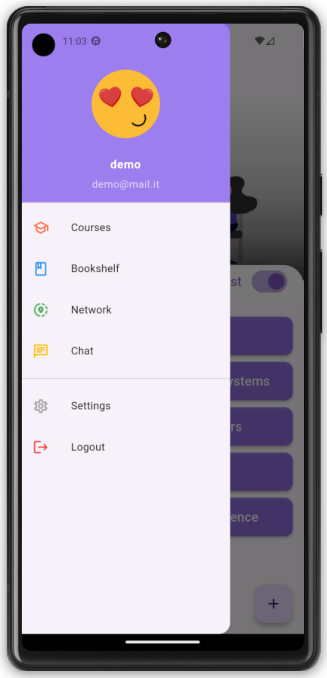
3.2.11. Add Course Note

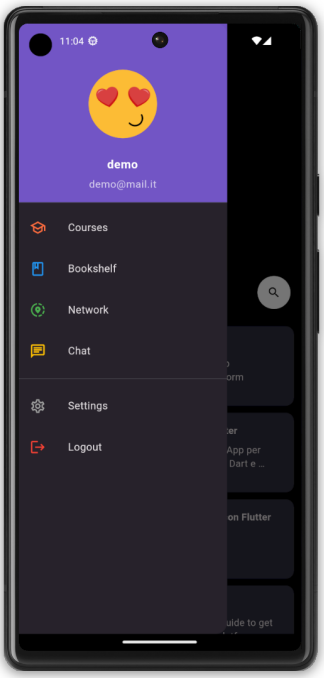


Same concepts of new review are used. The major changes regards the attachments, where a file_picker is used to select the files to upload in the note from the local memory. The same screen is used to modify a note, for the sake of continuity.

3.2.7. Drawer

For navigation a drawer is used, where all available functionalities are present. Also, the main information about the user is shown in the upper part.

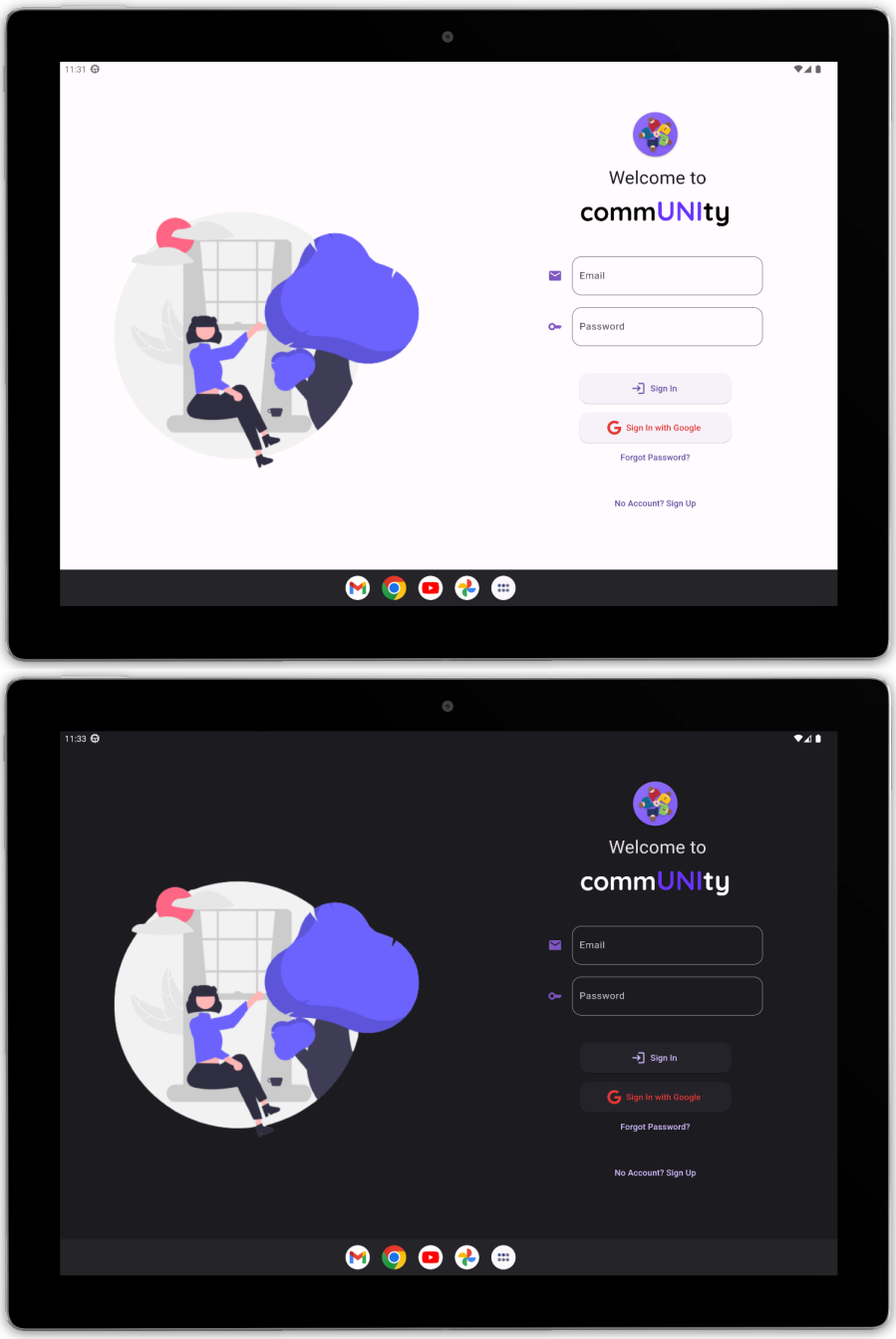




3.3. Tablet UI

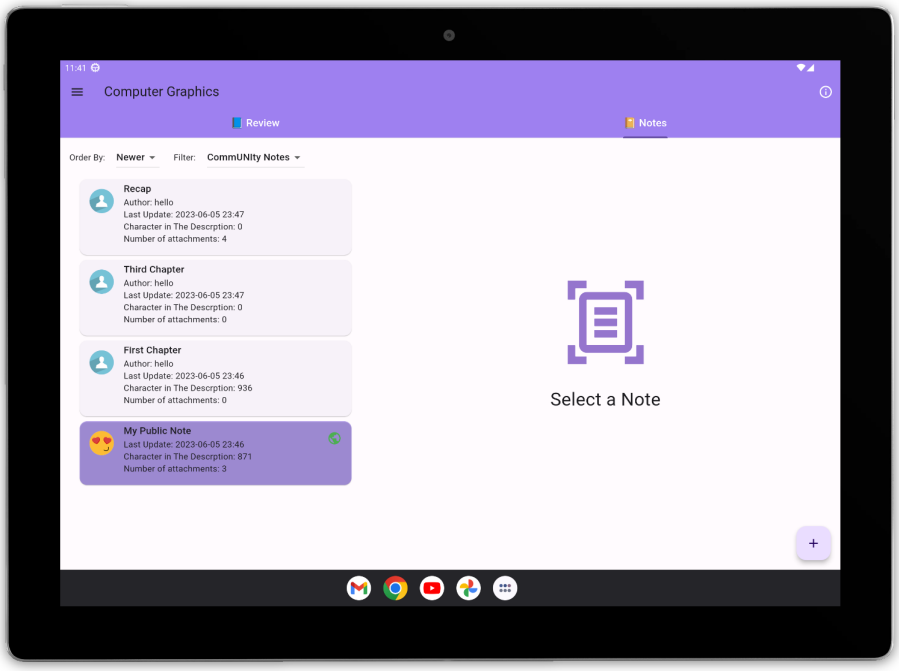
The User Interface changes dynamically according to the screen size: this is done in order to improve the usability and the *look-and-feel* of the application. In this paragraph some of the most relevant page modifications w.r.t. the smartphone design is shown.

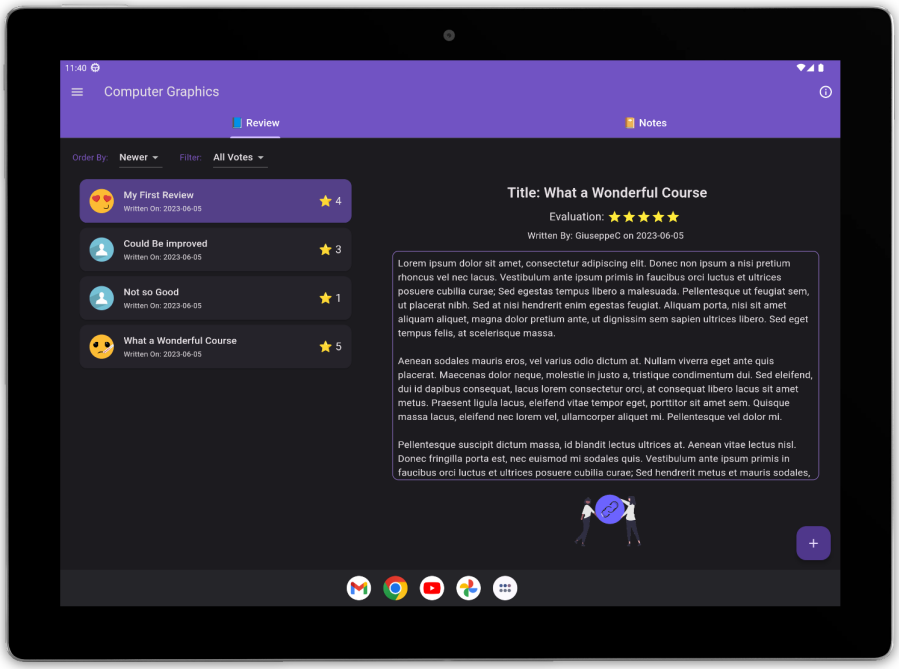
3.3.1. Login Page and Minor Changes



Some of the new pages designed for bigger screens do not add further functionalities to the app, but they improve the overall design by exploiting the new available space. The design is also chosen to be coherent with smartphone UI.

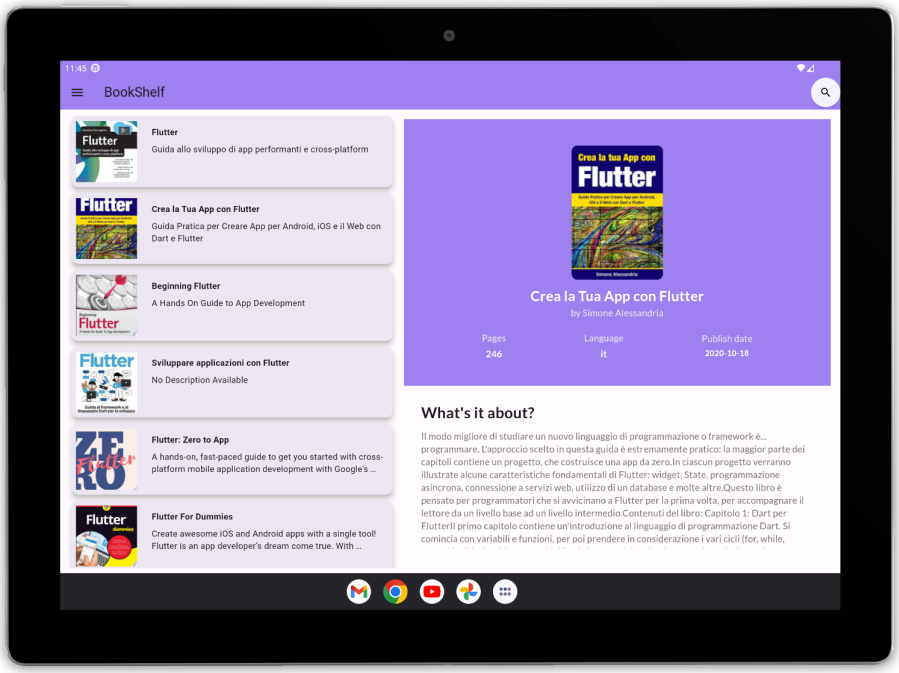
3.3.2. Course Page: Note and Review

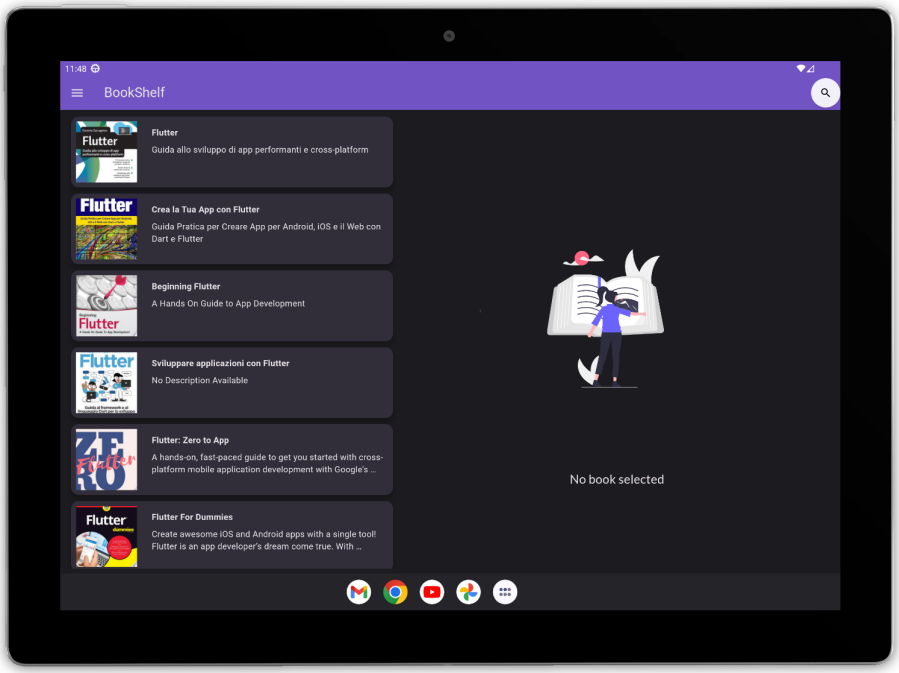




In the Course Page, thanks to the new width size, we managed to implement the list of reviews\notes side by side with the detailed review\note: this choice is done to improve the navigation between different elements of the lists.

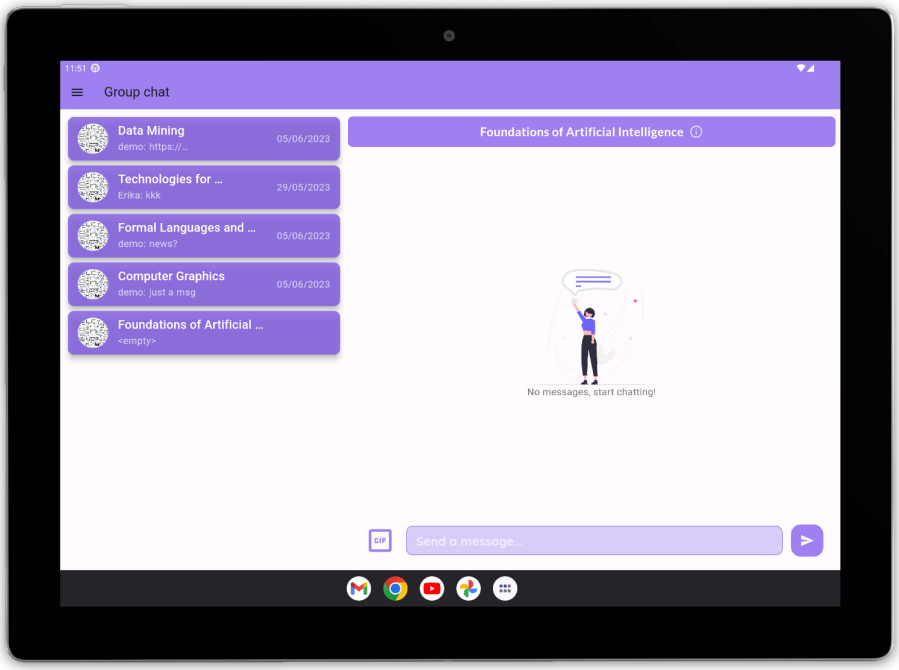
3.3.3. BookShelf Page

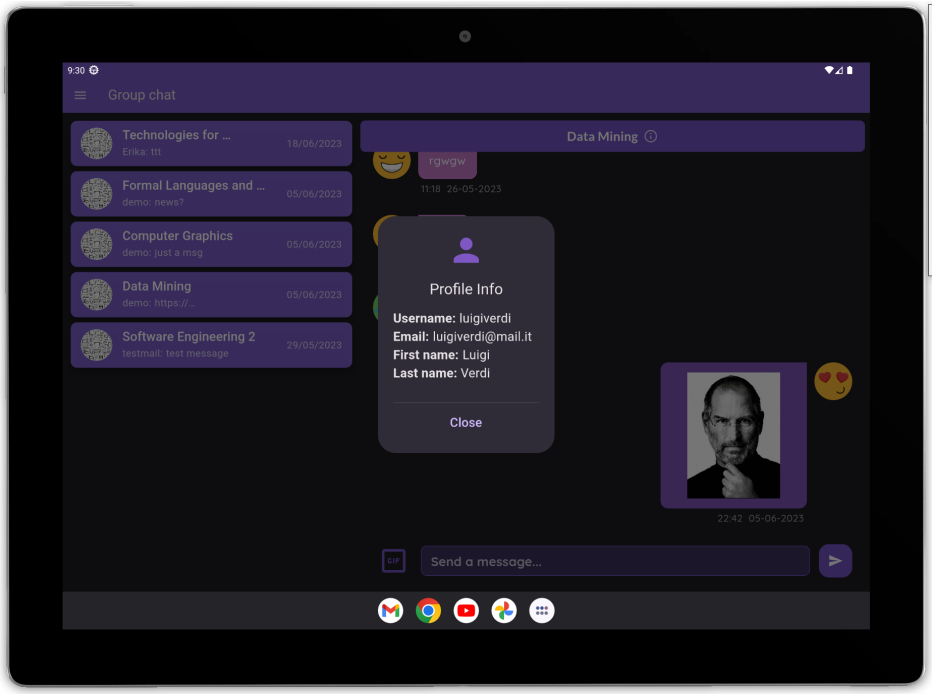




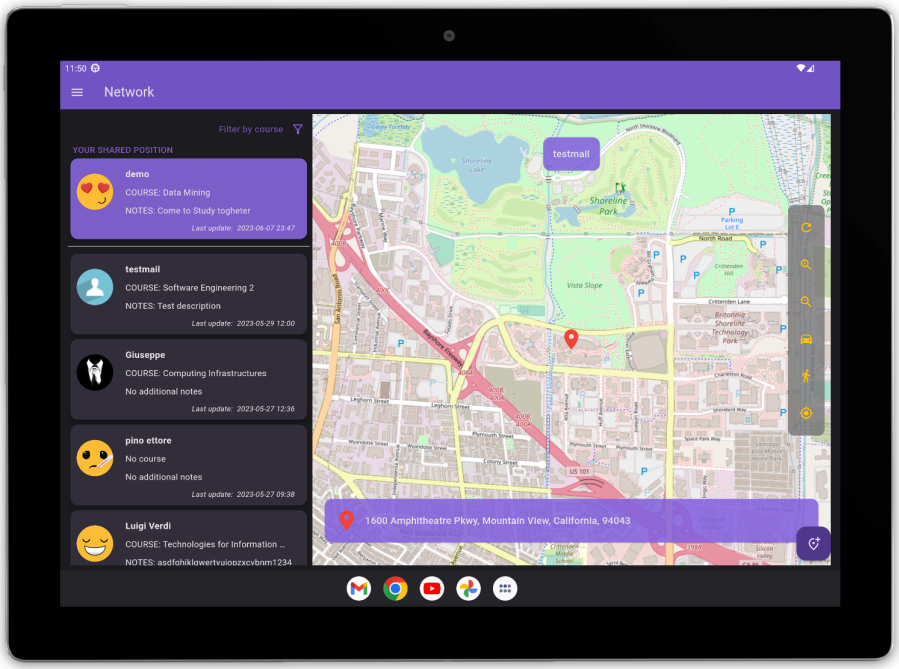
The same concepts of the Course page are used in the other pages, like the bookshelf, for the sake of design coherence and navigation facilities.

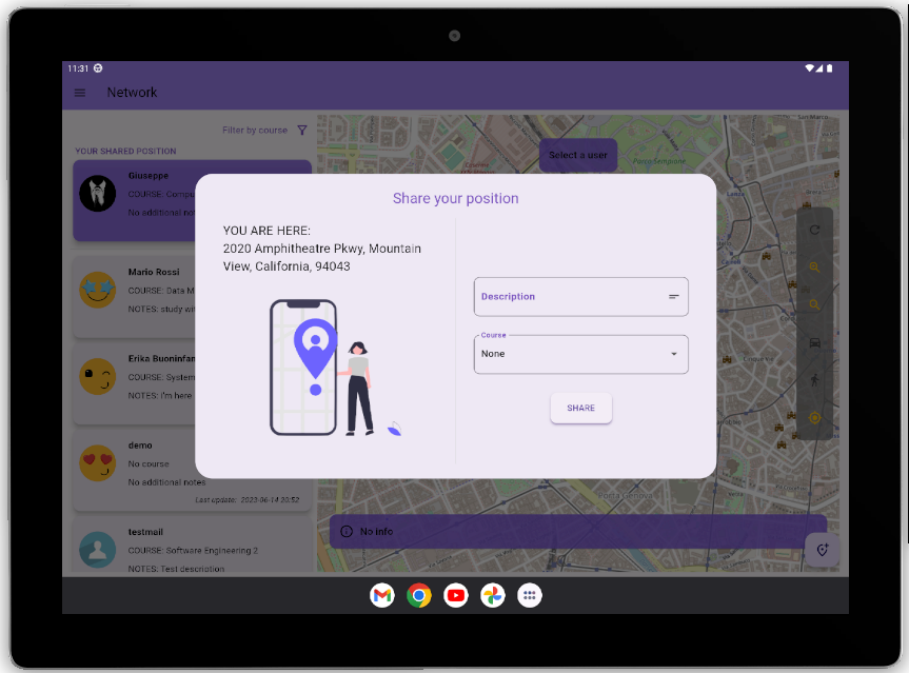
3.3.4. Group Chat Page





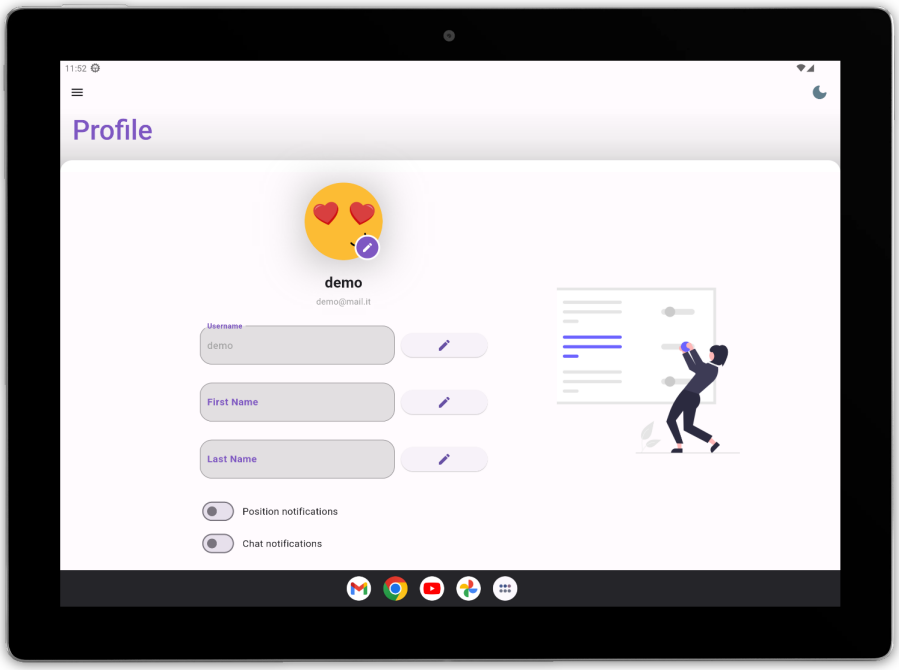
3.3.5. Network Page





The main difference w.r.t. smartphone screen is that, when the user first opens the Network page, a pop-up asking for localization permissions is shown, while in the smartphone version this happens only if clicking on a position tile or when opening the share position form. It works in this way because here the map is shown immediately on the page.

3.3.6. Settings Page



4. Testing Campaign

4.1. Testing Environment

The tests developed for the application are performed in 2 different environments according to the kind of test to perform:

- **Mock Environment**: In this environment all the APIs are replaced with a mock version of them; this is done to separate the behavior of the external service with respect to the implemented function that we tested and also to overcome the limitation of the flutter test environment, where no external services are working. Thanks to the architecture of the application, this environment is implemented simply by changing the implementation of our service classes with static data.
- **Deploy Environment:** In this environment the tests are supposed to run on a real device, this is costly and slows down the testing process, but is needed to test the integration of our app with the reliable external services we choose and also to test the integration of different parts of the app.

4.2. Unit Test

Flutter unit tests are supposed to run in the *mock environment* due to flutter constraints. Our app heavily depends on external services so most of the logic implemented is a trivial call to them with different parameters. Because of these limitations, no reasonable unit tests were to be performed.

4.3. Widget Test

The aim of this type of testing is to ensure that a proper widget is loaded and populated successfully, by feeding it with some data coming from mock APIs and checking through proper assertions if it is correctly displayed.

Flutter Widget tests run in the *mock environment* and they are structured as follows: test granularity is based on app screen, where for each screen, that is composed mainly by a group of basic widget, all the possible use cases are tested.

Both tablet and smartphone widgets are tested on the same kind of test, with small modification to match the relative design.

The list of the most important tests is available below (divided by screen):

Page tested	Name of the Test (description)
Login\Registration	Check registration with correct input
Login\Registration	Check login with empty input
Login\Registration	Check login with correct input
Login\Registration	Check forget password

Login\Registration	Check registration with empty input
Login\Registration	Check registration with wrong input
Course Page	Check render if no course subscription
Course Page	Check render if course subscription present
Course Page	Check render of all types of list
Course Page	Check add course render
Course Page	Check slider for course deletion
BookShelf	Check initial render screen
BookShelf	Check render if no book found
BookShelf	Check render if book found
BookShelf	Check render if click a book tile
Chat List Page	Check initial render
Chat List Page	Check render if no group available
Chat List Page	Check render if click on a group tile
Chat	Check group info dialog render
Chat	Check chat screen render
Chat	Check profile info dialog render
Network	Check initial render
Network	Check course filter
Network	Check delete your position dismissible
Network	Check refresh position list
Network	Check share position form
Network	Check open map render
Course Note	Check initial render
Course Note	Check filter menu
Course Note	Check empty list note render
Course Note	Check show note render
Course Note	Check delete a note dismissible
New Note	Check initial render
New Note	Check no title error render
New Note	Check go back button

Course Review	Check course information dialog
Course Review	Check filter menu
Course Review	Check empty list render
Course Review	Check show review render
Course Review	Check delete review dismissible
New Review	Check initial render
New Review	Check no title error render
New Review	Check go back button
Settings	Check initial render
Settings	Check open bottom dialog
Settings	Check delete user picture
Settings	Check choose avatar menu
Settings	Check edit field
Settings	Check cancel edit field
Settings	Check empty edit field
Settings	Check notification switch
Settings	Check snackbar

4.3.1. Coverage Analysis

When all the test designs were implemented, a coverage analysis of the code was done and the result is that on over 50 tests for view (tablet and smartphone, 100 tests in total) performed, the coverage on our code is 91%. A manual analysis is done on uncoverage code to check that the lines not included in the tests are not relevant for this kind of tests.

The coverage data are generated exploring flutter testing features, then a html report is generated for a more human readability. The report shows for each source file the line coverage percentage and the missing lines.

Directory	Line Coverage		
lib		100.0 %	7 / 7
lib/Atest_lib		96.7 %	900 / 931
lib/Atest_lib/classes		100.0 %	2 / 2
lib/Atest_lib/services		89.1 %	115 / 129
lib/Atest_lib/widgets/books		100.0 %	95 / 95
lib/Atest_lib/widgets/courses		92.5 %	196 / 212
lib/Atest_lib/widgets/courses/pages		84.4 %	222 / 263
lib/Atest_lib/widgets/group_chat		97.0 %	291 / 300
lib/Atest_lib/widgets/login		90.1 %	118 / 131
lib/Atest_lib/widgets/network		95.1 %	486 / 511
lib/Atest_lib/widgets/settings		94.7 %	358 / 378

4.4. Integration Test

This phase is an extension of the previous widget testing. In this case, we will focus on how different widgets interact with each other, and thus not on the single instance.

These kinds of tests are performed in the *deploy environment*: in this way we manage to test the stability of the system with the external APIs, reaching a good satisfaction level from this point of view.

Because of the nature of the test, we decided that the test should be imprinted to test the action needed to perform the main actions of the application starting from the initial page.

The resulting tests are:

- Sign in and go to the home page
- Log in and go to the home page
- Open a review
- Write a review
- Open a note
- Write a note
- Start to modify a note
- Open a book
- Share the current position
- Open a position of the list
- Write a message in the chat
- Open the settings and update a field
- Logout

4.5. User Test

This phase of the testing is performed to check mostly the interface design with people that have no bias from the implementation of the app.

The tests are performed taking a very restricted number of people and let them use the app in 2 different ways:

- perform an action taken from the Integration Test list
- free navigation in the app

The former test was useful to check if the most relevant functions of the application are easily accessible; the latter was used to check if the design of the app drives the user to perform the action he wants to do intuitively.

Due to the involvement of external people, this kind of test was by far the most expensive in terms of time according to the number of tests performed, but it gave us good quality feedback for the user interface.

5. Future developments

CommUNIty led to the development of several features in different spheres of interest of the mobile application design and development. The source code written since the first version release of the application covers most of the features we had in mind, but due to time constraint we decided to postpone the developments of secondary tasks that are less relevant for the “design and implementation of mobile application” course.

Thanks to the modular architecture of the application, all the future developments do not require modifying the source code written since now.

A list of future improvement is shown below:

- Improve the support for digital hand-written notes with capacitive pen for tablet: improving the support for capacitive pen amplifies the use cases of the application. These features do not add screen complexity and do not match the topics of the course, so we decided to leave it for the future.
- Substitute the back-end of the application with a custom one (and not dependent from firebase) for more customizable backend operations. Even this time, this feature was not immediately implemented because the choice of the back-end do not affects the implementation of the application in a strict sense (different type of backend do lead to the flutter design and implementation choice) so we decided to focus our attention on the front-end of the application, using a pre-built back-end. This led to some limitations in data management (like the absence of text-search) that could be overcome in a second moment.
- Start collaborations with universities to have access to their course database for a complete coverage of the courses data: having access to university data could be not so easy (and require lots of bureaucracy), so we decided to skip this step for the first implementation of the application.
- Implement administrator services for courses and chat data check: one of the design choice of the app is that not all data can be modified by a simple user (like the list of the courses or the messages in chats); this requires additional administrative services that, in this first release of the application, are carried out for the Firebase web-app available for our back-end.
- Extend the support for Apple devices: one of the most interesting features of flutter is the cross-platform one. Due to hardware (and budget) limitations, the deployment of CommUNIty on IOS devices was not tested.