POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

bloQo

Design Document

Authors: **Matteo Spreafico 10669138**

Ludovica Tassini 10663137

Course: **Design and Implementation of Mobile Applications**

Professor: **Luciano Baresi**

Academic Year: 2023-2024

Abstract

This document aims at describing the idea behind the "bloQo" project, including its features, its technical choices, its architecture, and its user interface design. Moreover, all the external services and plugins used will be addressed. Finally, detailed insights about the testing campaign will be provided.

Contents

1 | Introduction

In this chapter, we introduce the idea behind bloQo and the project's main features.

1.1. Project Description

For the Design and Implementation of Mobile Application course at Politecnico di Milano, we developed a mobile application called **bloQo**. The application aims at providing users with a single platform that allows them to learn from courses that other users create and share. To achieve this goal, the application offers the following main functionalities:

- A powerful, easy-to-use **in-app editor** that allows user to build courses from scratch and eventually publish them for the community.
- A **learning section** devoted to the courses in which users are enrolled.
- A **search engine** that allows users to discover and eventually enroll in the courses they can best enjoy.

These main functionalities are further supported by many other elements:

- Once a user completes a course they are enrolled in, they can **rate** it by leaving a star rating and a textual review. When users search for courses, they can view its average rating and read all textual comments, if present, which may provide helpful advice on whether to enroll or not.
- Once a user completes a course they are enrolled in, they can even generate a PNG **certificate** that proves the completion of the course. We believe that sharing it through social media may increase the users' engagement, potentially persuading other users to enroll to the same course.
- The application allows users to **follow each other**. When someone publishes a course, all users who follow the author will be notified thanks to an **in-app notification system**. Furthermore, we believe that, if the author of a course has a considerable amount of followers, such a good fact could positively affect the appeal of their courses.

- The author of a course may decide to **make its course private**. This means that other users cannot automatically enroll, but have to ask the author for enrollment. Practically, this means that the course author receives a special notification that gives them the control to decide whether to accept or deny the request. If the author accepts, the applicant will be notified. We have tailored this feature thinking of tutors who may desire to give access to their courses only to specific users.
- Users and courses can be easily searched thanks to **QR codes** that can be scanned via the **in-app camera**. Such codes can be easily obtained in various ways.

Below, we present the icon of bloQo's launcher. A splashscreen showing the logo together with the name of the application appears on application startup.

Figure 1.1: The logo of the application

1.1.1. Reasons Behind The Name

As a side note, we want to address the reasons behind the application's name (**bloQo**):

- The main reason is that courses are made of several, basic "building blocks" that are combined together. The next section will address how courses can be crafted in a more detailed manner.
- To give uniqueness to the name, we transformed the reference word "block" into "bloQ". These two words sound the same, but - as the logo reflects - the "Q" character also resembles a *magnifying glass*, which is (symbolically) strictly related to knowledge.
- The final "o" we added to "bloQ" allows the name to sound more "fun" and recognizable. As the primary focus of the application is education, we believe that this

could help children and adults to best appreciate our project.

1.2. Project Features

Now, we present how the main features of the application have been implemented.

1.2.1. Account Creation

To access bloQo's core functionalities, users need to be registered. This is possible by filling out a simple form that asks for an email, an appropriate password, a unique username and the user's full name. The user's full name is asked for certificate generation. If the user does not want to make their full name visible to other users, they can choose so. The application remembers your credentials so there is no need to log in again, but logout is possible at any time by pressing a button in the account and settings page.

1.2.2. Course Creation

In bloQo, courses have a title and an optional description. Each course features one or more **chapters**. Each chapter has a title and an optional description, and features one or more **sections**. Each section has a title and features one or more blocks. These blocks are the true *building blocks* of the courses, as they hold actual content. As of now, there are currently three types of blocks that users can create:

- **Text Blocks**. Users can type any text they wish into these blocks. These blocks also support a subset of the **Markup language**, so users can enrich their text with elements such as bold text, italic text, bullet lists, and so on.
- **Multimedia Blocks**. Users can add various kinds of multimedia files:
  - **audio files**. They must be uploaded from the device.
  - **image files**. They must be uploaded from the device.
  - **video files**. They can either be uploaded from the device or can be **embedded from YouTube** by providing the YouTube video link.
- **Quiz Blocks**. Users can create quizzes to assess the users' understanding of the presented topics. As of now, there are two types of quizzes that can be built:
  - **multiple choice quizzes**. The in-app editor allows users to enter a question and to generate a variable number of answer choices (at least one). At least one answer should be marked as correct, meaning that multiple correct answers

are possible.

- **open question quizzes**. The in-app editor allows users to enter a question and its answer. Users can control some parameters that may affect the comparison, such as the possibility of ignoring the case.

Notice that courses, chapters, sections, blocks, multimedia files and multiple choice answers can be individually deleted.

The home page displays the courses that a user is currently creating. Therefore, users can rapidly go to the page that allows the modification of a course's content from the home page as well.

At any time, users can **preview** a course they are building, to see how it would look like to somebody that enrolls in it.

1.2.3. Course Publication

Courses can be published by its author anytime. Before the course is actually published, the author has to decide whether the course should be public or not, and assign **tags** to its course. Tags help users filter for the courses they can best like. The author has to assign a tag value for each of the five tag categories:

- **Language**. It helps users understand the language in which a course is provided. As choices that differ from "Other", it supports the languages in which the application is localized.
- **Subject**. It helps users understand the general subject of the course topics. The application offers a wide range of subject choices.
- **Duration**. It helps users understand the time effort that a course requires in order to be completed. The application offers a range of choices that span from "less than one hour" to "more than three hours".
- **Modality**. It helps users understand whether the course offers only lessons, only quizzes, or both.
- **Difficulty**. It helps users understand whether the course is suitable for everyone or only for expert users.

1.2.4. Published Course Management

Course authors can view the content of their courses at any time, but cannot modify it as long as it is published. They can **view statistics** such as the number of people who enrolled in the course, the number of people who completed the course, and the reviews that have been published so far. Course authors can **dismiss** a published course, meaning that they can delete its "published" instance and automatically prevent previously enrolled users to access its content again. A dismissed course can be modified, but will be presented as a brand new one when published again.

1.2.5. Course/User Search and Enrollment

Users can search a course or a user either by scanning their QR code through the in-app camera or by filling-out a number of forms at user's choice among the ones available in the search page. These forms allow to enter the course's title, the course author's username, the minimum publication date, the maximum publication date. Users can even choose to filter out public or private courses. There are also choice menus that allow users to filter by tag and to order search results according to one of several criteria. Once the search criteria are set (or the QR code has been scanned), bloQo's search engine will return the result. If a QR code has been provided, the user is automatically redirected to the course/user page, otherwise the user can choose one of the search results and visit its page. A user can see the course's description and have a preview of its content when they are in a course page, and eventually choose to enroll in it (or to ask for enrollment). A user can see all the courses another user has published when they are in a user page, and eventually start to follow them.

1.2.6. Course Completion

Users can learn the content of the courses they are enrolled in. The application enforces the rule that course sections should be followed in order, but sections that have already been completed can be freely accessed at any time. A section is marked as completed when the user presses the appropriate button at the end of its blocks. However, if a section contains some quizzes, such button cannot be pressed if all the correct answers have not been submitted. Notice that users can understand if a multiple choice quiz has a single or multiple answers by the type of the check boxes. Plus, notice that users can control the streaming of an audio or a video media with a dedicated widget. Once a course is completed, it can be rated and the certificate can be generated. Users can unsubscribe from courses as long as they are not completed.

The home page displays the courses that a user is currently enrolled in. Therefore, users can rapidly go to the page that displays a course's content also from the home page.

1.2.7. Notification Management

Users can read the in-app notifications anytime by pressing the appropriate icon on the top-right corner of each page. Users can delete the notifications they have read by pressing the "Okay!" button on the widget. As already mentioned, there is another kind of notification that allows a user who published a private course to accept or deny enrollment requests. As one of these two options is chosen, the notifications will be deleted.

1.2.8. User Experience Customization

bloQo allows users to modify some parameters to get the best experience of use. More specifically, users can:

- **change the language of the application**. As of now, the application is completely localized in English and Italian.
- **change the appearance of the application**. As of now, two themes are available.

1.3. Technical Choices

The application has been developed using the **Flutter** framework. Therefore, the application is available for both Android-based and iOS-based devices, provided that:

- Android devices run with Android 6.0 or superior;
- iOS devices run with iOS 15.0 or superior.

Additionally, during design and development, **both smartphones and tablets** have been targeted. As a result, the application displays different layouts based on the screen width.

Since the developers use only iOS-based devices, they have been targeted the most during user testing. However, thanks to simulated devices and human testers, we have been able to thoroughly test Android devices as well.

2 | Application Design

In this chapter, we describe the architectural and design choices behind our implementation.

2.1. Architectural Choices

The application can be seen as a **two-tier** architecture, with a *thick* client. The client is the app itself, whereas the server is a **Firebase back-end**. We consider the client as "thick" because the Firebase server works only as a **database, storage, and authentication service**, and data are manipulated by the devices themselves. In the context of the university course, this solution was the easiest and came with no costs. However, if the application had to be published in a real-world store with a fast-growing audience, we believe that some data manipulation functions (such as the notification generation when a user publishes a course) - currently handled by the client - should be handled by the server instead.

2.1.1. Authentication

Users can login and register via **Firebase Authentication**. Currently, the supported sign-in method is **email** + **password**. Each user is then assigned a unique identifier that will be used in the Firebase Firestore to get and set user data. Notice that, in order for a password to be valid, it has to:

1. be at least 8 characters long;
2. be at most 32 characters long;
3. have at least one number;
4. have at least one special character;
5. have at least one lowercase letter;
6. have at least one uppercase letter.

2.1.2. Data Management

The application relies on the online and in-cloud database **Cloud Firestore**. The NoSQL nature of Cloud Firestore suits well the unstructured nature of course data. Aside from course data, such database also stores data about users, notifications, and reviews.

Our Cloud Firestore instance has the following collections:

Collection Name	Collection Description
blocks	It contains data about section blocks.
chapters	It contains data about course chapters.
courses	It contains data about courses. It can be accessed either by the course creator or by the users who enrolled in the published course document related to this course.
notifications	It contains data about notifications. The available fields differ based on the notification type.
published_courses	It contains data about published courses.
reviews	It contains data about course reviews.
user_course_created	It contains data about the relationship between a user and a course that they created.
user_course_enrolled	It contains data about the relationship between a user and a course that they enrolled in.
users	It contains data about users.

Table 2.1: Database Collections

Each document of every collection except "user_course_created" and "user_course_enrolled" has a unique UUID identifier that helps resolve references among collections.

"blocks" document fields

A typical document has the following fields:

- content. It is the string containing the block's content (it can be plain text, a link to a multimedia file, a formatted string that contains quiz data, ...).
- id. It is the block's unique UUID.
- name. It is name of the block. It is automatically generated based on content, and cannot be modified by the user (unless the type of the block changes).

- number. It is the ordering number of the block in its section.
- super_type. It is the string representation of one of the three main categories of blocks.
- type. It is the actual block type (e.g. multimedia blocks are divided into audio blocks, image blocks, and video blocks).

"chapters" document fields

A typical document has the following fields:

- description. It is the (optional) description of the chapter.
- id. It is the chapter's unique UUID.
- name. It is the chapter's name. It is initialized with a default value, but can be modified by the user.
- number. It is the ordering number of the chapter in its course.
- sections. It is an array containing the ids of the chapter's sections.

"courses" document example

A typical document has the following fields:

- author_id. It is the id of the course author.
- chapters. It is an array containing the ids of the course's chapters.
- creation_date. It is the timestamp in which this course was created.
- description. It is the (optional) description of the course.
- id. It is the course's unique UUID.
- name. It is the course's name. It is initialized with a default value, but can be modified by the user.
- publication_date. It is the timestamp in which this course was published, or it is null if the course has not (yet) been published.
- published. It is true if this course has been published, false otherwise.

"published_courses" document example

A typical document has the following fields:

- author_id. It is the id of the course author.
- author_username. It is the username of the course author. We decided to pay a small amount of data redundancy to diminish the number of queries.
- course_name. It is the name of the published course. Again, we decided to pay a small amount of data redundancy to diminish the number of queries.
- difficulty. It is the string representation of the difficulty tag value chosen.
- duration. It is the string representation of the duration tag value chosen.
- is_public. It is a flag that denotes whether the published course is public or not.
- language. It is the string representation of the language tag value chosen.
- modality. It is the string representation of the modality tag value chosen.
- number_of_completions. It is the number of people who completed this course.
- number_of_enrollments. It is the number of people who enrolled in this course.
- original_course_id. It is the id of the course this document refers to.
- published_course_id. It is the published course's unique UUID.
- rating. It is the average of the star ratings the published course has received.
- reviews. It is an array containing the ids of the published course's reviews.
- subject. It is the string representation of the subject tag value chosen.

"notifications" document example

A typical document has the following fields:

- id. It is the notification's unique UUID.
- timestamp. It is the timestamp in which the notification was created.
- type. It is the string representation of the notification type.
- user_id. It is the id of the user who has to receive the notification.

Based on the notification type, there are further fields:

- if the notification is an enrollment request, "private_published_course_id" and "applicant_id" are requested;
- if the notification is an accepted enrollment request, "private_course_name" and "private_course_author_id" are requested;
- if the notification is an update about a new course being published by someone the user is following, "course_author_id" and "course_name" are requested.

"reviews" document example

A typical document has the following fields:

- author_id. It is the review author's id.
- author_username. It is the username of the review author. We decided to pay a small amount of data redundancy to diminish the number of queries.
- comment. It is the main textual content of the review.
- comment_title. It is the title of the review.
- id. It is the review's unique UUID.
- rating. It is the star rating assigned to the review.

"sections" document fields

A typical document has the following fields:

- blocks. It is an array containing the ids of the section's blocks.
- id. It is the section's unique UUID.
- name. It is the section's name. It is initialized with a default value, but can be modified by the user.
- number. It is the ordering number of the section in its chapter.

"user_course_created" document fields

A typical document has the following fields:

- author_id. It is the id of the course author.
- course_id. It is the id of the course its author is creating.

- course_name. It is the name of the course its author is creating. We decided to pay a small amount of data redundancy to diminish the number of queries.
- last_updated. It is the timestamp of the latest time the author modified the course.
- num_chapters_created. It is the number of the chapters currently present in the course. We decided to pay a small amount of data redundancy to diminish the number of queries.
- num_sections_created. It is the number of the sections currently present in the course. We decided to pay a small amount of data redundancy to diminish the number of queries.
- published. It is true if the related course has been published, false otherwise. We decided to pay a small amount of data redundancy to diminish the number of queries.

"user_course_enrolled" document fields

A typical document has the following fields:

- chapters_completed. It is an array containing the ids of the course chapters the user has completed.
- course_author_id. It is the id of the author of the course the user is enrolled in.
- course_author_username. It is the username of the author of the course the user is enrolled in. We decided to pay a small amount of data redundancy to diminish the number of queries.
- course_id. It is the id of the course the user is enrolled in.
- course_name. It is the name of the course the user is enrolled in. We decided to pay a small amount of data redundancy to diminish the number of queries.
- enrolled_user_id. It is the id of the user who is enrolled in this course.
- enrollment_date. It is the timestamp in which the user enrolled in the course.
- is_completed. It is true if the related course has been completed by the user, false otherwise.
- is_rated. It is true if the user published a review for the related course, false otherwise. We decided to pay a small amount of data redundancy to diminish the number of queries.

- last_updated. It is the timestamp of the last change in the number of course chapters or sections learned.
- published_course_id. It is the id of the published course instance related to the course. We decided to pay a small amount of data redundancy to diminish the number of queries.
- section_name. It is the name of the next section the user has to complete. We decided to pay a small amount of data redundancy to diminish the number of queries.
- section_to_complete. It is the id of the next section the user has to complete.
- sections_completed. It is an array containing the ids of the course sections the user has completed.
- tot_num_sections. It is an integer representing the total number of sections present in the course, and allows easy calculation of the user's progress bar related to the course. We decided to pay a small amount of data redundancy to diminish the number of queries.

"users" document fields

A typical document has the following fields:

- email. It is the email of the user.
- followers. It is an array containing the ids of the users who follow the user.
- following. It is an array containing the ids of the users that the user follows.
- full_name. It is the user's full name.
- id. It is the user's unique UUID.
- is_full_name_visible. It is true if the user allows other users to see their full name, false otherwise.
- picture_url. It is the URL where bloQo stores the user's profile picture.
- username. It is the user's username.

Limitation of Firebase's Free Tier

Our search page offers numerous filter and sorting options that users can use to find the courses they wish. However, in order to perform such complex queries, Firebase requires

manually-crafted indices. Most of such indices have been created. However, since we use Firebase's free tier of services, we are limited in the number of indices that we can create, and not all the combinations of queries are actually supported. This is notified by a warning in the application. Most of the time, removing the sorting option will make the query actually possible.

2.1.3. File Storage

When it comes to multimedia files, such as audio files, images, and videos, the application relies on the online and in-cloud storage **Firebase Storage**. In such storage, users can secure the multimedia content for their blocks and their profile pictures.

2.2. Custom Widgets

bloQo is an application with relatively little logic. In fact, its core is the community: the more users bloQo is able to draw, the more courses will be made available and the more vast will be its educational offer and its usefulness. Therefore, the biggest amount of time during development has been dedicated to custom widgets design, aiming at providing easy-to-use, powerful tools to creators, as well as compelling experiences to learners.

For this application, we created **36 new widgets**. These widgets assemble the vast majority of our application components. They fall under some categories:

- buttons (3). They are clickable buttons that vary both for aesthetic and functionality reasons.
- complex (12). They are made out of other custom widgets and often occupy a considerable portion of the view. They are often clickable and their complexity is given either by the amount of information carried or by the amount of different clickable components. Notice also that they may significantly vary based on the data that is fetched from the database.
- containers (2). They are simple widgets with aesthetic purposes that are parent to another widget.
- custom (3). They are relatively simple widgets that either offer some functionality that is currently not supported by other Flutter widgets, or wrap up existing widgets by applying some aesthetic modifications or standardizations.
- forms (3). They respond to the user's input by modifying their state. They wrap up existing Flutter widgets and provide aesthetic modifications and easier integration

with other components.

- multimedia (3). They allow users to interact with multimedia files (e.g. stopping or resuming the streaming of an audio or a video file).
- navigation (3). They allow users to easily navigate among the several pages that bloQo offers.
- notifications (3). They inform users about events that happened and that may require their attention. They can also be interacted with.
- popups (2). They are used to draw the user's attention, either to ask for an acknowledgement/confirmation or to alert that something went wrong. For this reason, they create an overlay that partially obscures everything else.
- quiz (2). They are used to test the users' understanding of some topic and react differently based on whether the given answers are correct or wrong.

2.3. Main Pages and Navigation

Before the user logs in or registers, they can only navigate between the welcome/login page and the register page. However, as the user signs in, the core pages of bloQo are loaded. In particular, we can define **5 main navigation stacks**, which are independent of one another and each corresponds to a navigation item on the bottom navigation bar:

1. **home stack**. It is the least complex stack among all the others, as it never pushes another pages on top of the first one. Indeed, the home page shows the courses the user is currently learning or creating, and when one of them is clicked, the user is redirected to either the learn or the editor stack.
2. **learn stack**. Its first page displays either the courses the user is currently learning or the courses the user has already completed learning. Thanks to two separate tabs, the two different views are never mixed up. Other relevant pages that can be accessed from this stack are the course content page, the section content page (with the blocks to learn from), and the page where users can rate a completed course.
3. **search stack**. Its first page allows user to set the filters and sorting options before querying the bloQo database, as well as to open the in-app camera and scan some QR code. Relevant pages that can be accessed from this stack are the course content page (in a version that allows enrollment) and the page that displays the profile and the published courses of a user.

4. **editor stack**. Its first page displays either the courses the user is currently creating or the courses the user has already published. Thanks to two separate tabs, the two different views are never mixed up. Other relevant pages that can be accessed from this stack are all the pages that allow course content modification (there is a dedicated page for modifying course details, chapter details, section details, and block details - one per each type of block), the course publication page and a page that allows the user to view some statistics about a published course.

5. **user stack**. Its first page displays all the information bloQo stores and may share about the user. Plus, it also shows some buttons that allow users to jump to a page where it is possible to adjust account or application settings. Finally, there's a button that allows users to log out and go back to the login page.

For this application, we built a total of **28 different pages**, the most of which significantly changes based on the data retrieved or the arguments passed. Notice that, even if the majority of those pages are exclusive to a certain stack, there are some pages (e.g. the page that shows a QR code) that can be reached from multiple stacks. Furthermore, by clicking the icon on the top-right corner of the app bar, it is possible to reach the notifications page, which does not allow to navigate between the other stacks and requires the user to go back through the specific button on the top-left margin of the app bar. However, it is possible to reach the notifications page from any stack.

2.4. Required Device Permissions for Full Functionality

Users who want to use all the functionalities of the application need to grant the application some permissions:

- access to the device's **photo/video gallery** is necessary to upload profile pictures and photo/video files.
- access to the device's **camera** is necessary to use the in-app camera to scan QR codes.
- access to the device's **file storage** is necessary to upload audio files (iOS-based devices do not need to grant this permission as it is always granted).
- access to **Internet** is necessary to interact with Firebase's authentication, database, and storage services, as well as with the iFrame Player API (iOS-based devices do not need to grant this permission as it is always granted).

Notice that write permission to the device's file storage is automatically granted both

for Android-based and iOS-based devices, and it is used to save the course completion certificate (if the user chooses to save it locally and does not share it instead).

2.5. External Services

As already mentioned in the first section, bloQo uses Firebase's authentication, database, and file storage as essential external services. Furthermore, the application also uses Google's **iFrame Player API** to play and control YouTube videos. Such API is integrated thanks to a specific plugin (which we will present in the next chapter) and to a custom widget created by us.

3 | Dependencies

In this chapter, we present the plugins we have incorporated in the application.

3.1. Plugins

The following table lists all the plugins that we have incorporated as dependencies for the application. For each plugin, we provide the reason why we decided to use it.

Plugin Name	Justification
audioplayers	It allows the creation and management of an audio channel. It is used to reproduce audio files in blocks.
back_button_interceptor	It helps intercepting the back button typical of Android devices, in order to best handle it.
cloud_firestore	It allows the safe and in-cloud storage of application data.
fake_cloud_firestore	It allows the application to interact with a mock database during integration testing.
file_picker	It allows the user to pick files from their device's storage.
firebase_auth	It allows users to safely login and register to the application.
firebase_auth_mocks	It allows the application to interact with a mock authentication service during integration testing.
firebase_core	It allows the application to connect to its Firebase app.
firebase_storage	It allows the safe and in-cloud storage of files data.
firebase_storage_mocks	It allows the application to interact with a mock file storage service during integration testing.
flutter_launcher_icons	It allows the application to generate appropriate launcher icons based on different devices.

flutter_localizations	It allows developers to easily localize the application in multiple languages.
flutter_lints	It contains a recommended set of lints for Flutter apps, packages, and plugins to encourage good coding practices.
flutter_markdown	It allows users to write and render Markdown text. It is used in text blocks.
flutter_native_splash	It allows the application to generate a splash screen to display before the first page is built.
flutter_phoenix	It allows the application to easily destroy any saved state when logging out and go back to the first page.
flutter_rating_bar	It allows users to easily rate a course by tapping the desired amount of stars.
flutter_test	It allows developers to easily write widget and unit tests.
image_picker	It allows users to open their photo gallery and pick a file.
image_cropper	It allows users to crop their profile pictures to match the aspect ratio required by the application.
integration_test	It helps developers write useful integration tests.
intl	It allows developers to easily localize the application in multiple languages.
loader_overlay	It allows the application to prevent users from interacting with the application when relevant data are being updated to or fetched from the database.
loading_animation_widget	It is a package that contains many loading animations.
path_provider	It allows the application to save a course certificate to the device's storage.
percent_indicator	It allows users to visually understand the progress they have reached with respect to a course they are learning.
permission_handler	It allows users to give the application the necessary permissions to access several device resources.

provider	It allows the application pages to effectively share some state during the whole application lifetime, and notify some pages or widgets when some changes happen.
qr_code_scanner	It allows users to scan QR codes via the in-app camera.
qr_flutter	It allows users to view QR codes they want other users to scan.
share_plus	It allows users to save or share the course completion certificates they generate.
shared_preferences	It allows users who have already logged in to the application to skip further logins by saving the credentials on the application's cache files.
uuid	It allows the application to generate unique identifiers for the newly created documents.
video_player	It allows users to interact (e.g. play and pause) the videos they have loaded in their blocks.
youtube_player_flutter	It allows users to interact (e.g. play and pause) the videos they have embedded from YouTube in their blocks.

Table 3.1: Plugin Dependencies

Notice that these plugins have transitive dependencies that are required but not listed here since they are not directly related to the application.

4 | User Interface

In this chapter, we show and briefly discuss the user interfaces we produced for the application.

4.1. User Interface Design

Before starting the implementation of the application, we carefully designed the user interface of both pages and widgets in a FlutterFlow project. When such process was completed, we decided to use that work as a reference when actually implementing pages and widgets. This decision came from the fact that many of the libraries that FlutterFlow uses are proprietary, so we decided to build our components from scratch; in this way, we believe that we have also learned more from this project.

The following images will display the default "Purple orchid" theme. At the end of the chapter, a sample page with the other "Ocean cornflower" theme will be displayed, so that the readers can see the color palette of the alternative theme we made available. However, it is worth noticing that we paid effort to make it extremely easy to add new themes: in fact, building a theme only requires extending an abstract class and implementing its required attributes.

4.2. Mobile and Tablet Interfaces

We are about to show the mobile and tablet version of the interfaces of every page we created in such a way that we also show almost all the widgets we created in all the possible layouts.

4.2.1. Welcome Page

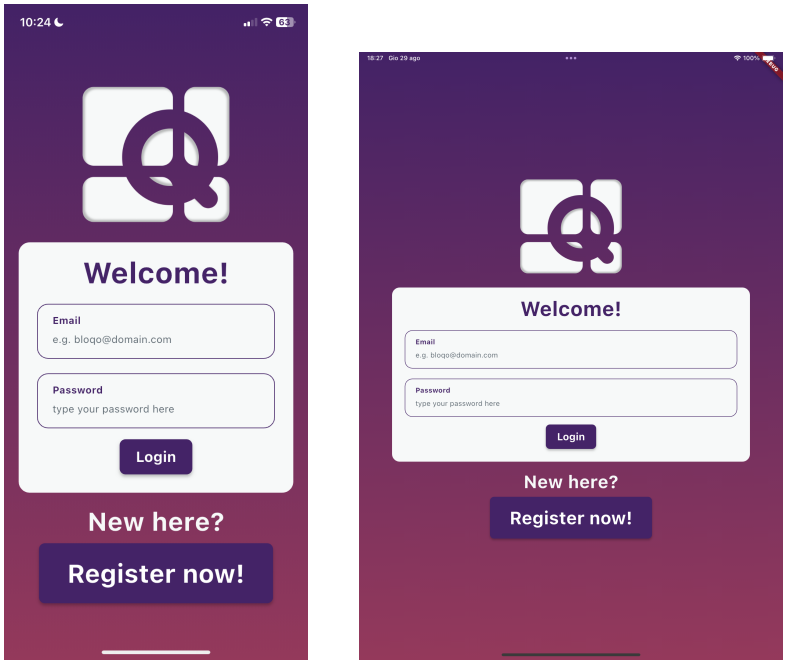

Figure 4.1: Welcome Page

4.2.2. Register Page

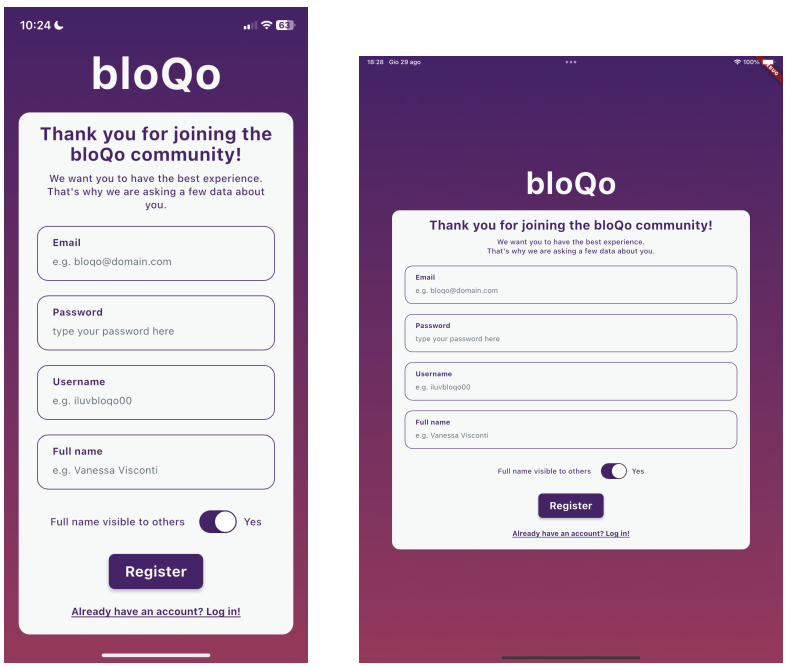

Figure 4.2: Register Page

4.2.3. Home Page

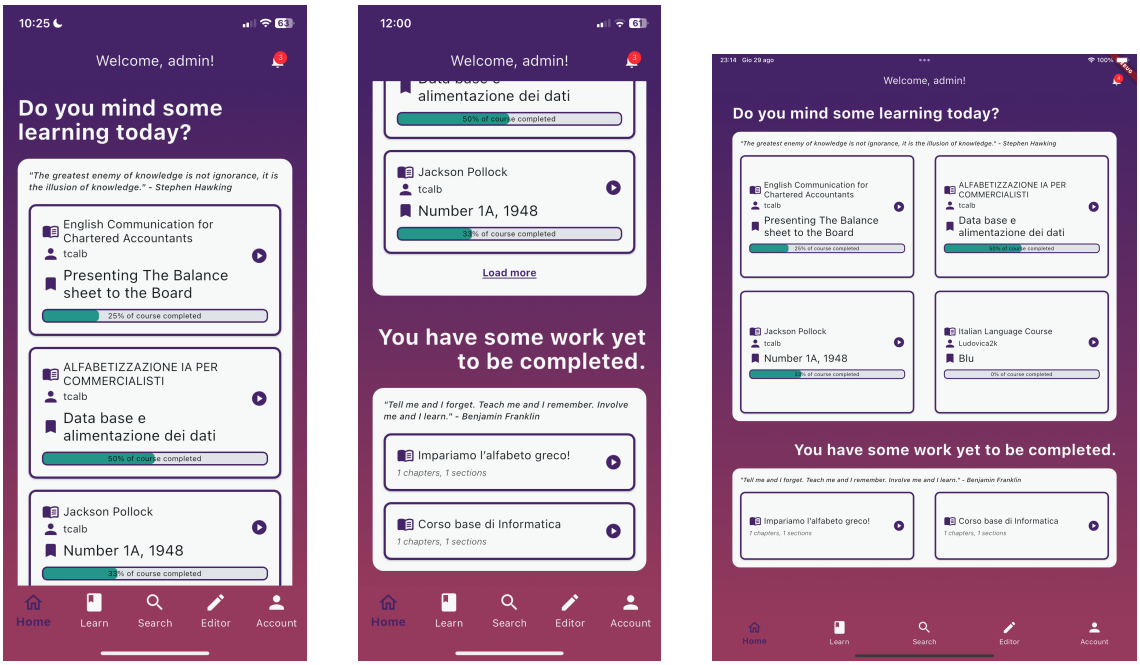

Figure 4.3: Home Page

4.2.4. Learn Page

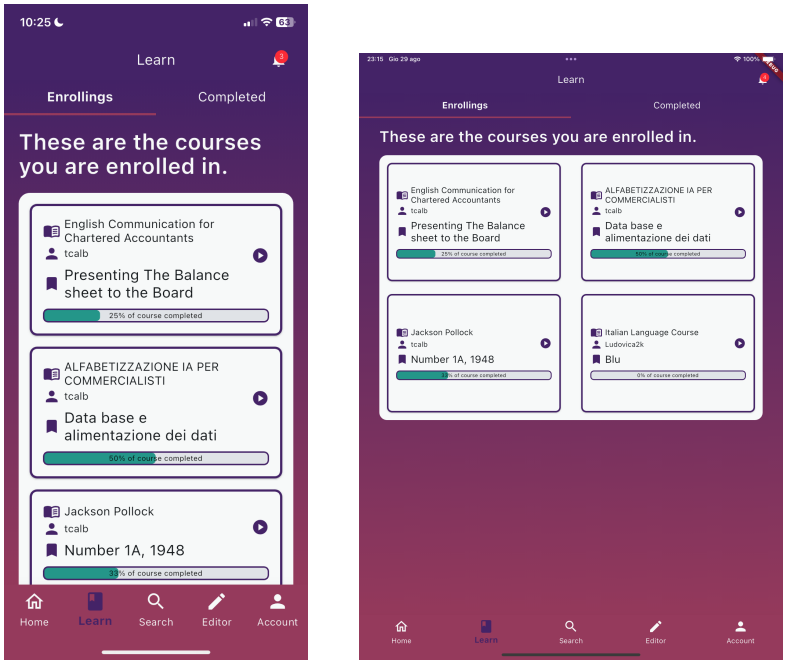

Figure 4.4: Learn Page Enrollings

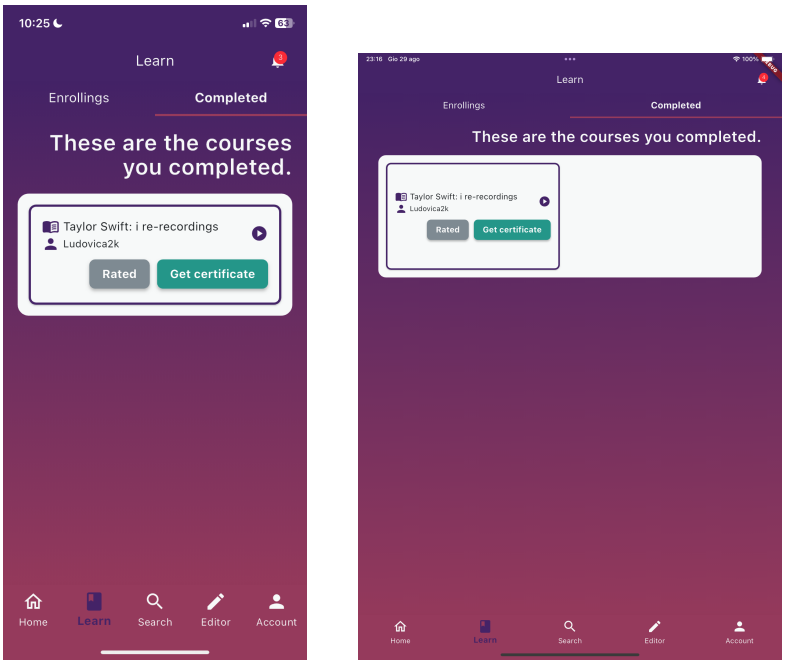

Figure 4.5: Learn Page Completed

4.2.5. Search Page

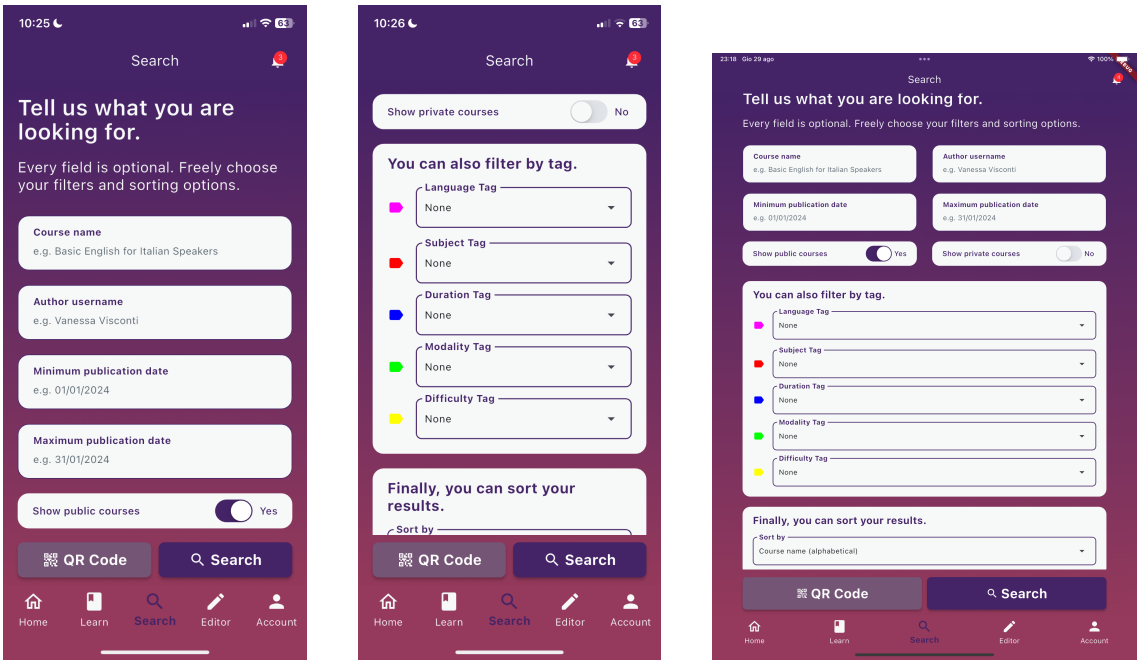

Figure 4.6: Search Page

4.2.6. Editor Page

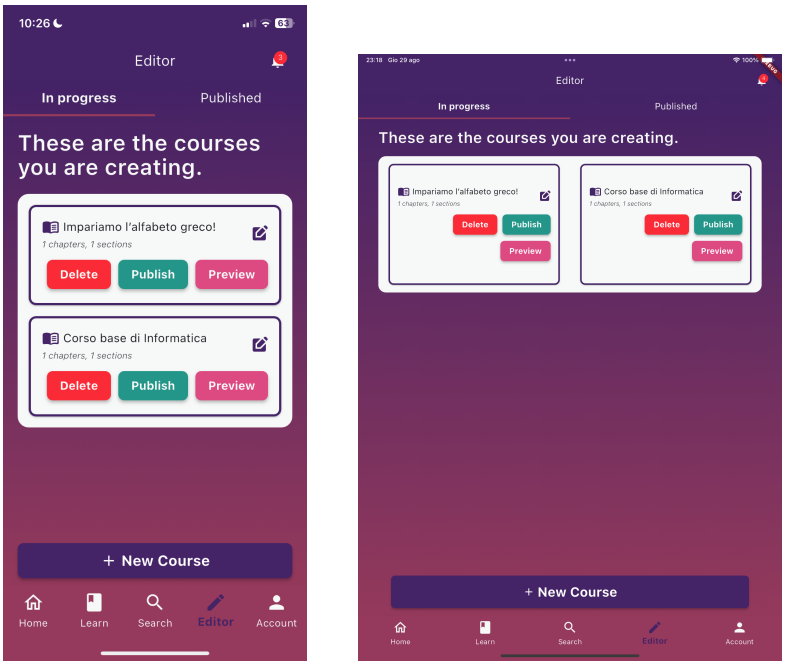

Figure 4.7: Editor Page In Progress

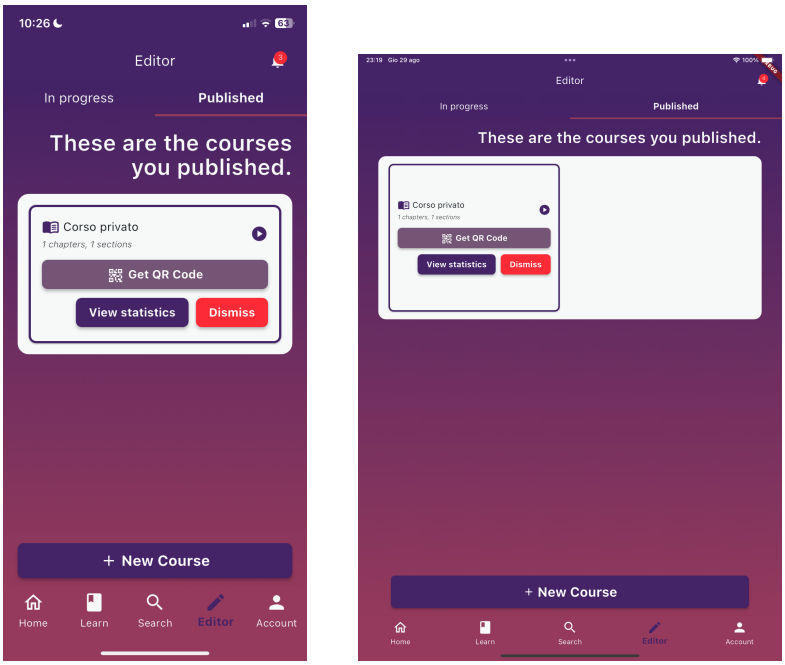

Figure 4.8: Editor Page Published

4.2.7. Account Page

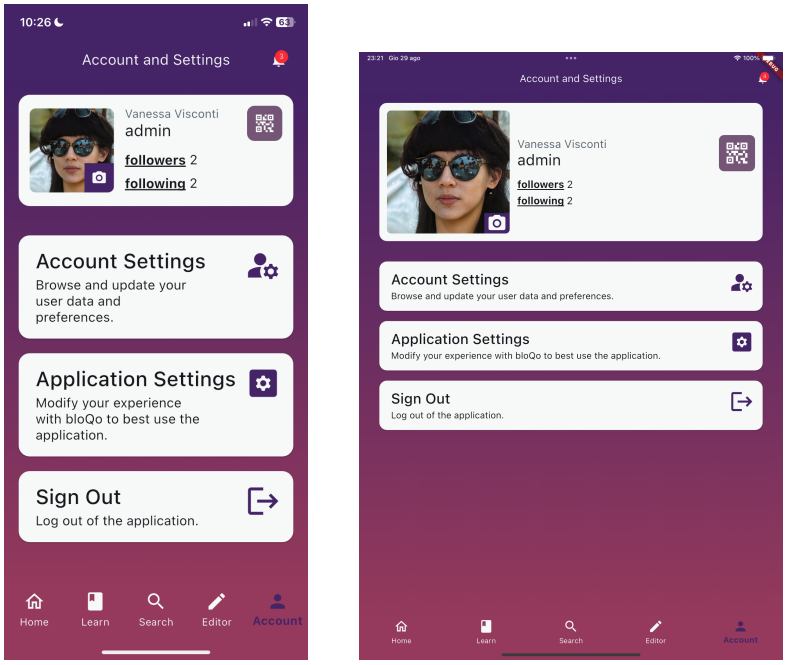

Figure 4.9: Account Page

4.2.8. Notifications Page

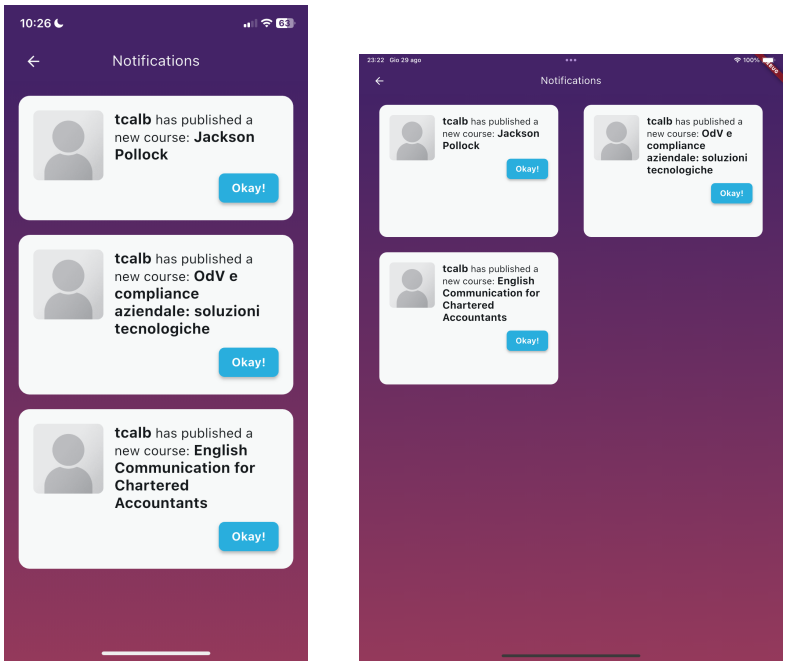

Figure 4.10: Notifications Page

4.2.9. Course Content Page

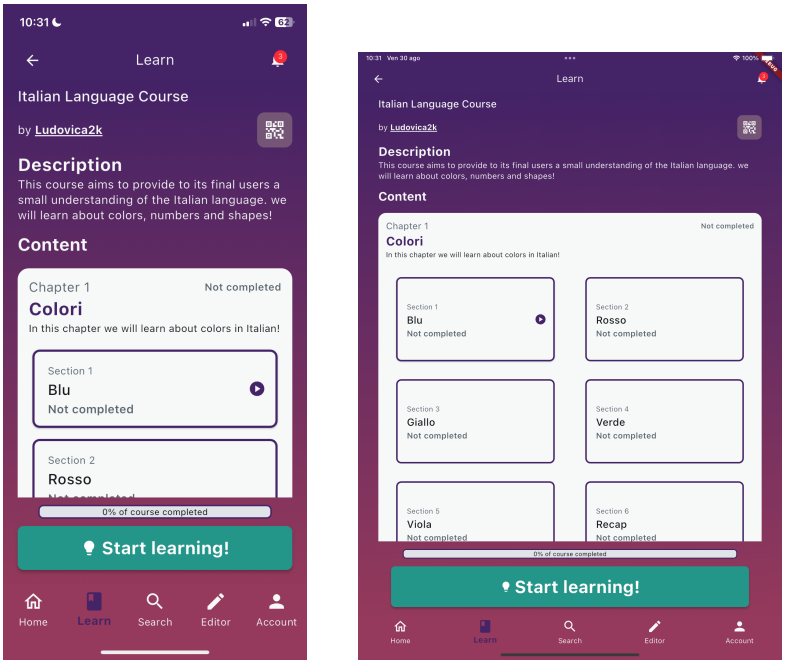

Figure 4.11: Course Content Page

4.2.10. Section Page

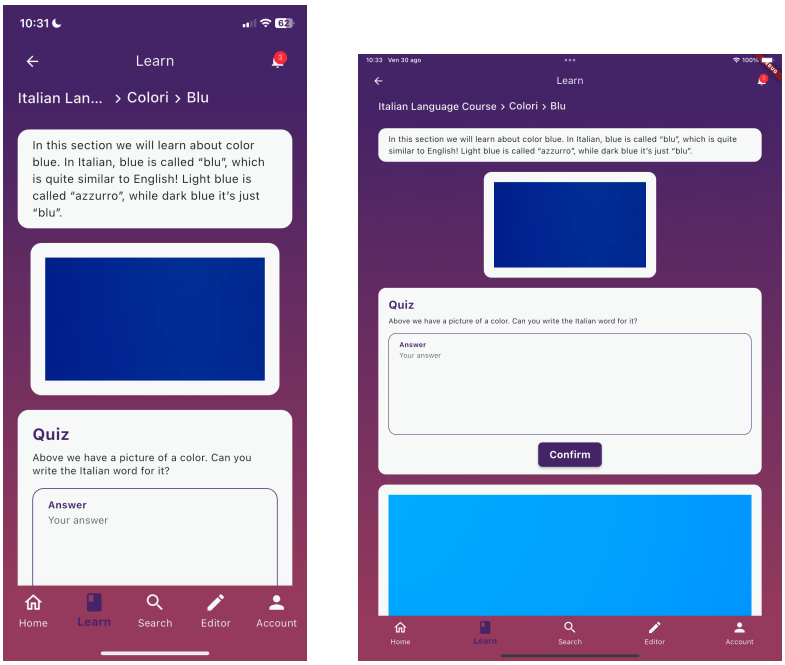

Figure 4.12: Section Page

4.2.11. Review Page

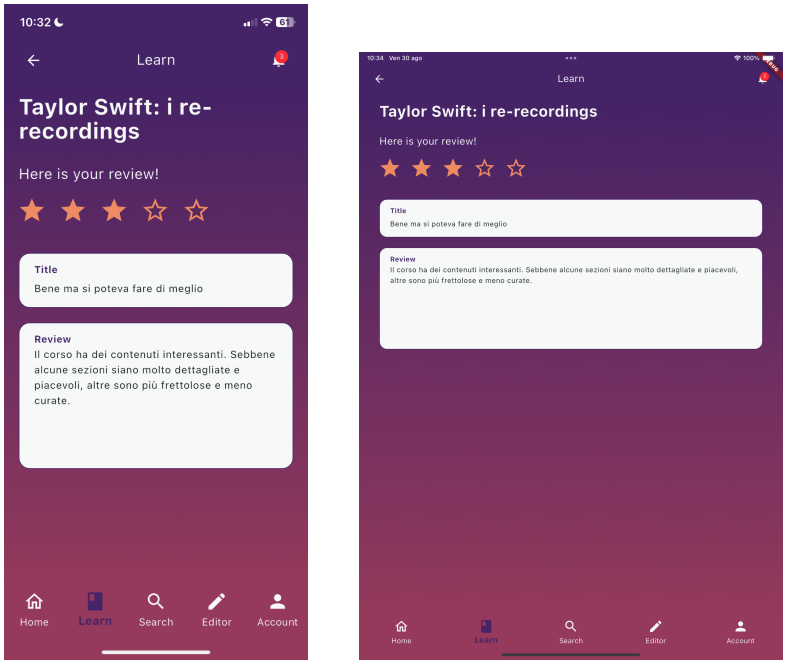

Figure 4.13: Review Page

4.2.12. Search Results Page

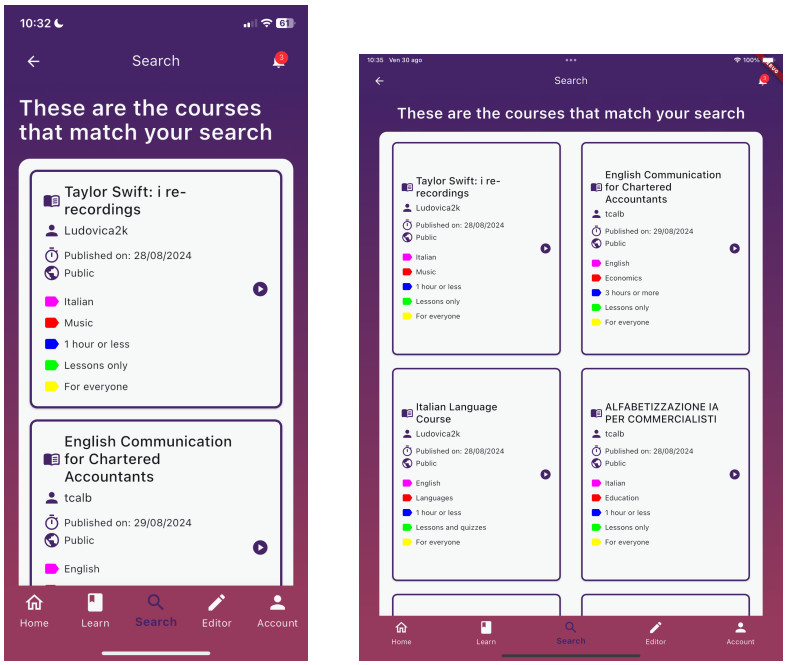

Figure 4.14: Search Results Page

4.2.13. Course Search Page

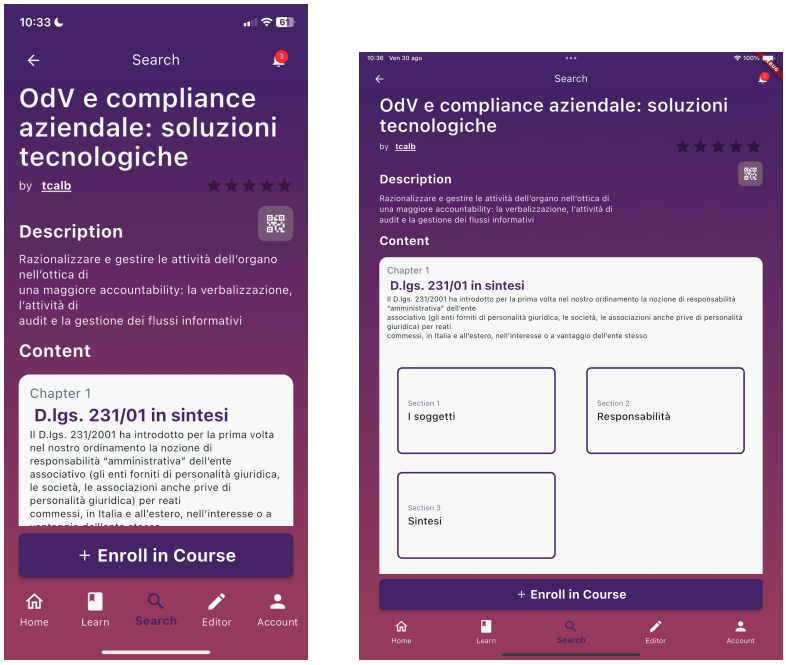

Figure 4.15: Course Search Page

4.2.14. QR Code Scan Page

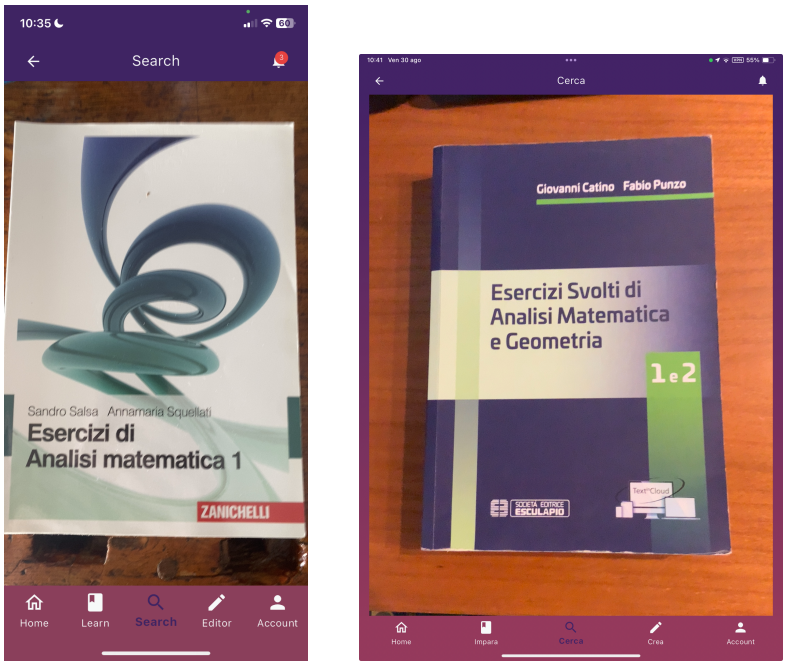

Figure 4.16: QR Code Scan Page

4.2.15. Edit Course Page

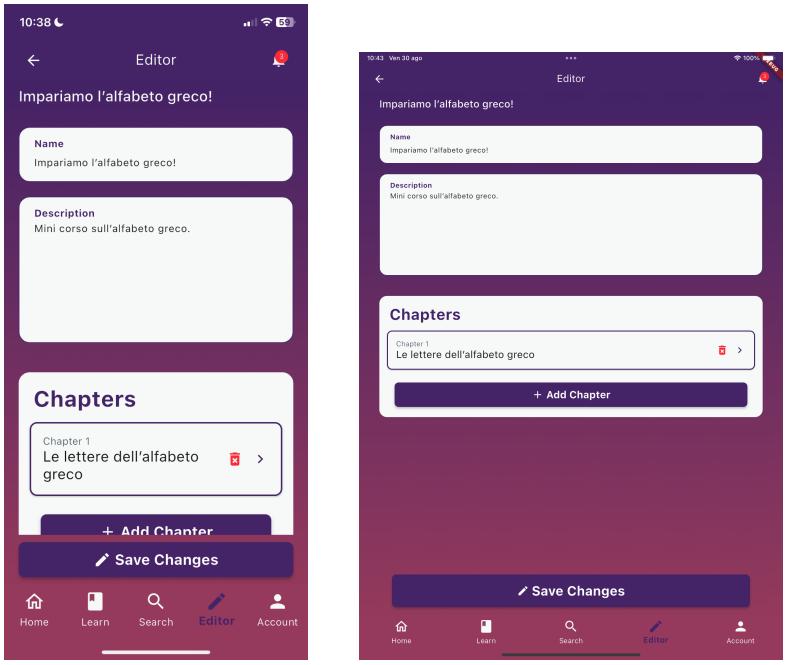

Figure 4.17: Edit Course Page

4.2.16. Edit Chapter Page

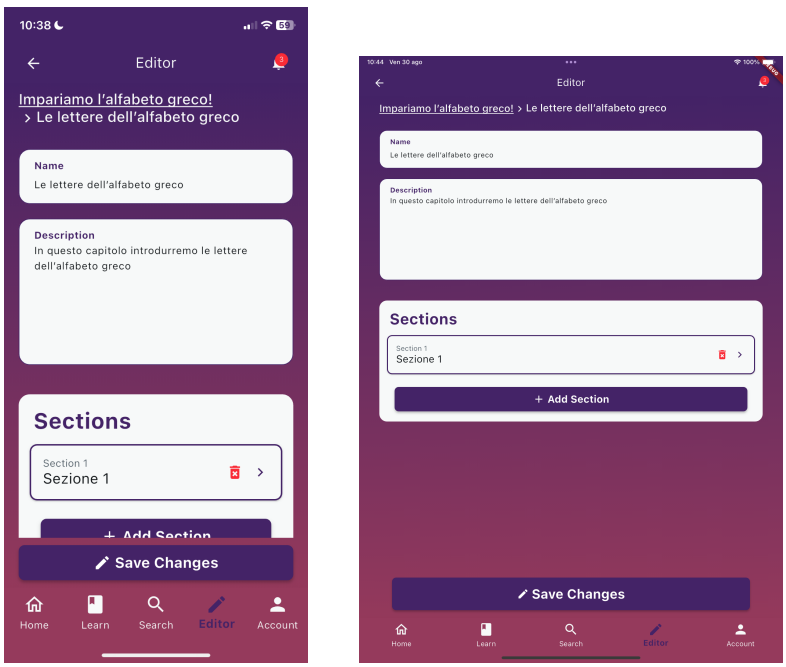

Figure 4.18: Edit Chapter Page

4.2.17. Edit Section Page

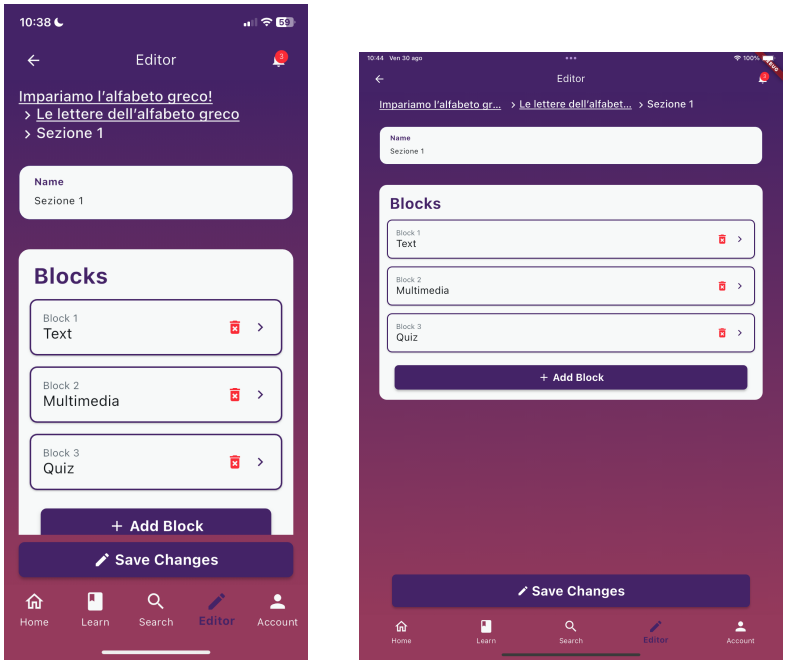

Figure 4.19: Edit Section Page

4.2.18. Edit Text Block Page

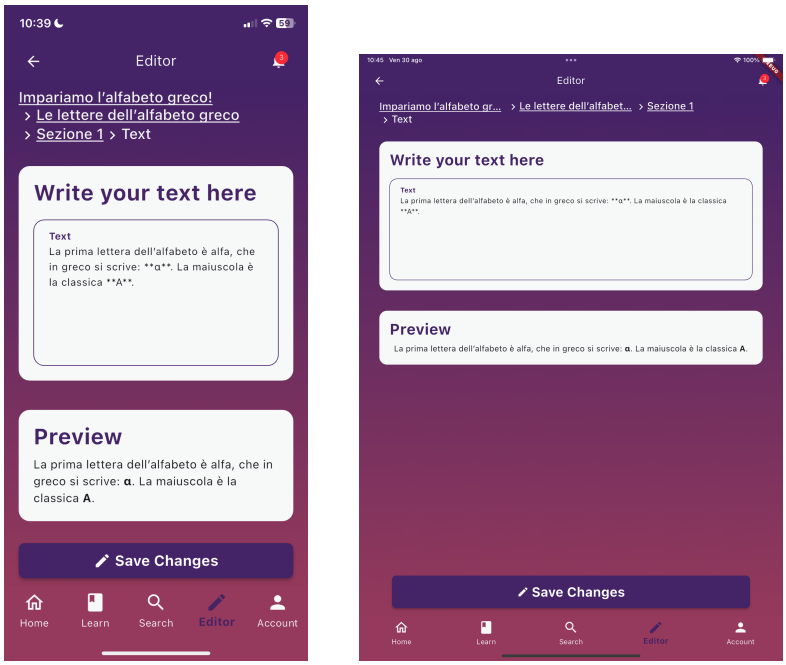

Figure 4.20: Edit Text Block Page

4.2.19. Edit Multimedia Block Page

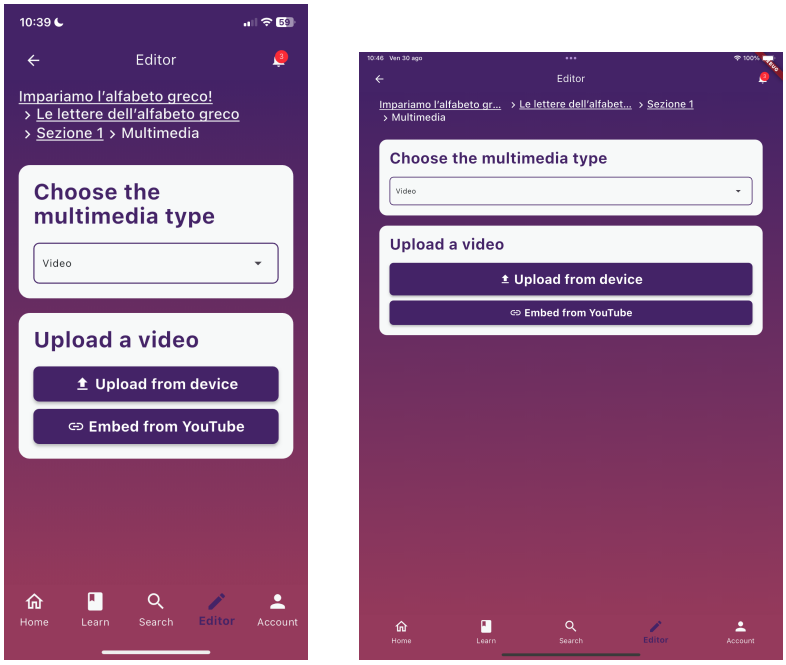

Figure 4.21: Edit Multimedia Block Page

4.2.20. Edit Quiz Block Page

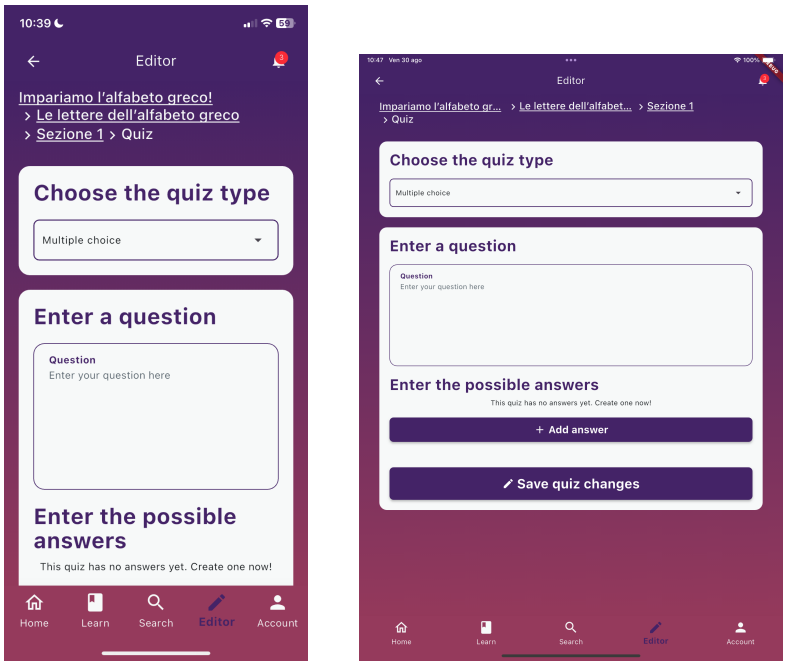

Figure 4.22: Edit Quiz Block Page

4.2.21. Course Content Preview Page

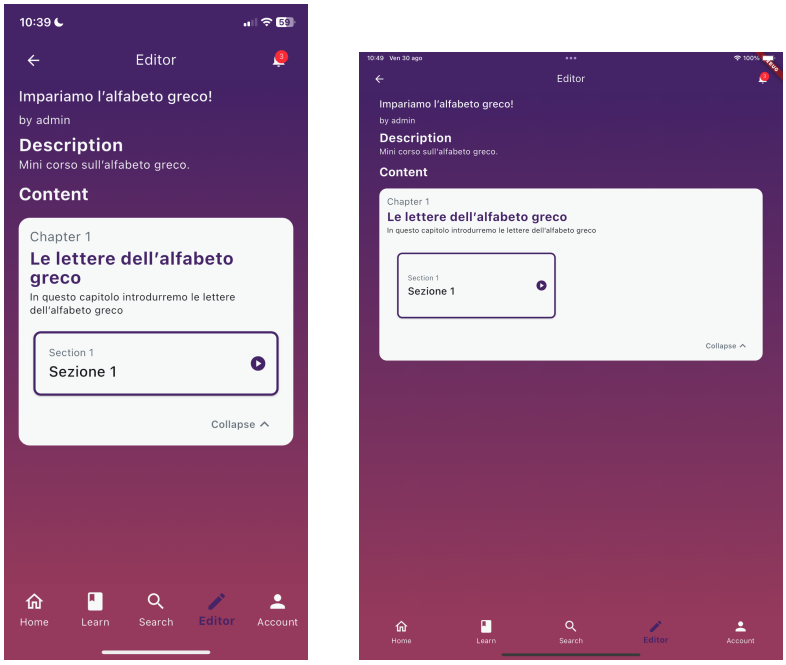

Figure 4.23: Course Content Preview Page

4.2.22. Section Preview Page

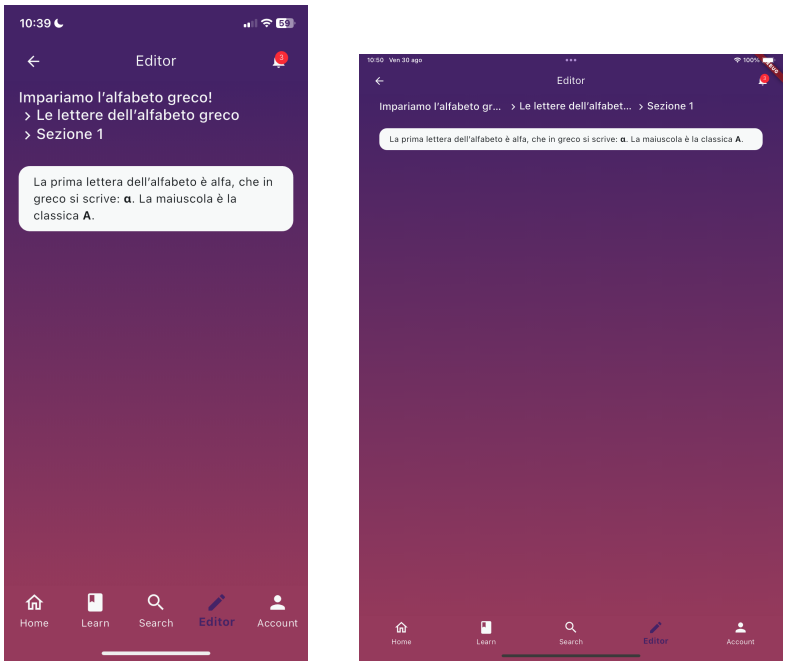

Figure 4.24: Section Preview Page

4.2.23. Publish Course Page

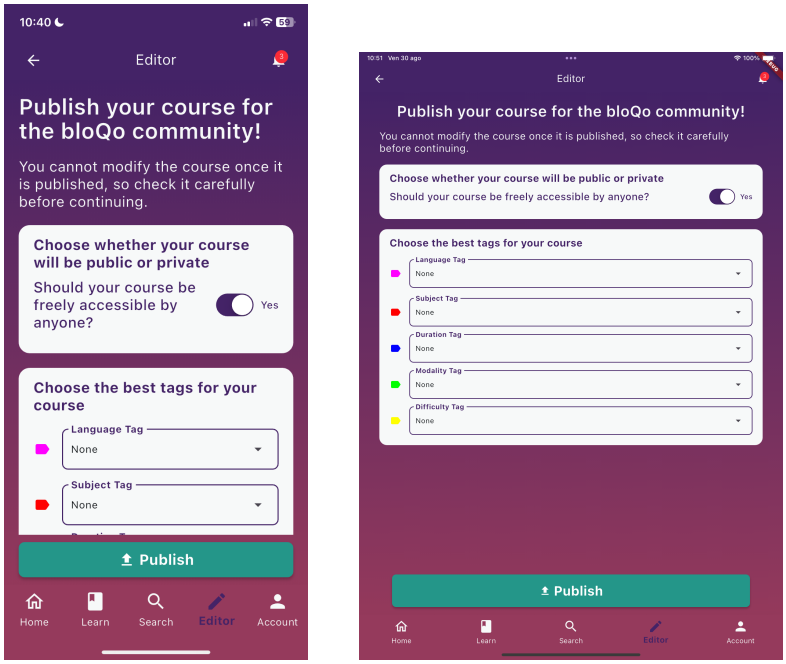

Figure 4.25: Publish Course Page

4.2.24. View Statistics Page

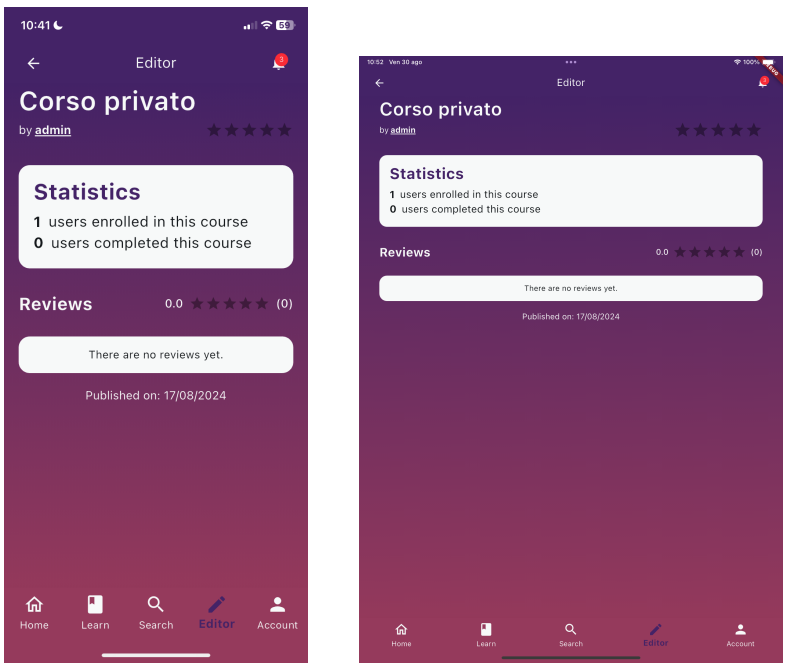

Figure 4.26: View Statistics Page

4.2.25. Settings Page

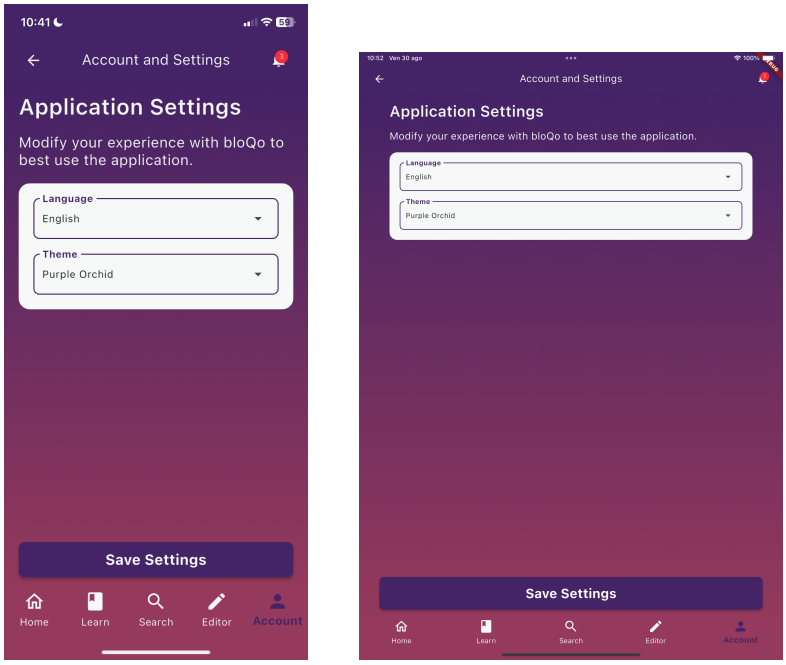

Figure 4.27: Settings Page

4.2.26. QR Code Page

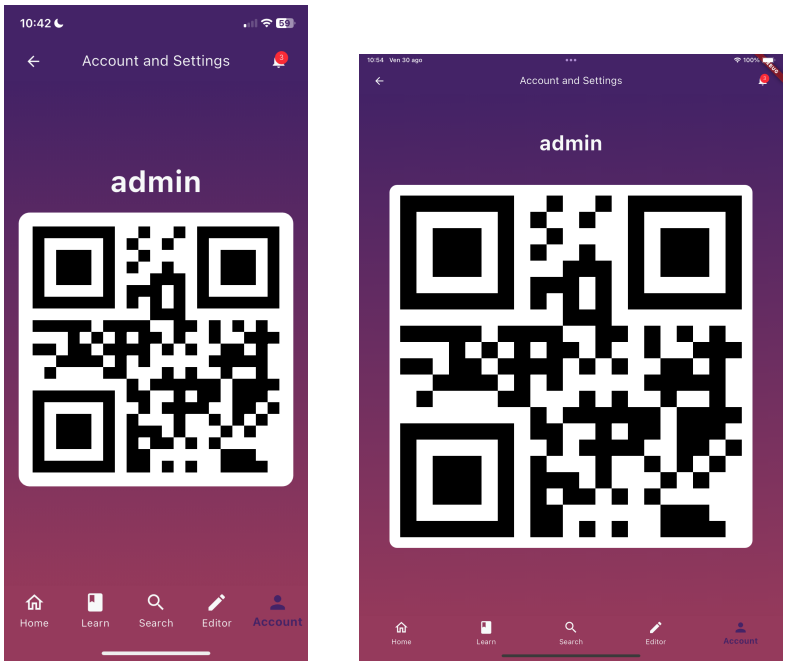

Figure 4.28: QR Code Page

4.2.27. User List Page

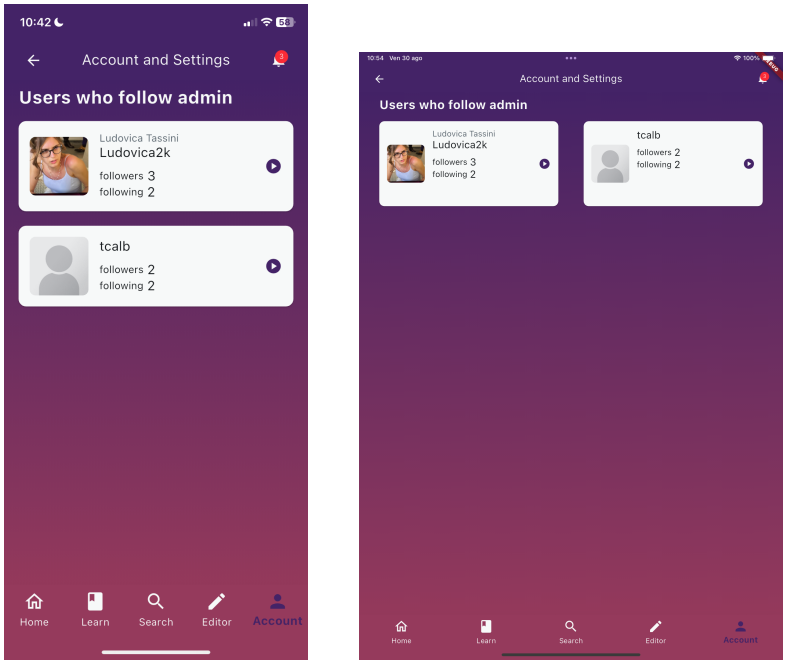

Figure 4.29: User List Page

4.2.28. User Profile Page

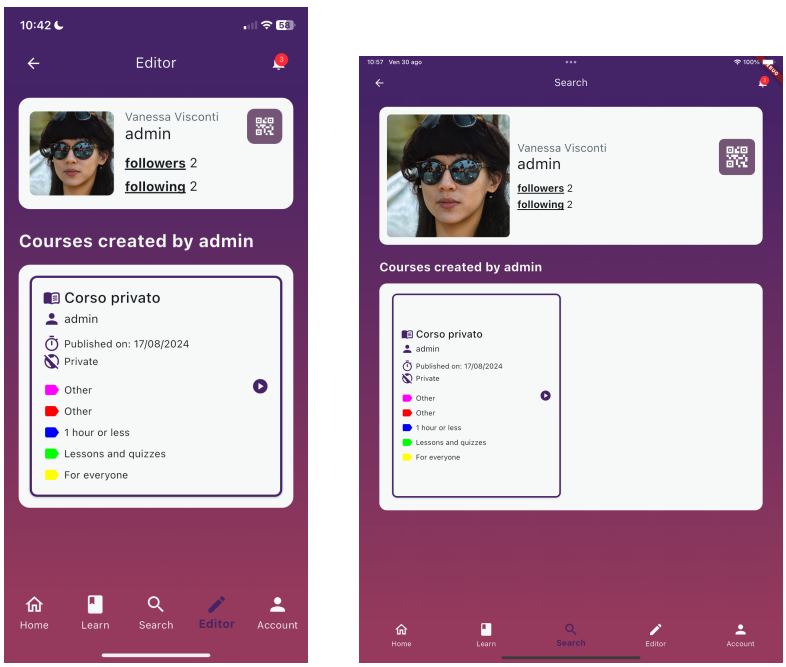

Figure 4.30: User Profile Page

4.3. Sample Page with Different Themes

We now show the Account page with the two different themes currently available: *Purple orchid* and *Ocean cornflower*.

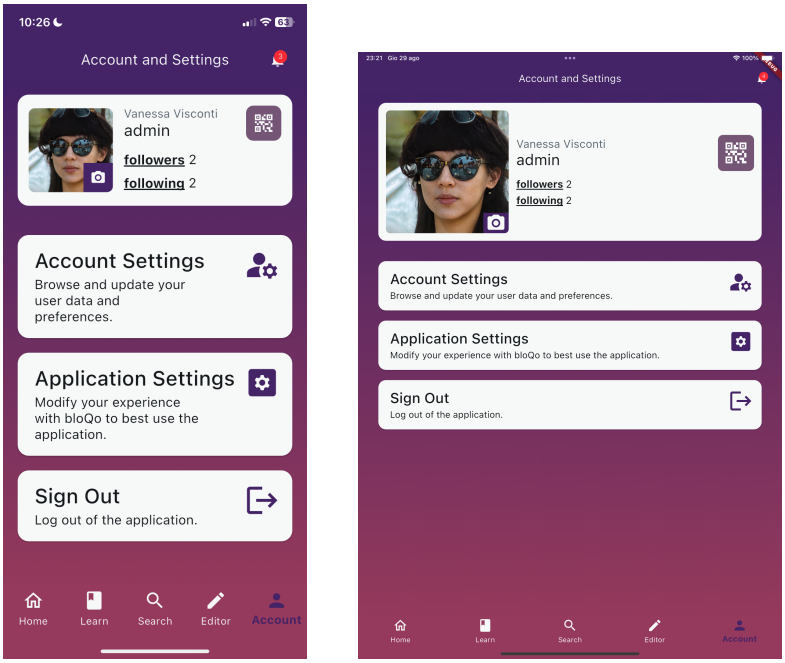

Figure 4.31: Account Page with Purple orchid theme

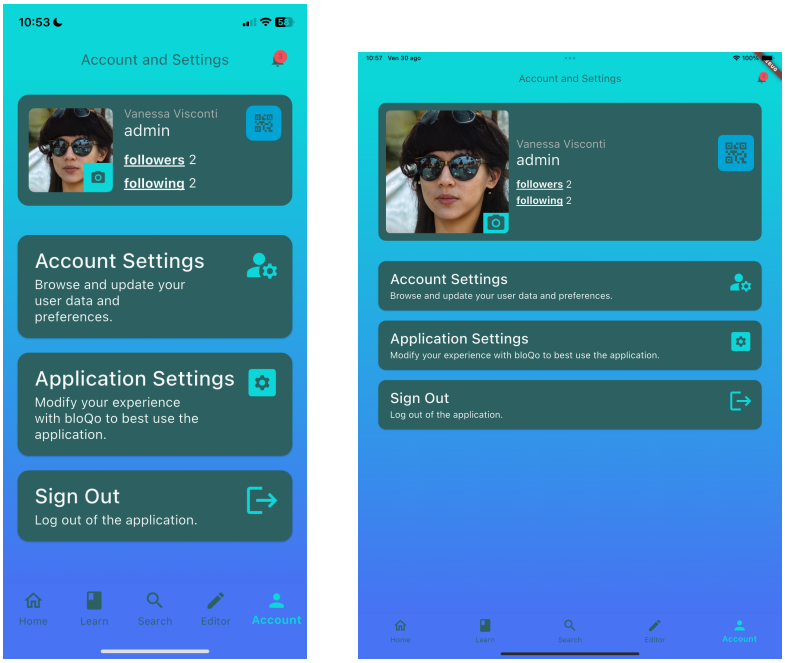

Figure 4.32: Account Page with Ocean cornflower theme

4.4. User Experience Flow Diagram

We now show a diagram that describes the actions required to navigate through the pages we have just presented. Notice that, due to the high complexity of the interaction scenarios, we will not show all the possible interactions but just the main ones, still covering all the application pages.

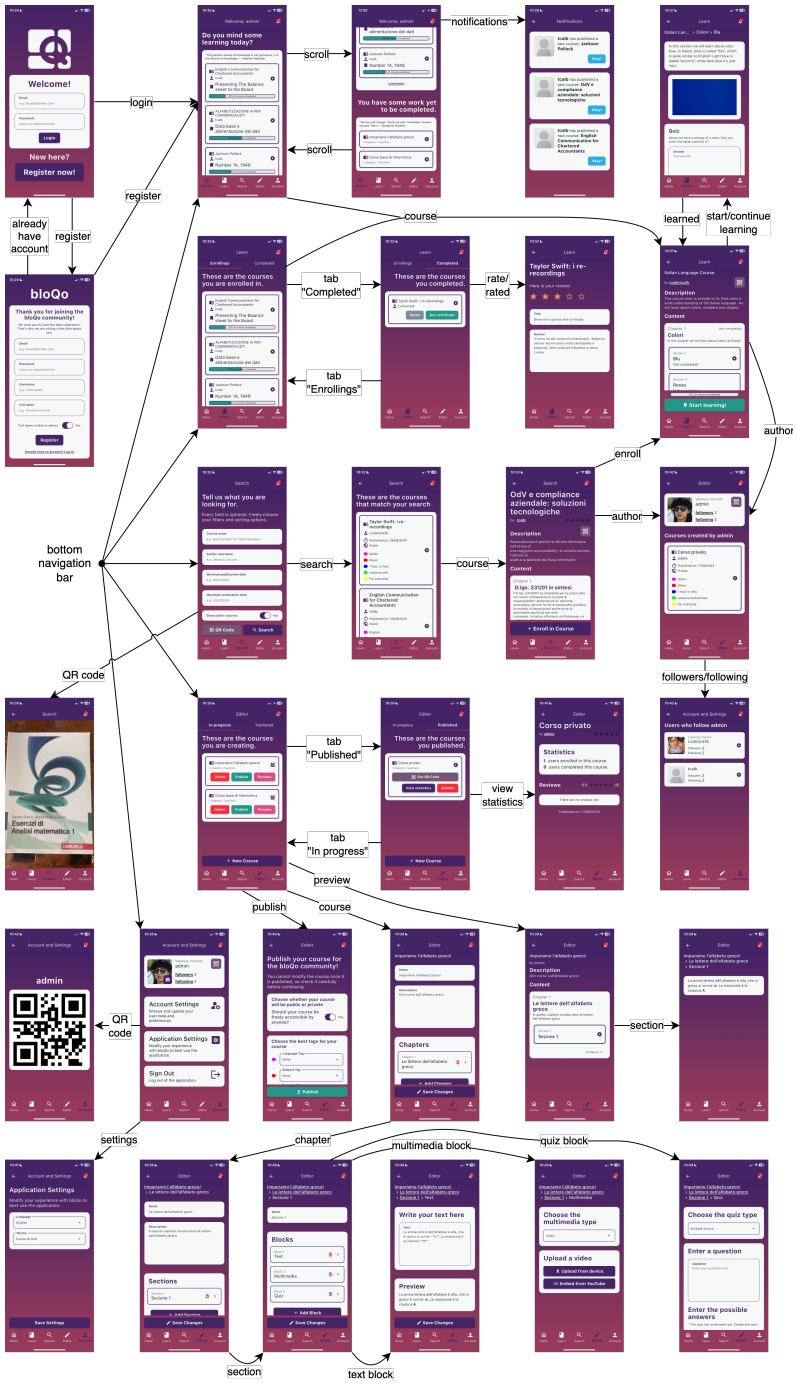

Figure 4.33: UX Flow Diagram

5 | Application Testing

In this chapter, we detail the numerous testing phases that the application went through.

5.1. Unit Testing

Unit testing is a kind of testing that aims to verify that the application logic (in the form of a method or a class) works as expected. As already mentioned in the previous chapters, bloQo is not an application with relevant parts of logic that require testing outside of the widget context. Therefore, our efforts with respect to unit testing are limited to testing the text validation functions that we wrote for various forms. Since those were the only logic parts we could focus on, we wrote a thorough group of unit tests, testing every possible case.

We wrote a total of **67 unit tests**. In particular:

- 4 tests are dedicated to email validation;
- 52 tests are dedicated to password validation;
- 3 tests are dedicated to username validation;
- 4 tests are dedicated to full name validation;
- 2 tests are dedicated to quiz question validation;
- 2 tests are dedicated to quiz answer validation.

Notice that quiz question / answer validation is needed to prevent unwanted data manipulation (due to the way their data is stored). Additionally, in the source code, each unit test has a unique description that helps to easily understand what each test is specifically testing.

5.2. Widget Testing

Widget testing is a kind of testing that aims to verify that a widget looks and interacts as expected. This kind of testing is crucial for bloQo's goal, therefore we wrote a group of **192 widget tests** that thoroughly test all the 36 custom widgets we created for our application. It would not be practical to describe individually each test. Furthermore, each widget test in the source code has a unique description if one wants to know more about what was specifically tested.

Based on the widget category, we wrote:

- 8 button tests, ensuring that buttons are correctly displayed and clickable. Furthermore, we ensured that our custom popup menu button correctly opens its menu and presents clickable choices.
- 58 complex tests, ensuring that every interactable element of every complex component is clickable correctly.
- 2 containers tests, simply ensuring that the custom containers are correctly shown and raise no exceptions.
- 7 custom tests, ensuring that the widgets correctly show up the special information they carry (e.g. progress bar) or they correctly react to specific user input (e.g. rating bar).
- 74 forms tests, especially ensuring that the validation mechanisms tested with the unit tests also eventually produced an alert effect on the text fields and that switches and dropdowns correctly respond to user input.
- 3 multimedia tests, simply ensuring that the custom widgets are correctly shown and raise no exceptions. Due to the fact that widgets, in such a testing environment, cannot have access to multimedia channels, no further tests of this kind can be performed.
- 8 navigation tests, ensuring that the bottom navigation bar, the app bar, and the course breadcrumbs allow to navigate the application pages as expected.
- 3 notification tests, simply ensuring that the custom widgets are correctly shown and raise no exceptions.
- 15 pop-up tests, ensuring that alerts correctly close (and eventually trigger something) when one action is pressed.

- 14 quiz tests, ensuring that users can interact with them and that quizzes correctly react to users asking for an answer check.

5.3. Integration Testing

Integration testing is a kind of testing that aims to verify that all widgets and services work together as expected. They are carried out as if there were a real tester, meaning that the tests make the "virtual tester" perform all the gestures that a real-world user would do to get the same result. For this reason, mock database, authentication, and storage services are provided and initialized with made-up data. These tests are extremely important and meaningful. In addition, due to their complexity, they require more time to execute. Notice that, since the goal of this testing phase is not to test how the application looks like, sometimes we extended the render area size to avoid useless scrolls.

We wrote a total of **91 integration tests**. Each integration test, in the source code, comes with a unique description in case one wants to know exactly what every single test specifically tests. In general, we can say that all the tests that we wrote mimic one different real use scenario each. Lastly, notice that we designed the integration tests in such a way that all possible application use cases are covered.

Now we discuss our integration tests based on the categories they have been grouped into:

- main tests (72). These integration tests focus on the main use case scenarios, such as creating, publishing, searching for, and learning from courses. They also focus on testing all the interaction possibilities that all the widgets encountered along the way offer. These tests are further grouped by main stack, each group testing the functionalities and widgets of that specific stack. There are:
  - 8 account tests, ensuring that users can browse their profile information, change account and application settings, and logout.
  - 28 editor tests, ensuring that users can create, modify, preview, delete, and publish courses. In addition, they ensure that once a course is published, it can be dismissed, accessed, or its statistics and QR code can be viewed.
  - 2 home tests, ensuring that the home page shortcuts work as expected.
  - 20 learn tests, ensuring that users can actually learn the course content, respecting the section order, and access various information such as the course author's profile and the course's QR code. In addition, they ensure that once a course is completed, users can rate it and get a certificate.

  - 1 notification test, ensuring that users can correctly see and interact with notifications.
  - 13 search tests, ensuring that users can correctly search for courses and enroll in them, as well as view course authors' profile pages, and follow or unfollow them.

- navigation tests (11). These integration tests focus on making sure that all the navigation options that are offered to users actually lead to the intended places. In particular, navigation tests focus on the functionalities of the bottom navigation bar, the app bar, the course breadcrumbs, and the navigation between the welcome / log-in and the registration page.
- welcome tests (8). These integration tests are focused on ensuring that users can register and log in. They also ensure that incorrect credentials or badly formatted data produce the expected error alerts.

5.4. Automated Testing Summary with Coverage Report

The following table summarizes some statistics on the testing campaign that we conducted:

Testing Type	Total Number of Tests
Unit Testing	67
Widget Testing	192
Integration Testing	91
Total	**350**

Table 5.1: Automated Testing Summary

Our test coverage report states that **77% of the total lines of code** have been tested with the automated testing campaign. This is an impressive number, considering that a substantial portion of uncovered code cannot be tested with automated testing, including:

- code that asks for device permissions;
- code that interacts with the device's multimedia channels (audio and camera) and persistent storage;
- code that uploads multimedia files from the device.

5.5. User Testing

During all the development phases, the application underwent extensive user testing on both simulated and real devices. This allowed us to fix problems and bugs on a day-to-day basis. As already mentioned, we use iOS-based devices, so they have been more thoroughly tested. However, thanks to simulated devices, colleagues and friends, we have been able to collect valuable feedback also from Android users, which led to more bug fixes.

6 | Effort Spent

In this chapter, we present the estimated time we have spent working on this project.

6.1. Breakdown of Estimated Workload

The following table sums up the estimated time effort for all the design and development phases based on data we collected:

Phase	Matteo Spreafico	Ludovica Tassini
Interface design	50h	40h
Architecture and Database design	10h	20h
Implementation	180h	180h
Documentation	10h	10h
Presentation	2h	2h
Total	**252h**	**252h**

Table 6.1: Total Effort Spent (in hours)

List of Figures

List of Tables