Mobile Programming and Multimedia
A.y. 2022-2023

EduKid
Project report

Filippo Brugnolaro, n. 2087666
Marta Greggio, n. 2096579

Master's degree in Computer Science

Index

1 Introduction **3**

1.1 Description 3

2 Flutter framework **3**

2.1 Classification according to Raj and Toleti 4

2.2 Advantages 4

3 Project Architecture **5**

3.1 Clean Architecture 5

3.1.1 Goals 5

3.1.2 Idea 6

3.2 BLoC 7

4 Edukid application **8**

4.1 Structure of application 8

4.1.1 Login 8

4.1.2 Wizard 10

4.1.3 Homepage 11

4.1.4 Profile 13

4.1.5 Statistics 14

4.1.6 How to Play 16

4.2 General design choices 17

4.2.1 Device orientation 17

4.2.2 Colors 17

4.2.3 Content's size 17

4.2.4 Images 19

4.3 Testing 19

4.4 Extensions 20

4.4.1 Push notifications 20

4.4.2 Statistics scaling 20

4.4.3 Category's leaderboard 20

4.4.4 Virtual rooms 21

5 Default configuration **22**

5.1 Accounts 22
5.2 Database 22

1 Introduction

1.1 Description

This project has been developed for the *Mobile programming and multimedia* course and it consists in the development of a quiz application.
This app is available both for *iOS* and *Android* platforms, since it has been implemented with a cross-platform framework combined with *Google Firebase*.
The aim of our application is to provide a tool both for teachers and kids to learn primary school's subjects.

2 Flutter framework

Flutter is an open-source framework developed by Google and released in 2017 for building high-performance, cross-platform applications. It allows to **write code once** and **deploy it on multiple platforms**, including mobile (both *iOS* and *Android*), web, and desktop.
It uses a widget-based architecture to create appealing user interfaces, and its native-like performance and extensive widget library make it a popular choice.
Let's see its architecture:

- **Dart**: dart is a programming language developed by Google. It is executed just in time thanks to the *Dart Virtual Machine*.
  One of the main advantages is the hot reload, that allow the developer to immediately see the modifications of the *UI*, while developing.
  This feature has been widely used during the development of our application, since we focused a lot on making the *UI* appealing and responsive;
- **Flutter engine**: it is written in C++ and it supports the rendering of the application;
- **Foundation library**: it is written in Dart and it makes available many functions and classes to create and manipulate graphical elements in the application;
- **Widgets**: there are two different types of widget, one for each operating system. *Material Design* widgets are used for Android while *Cupertino* widgets for *iOS*. In *Flutter*, everything is a widget.

2.1 Classification according to Raj and Toleti

Cross-platform frameworks for mobile development like *Flutter* reduce the negative effects of market fragmentation. They allow the distribution of the application in several stores and the app is developed on time using just one programming language. In particular, *Flutter* follows a **cross compiled approach**: this ensures native look and feel for different operating systems, good performances and fast development.

2.2 Advantages

In this section there is a list with some of the advantages of *Flutter*:

- **Hot reload**: as already stated before, this feature allows to build and reload the code during run-time making the modification immediately visible on the emulator.
- **Fast design**: *Flutter* allows to create appealing *UI* in a fast and easy way. This makes the framework very suitable for the development of demo application and MVP (Minimum Viable Product).
- **Support**: *Flutter* is being continuously improved not only by Google but also by many other developers. This means that the community is active and it is very easy to find support, both on blogs like *Stack Overflow* or even in the official page of the framework.
  Moreover, *Flutter* interfaces really well with other Google products such as *Google Firebase*, which we used to get a real-time database and authentication APIs ready for our project.

3 Project Architecture

After choosing the language, it was important to decide how to structure the code in order to favor decoupling and independence between *business logic* and *presentation logic*. So the *Clean Architecture* has been used to achieve this goal and the *BLoC pattern* has been implemented only in two features because of lack of time.

In the next sections, it will be briefly presented how the architecture and the pattern work and why they have been chosen. The description will be from an internal to an external point of view.

3.1 Clean Architecture

Clean architecture is a software design principle which has the goal to give an idea of organizing and structuring code to make it maintainable and scalable. The primary goal of *Clean Architecture* is to separate concerns and dependencies, making the base of the code more flexible and adaptable to changes.

3.1.1 Goals

Some of the main goals of this approach can be summarized in these points:

- **Independency from frameworks**: this makes the software more portable and adaptable to changes in the underlying technology;
- **Independency from *UI***: this makes the software more reusable and adaptable to the changes of *UI*;
- **Independency from database**: this makes the software more portable and adaptable to changes in the underlying database technology;
- **Independency from external agencies** (cloud providers or message brokers): this makes the software more portable and adaptable to changes in the underlying infrastructure.

3.1.2 Idea

Clean Architecture consists of several layers, each with its own specific responsibilities and dependencies.
Here there is a graphical representation of the architecture (*Figure* 1).

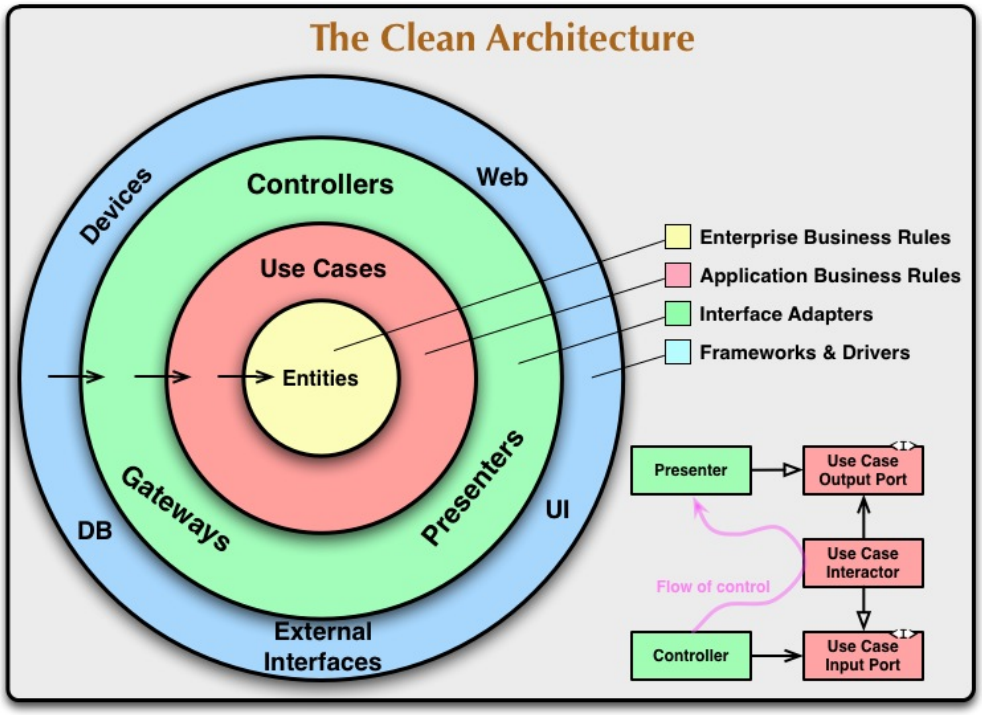


Figure 1: Clean Architecture

The key idea of this approach is to encourage a clear separation of concerns, having the most important and stable *business logic* at the center and the details at the side. By looking at the code of the application, it is clear to see that Use Cases have not been used. Still the goals of this approach seems to be achieved. In fact it is possible to adapt the idea based on the needs of the project.

The layers which have been used are described as follow:

- **Data layer**: it consists of data source code such as access to the *Firebase database* or other sources;
- **Domain layer**: it contains the core business logic of the application such as entities and it is independent of any other layer (*frameworks*, *UI* or *databases*);
- **Presentation layer**: it consists of the code to access the data of the app from a repository. There is also the code for *BLoC state management*.

It is important to underline that *Service Locator* pattern has been implemented using *GetIt* in order to decrease the number of dependencies. In this way the application result more loosely coupled.

Then the project is organized in a way such that each feature is completely independent from the others.

There are 2 different main folders:

- **features**: it contains all the application's features and each of them contains all the layers described before;
- **core**: it contains all the features and the utilities which are in common between more than one feature.

This is done in order to obtain the goals described before (Section § 3.1.1).
For example, if it is decided the authentication will not be managed using *Firebase* in the future, it is possible to do it by creating a new class which implements the abstract class methods.

3.2 BLoC

BLoC stands for *Business Logic Component*. It is a state management pattern that is popular in *Flutter* and it separates the *business logic* of your app from the *UI* code. This makes it easier to make changes to the *UI* without affecting the *business logic*.

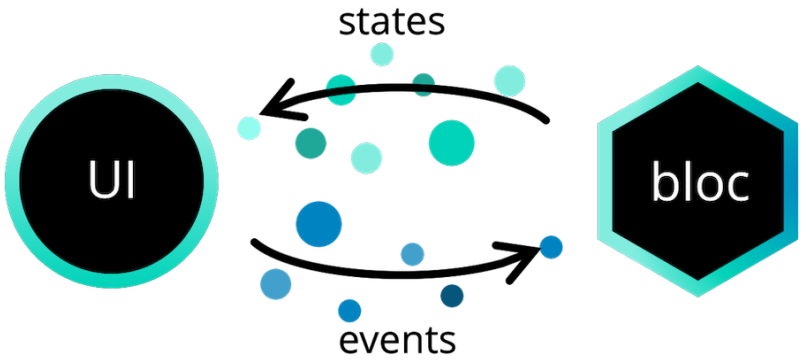


Figure 2: Bloc Pattern

BLoC works by having a stream of events that drive the state of the application (*Figure* 2). When an event occurs, the *BLoC* updates its state and emits a new one to the stream. The *UI* widgets listen to the stream of states and update themselves when necessary.
As underlined in the section §3, the pattern has been implemented only in the authentication and in the questions because we did not have time to apply it in all the others. Thus, stateful widgets have been used instead for state management.

4 Edukid application

You can find the source code at this GitHub repository link:

https://github.com/xhttpMar/edukid_mpm

Edukid is an application that allows children to learn and assess their knowledge in primary school subjects such as Math, Geography, History, Science etc.
The application works on the following minimum versions: Android 11 (Android SDK level 30) or iOS 11 and it requires at least 50MB of space.

4.1 Structure of application

For the description and analysis of each page we will follow the sequence of actions that the user will face.
Edukid works both on Android and iOs: since there are no great differences on the UI, the following applies for both the platforms and any differences (if present) will be highlighted in this section.

4.1.1 Login

As soon as the user opens the application, it requires to login or, in case the user does not have an account yet, to signup.

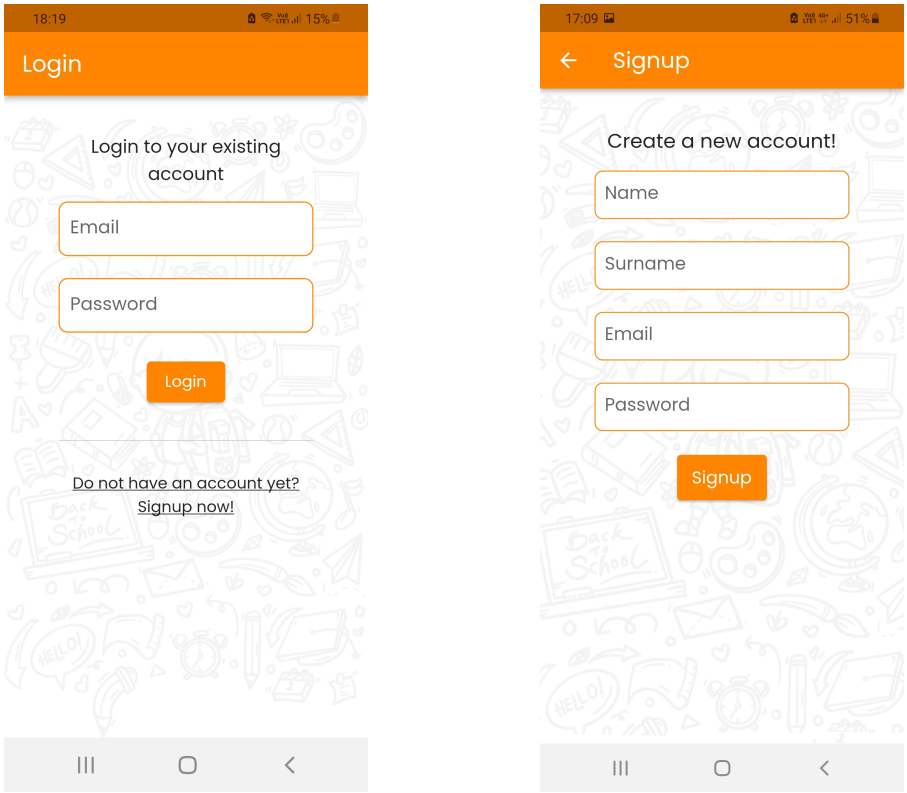
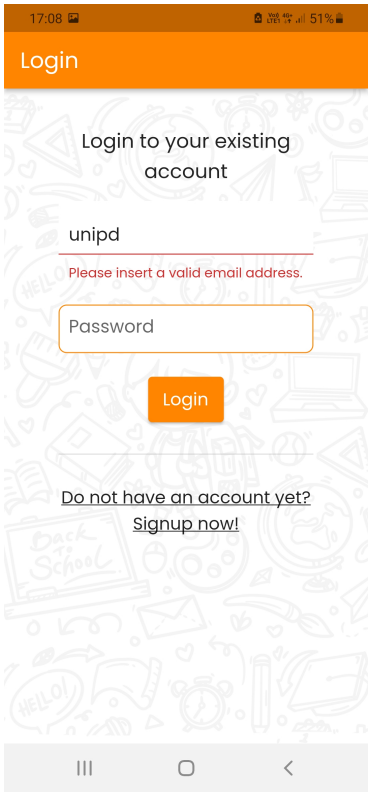


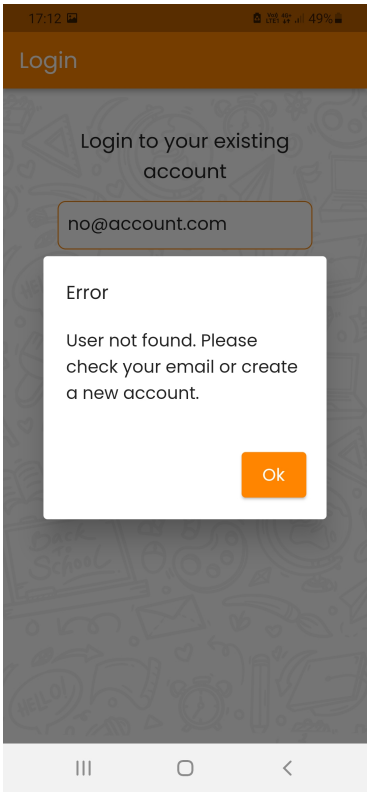
Figure 3: Login and Signup screen

The interface has been kept as simple as possible: for the login we just need the *email* and *password* and we perform a validation both on client and on server side. For what concerns the signup we just need name, surname, email and password. Since we do not ask for other secondary information, there are just 4 fields, which is an acceptable amount for a form.
The validation of the input is the following and it happens while the user is still filling the form, to reduce the number of interactions:

- email has to be in the correct format, otherwise a text message appears under the input field as you can see in Figure 4a;
- password must be at least 6 characters long, otherwise a text message appears under the input field ;
- fields cannot be empty, otherwise a text message appears under the input field;
- if user hits the *Login* button and email address is not recognized or if the password is wrong, user gets an alert dialog explaining the error, as shown for example in Figure 4b;
- if user hits the *Signup* button and email address is already associated with an account, user still gets alerted with a dialog.



(a) Error on client side that appears under the input field



(b) Error on server side the account use when email used to login does not exists

Figure 4: Errors during login or signup

When the user signup correctly, he/she is logged in automatically and he/she is redirected to the homepage (or to the wizard as explained below).

It is also important to underline that when the user is correctly logged in (either with login or signup), *Firebase Authentication* maintains the persistence.
So, for example, let's suppose the user is logged in and he/she kills of the application without doing logout. Then, when he/she opens the application, he/she will be automatically logged in and ready to use the application. This is useful because it allows to reduce the effort of the user he would be subjected, if he/she had to do the authentication every time.

4.1.2 Wizard

Once the user correctly entered in the application, if he/she just created the account it appears a wizard, as you can see in Figure 5.

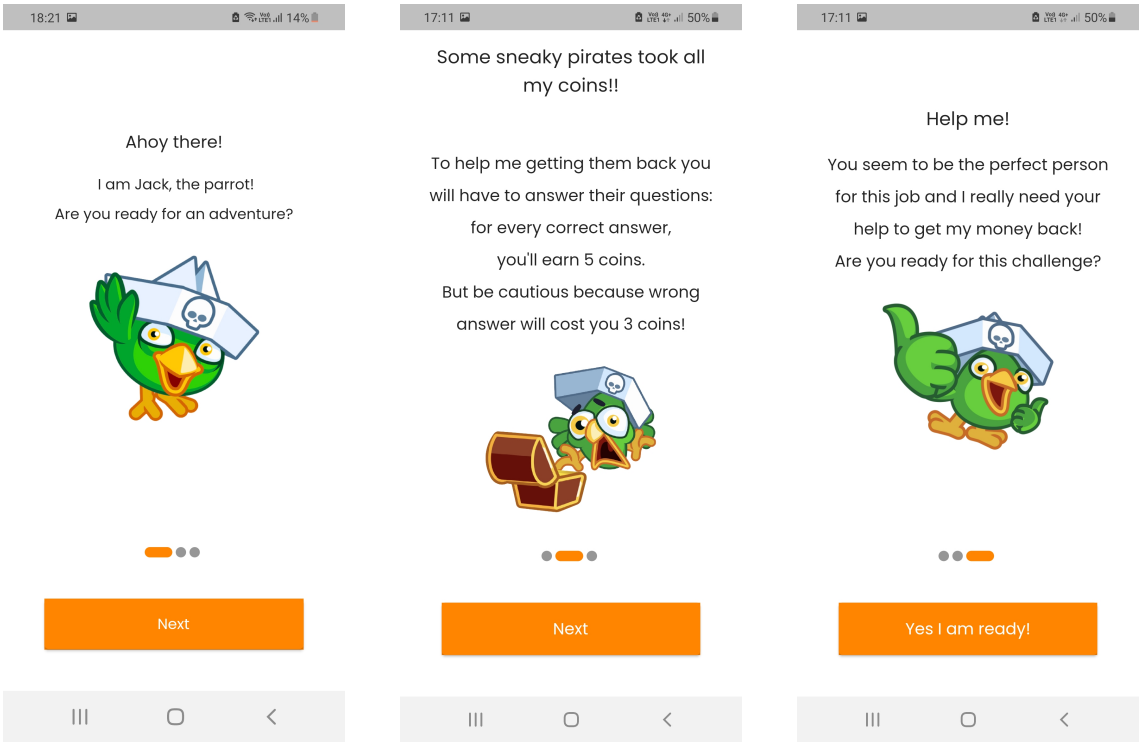


Figure 5: Wizard steps when user opens app for the first time

This introduction story helps in engaging the kid, and *Jack the Parrot* then becomes the mascot of the application, as we will see later.
Some pirates took away all its coins and it then asks the user to help in getting them back.
User can both use the *Next* button or he/she can use the swipe gesture to go back and forth. Once the user comes to the end of the story and clicks on *'Yes I am ready!'* button, he/she gets redirected to the homepage.

4.1.3 Homepage

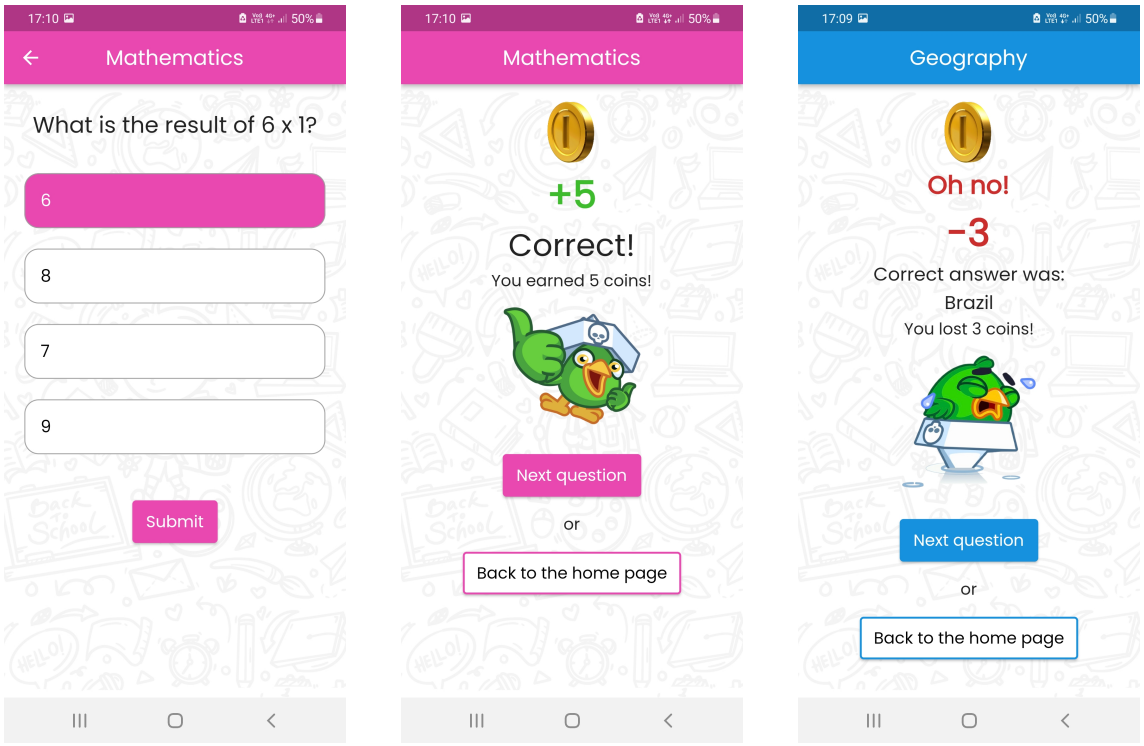
The homepage shows the different categories displayed as clickable cards in which the kid can focus on: Mathematics, Geography, Science and History. You can see a screenshot in Figure 6.

Figure 6: Homepage

The homepage also shows the current number of points.
We chose to place this information in the top left part of the screen, since it is something the kid may want to know, but it does not need to be easily reachable by the thumb.
On the right side, that is slightly more reachable, there is an info button. When clicked, it appears a dialog telling the user how to play.
As we will see later, there already is a specific page with the instructions on how to play, but this dialog serves as a shortcut and only reports the essential information.

Another important thing we want to highlight is the red dot that appears under each category when the user has not answered to any questions yet. This is an important information that aims to encourage the kid trying out all the subjects. In the example in Figure 6 the user has set a score only for Mathematics and Geography.

When the kid chooses the category, the first question appears, as you can see in Figure 7a.



(a) Question (b) Correct answer result (c) Wrong answer result

Figure 7: Question page and question results

The color of the question page is the same as the color showed in the different cards of the homepage: this helps with the synesthesia phenomenon, i.e. when you are able to associate a color with a specific thing.
In the screenshot example you can see Maths is associated with fuchsia and Geography with light blue. This combination of colors is persistent throughout the whole application, as we will also see later.

In order to engage with the kid and stick with the introduction story (i.e. the pirates stole the parrot's money), Jack the Parrot appears and he is happy if answer was correct (Figure 7b), while he is very sad if answer was wrong (Figure 7c). In the latest case, correct answer is displayed so that the kid can learn it.

It is important to underline that kids are usually very clumsy and they could click on a category card unintentionally. As you can see while in *Android* there is the possibility to get back to the homepage with the back button provided by the operative system, in the *iOS* environment there is not that opportunity. So, in order to allow to all the devices to come back to the homepage, a back button was added to the top left corner.
Therefore kids can then return to the homepage or keep answering to the chosen category question.

Last but not least, coming back to homepage (Figure 6) as you can see on the left of the page title there is an hamburger menu. The open menu is shown in Figure 8

and it can be opened both by clicking the icon or swiping right (this gesture works both on Android and iOs).
From this menu, the user can navigate through the different pages in the application and he/she can also directly logout. To show the user in which page he/she is currently in, the menu entry is highlighted in orange

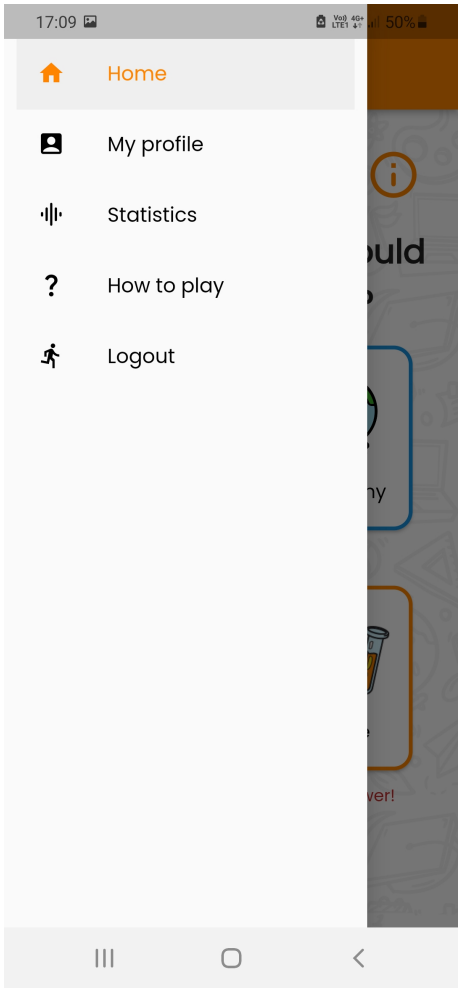
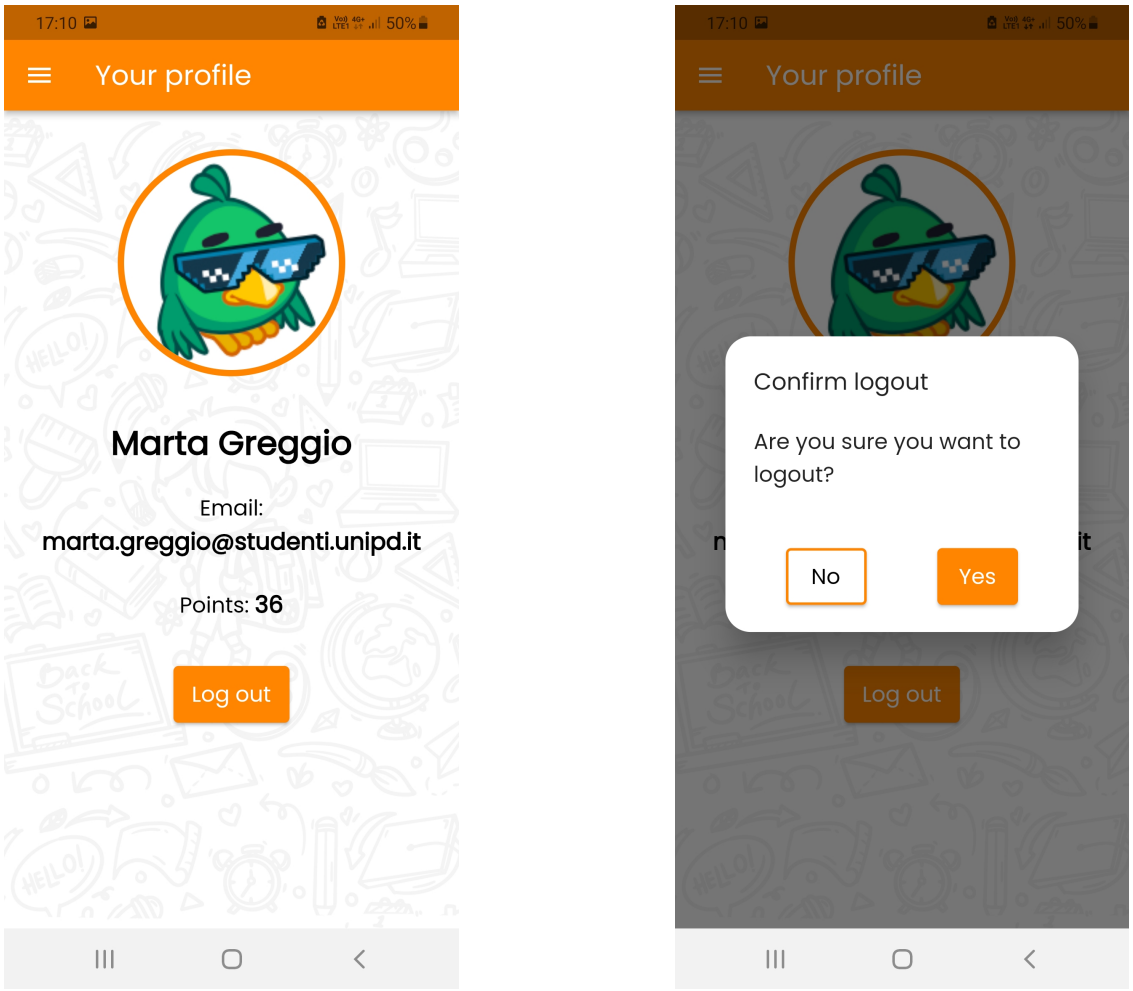


Figure 8: Open side menu

4.1.4 Profile

From the drawer menu, the first page the user can reach is the one about his/her personal information. A screenshot is shown in Figure 9a. The user here can see the account with which he/she is currently logged in and can also perform the logout operation.
As it happens with the *'Logout'* entry on the menu drawer, when the user clicks on the button it appears a dialog asking for confirmation, as you can see in Figure 9b.



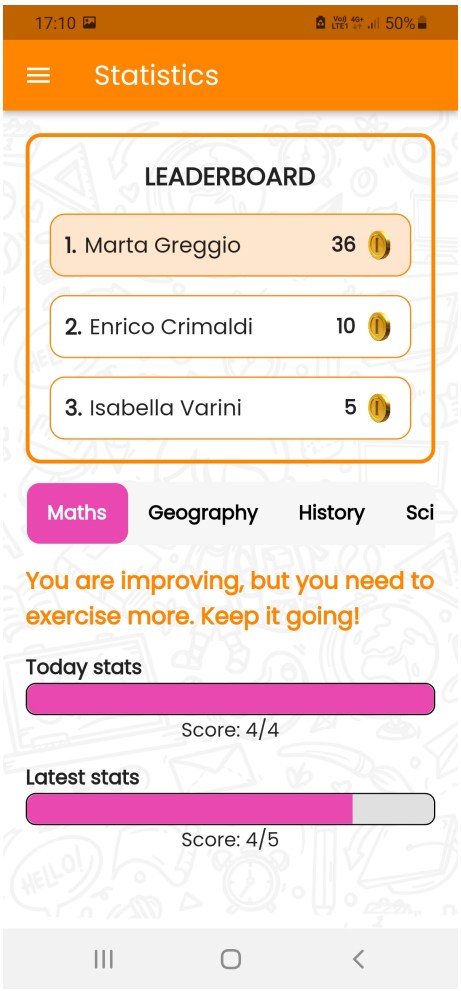
(a) Personal information page

(b) Confirm logout dialog

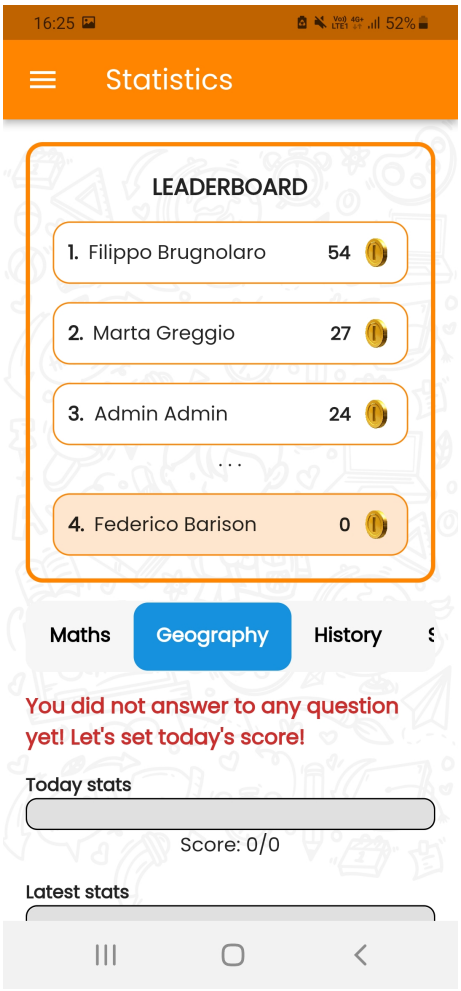
Figure 9: Personal information page and confirmation logout dialog

4.1.5 Statistics

The most interesting and actually useful feature of the application is the Statistics page.
From this page the user can monitor his/her progress. The screenshot is shown in Figure 10.



(a) Leaderboard when user is on the podium



(b) Leaderboard when user is off the podium

Figure 10: Statistics page

The leaderboard shows the first three users according to their amount of points. The logged user is highlighted in orange.
If the user is not in the podium, the user's card is still shown highlighted in orange with the position in the ranking and the number of points, as you can see in another screenshot in Figure 10b.
Leaderboards introduce gamification. It is a way to let the users compare their progress and points with those of their classmates or competitors.
It has been shown that contests with leaderboards push learners' competitive spirit and focus that competition on fun — while also encouraging and rewarding engagement with training and progress toward learning goals.

Then we have the statistics for each category. It has been implemented as a (horizzontally scrollable) tab view. The four tabs represents the four categories and for each of them there are two progress bars reporting the current performance (Today's stats) and the latest performance (it represent the last day the kid played with the application, it has not to be the previous day).

Even in this case each category is associated with a color, and both the tab title and the progress bar have that color.
As one may notice, there is also a sentence cheering for the kid in case he is doing well. This sentence obviously changes according to the kid's performance.
For example, if today's score is higher than the last one, but the number of answered questions is smaller, then the app will show the kid that he is improving but he still needs to exercise more.
If you are interested in seeing the full list of cases and different messages, you can find it in file `features/statistics/presentation//pages/statistics.dart` from line 394 to 479.

4.1.6 How to Play

This is just an instruction page and it reports the rules of the game. You can see a screenshot in Figure 11.

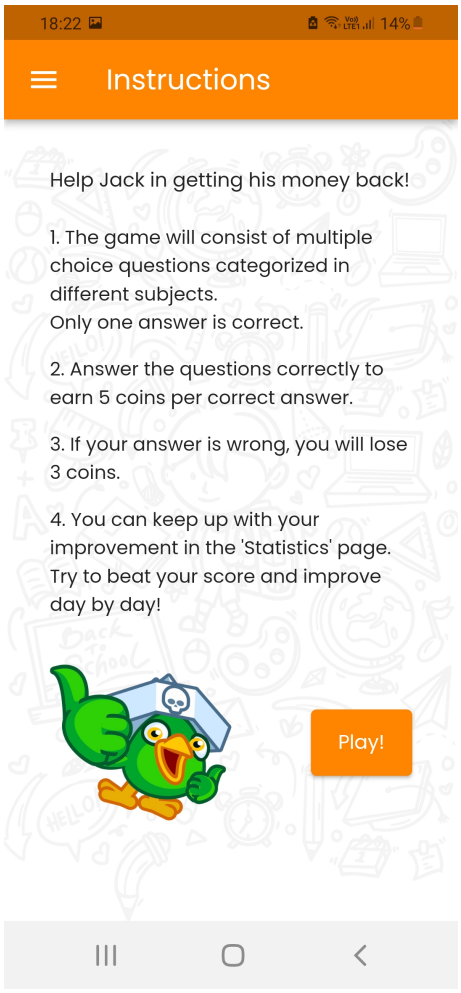


Figure 11: How to Play page

The different rules appear as a numbered list, explaining how the game works. As you can see there is still a reference to Jack the Parrot so that it is consistent within the application.
Once you click to *'Play'* you get redirected to the homepage.

4.2 General design choices

4.2.1 Device orientation

The application has been designed to be used only in portrait mode. The reason behind this choice is based on the fact that we wanted to minimize the vertical scroll. For example, while answering to a quiz from smartphones, we have found that if we gave the possibility to have the landscape mode, the user would have to scroll in order to see all the options and this would ruin his/her experience.
Furthermore, human eye is more used to see in portrait mode when it comes to screens and for the purpose of the application we thought it was the best solution.

4.2.2 Colors

Since it is an application for kids, we chose bright colors such as orange, green, fuchsia, blue, etc.
The main color is orange, since it is frequently associated with optimism and energy and it helps in grabbing attention.

4.2.3 Content's size

Every dimension is expressed in relative units (Scale-independent pixels for texts, height and width of the device for the rest). Thanks to the `sizer` package in *Flutter* we have been able to resize buttons, spaces, texts, etc according to the height and width of the used device. In this way we are guaranteed that our *UI* is responsive and works on different devices.
In particular, almost all the text widgets in the app have been made scalable with respect to the user's settings on the device. In this way, the user can make the text bigger and apps that support dynamic type will adjust according to the preferred reading size. You can see some screenshots in Figures 12 and 13.
There is only an exception in the homepage: the text present in each card (Math, Geography, etc.) and the eventual message *'No Answer!'* are fixed, because when we tried to make it bigger we noticed it was ruining the appearance of the homepage. Since the card is already self-explanatory with the illustration inside it, we thought we could leave it like this, not enabling the scaling factor.
Widgets' size has been also checked through *Flutter Inspector* so that we could diagnose layout issues and thanks to this tool we also guaranteed the minimum size for widgets (e.g. 44px for buttons).

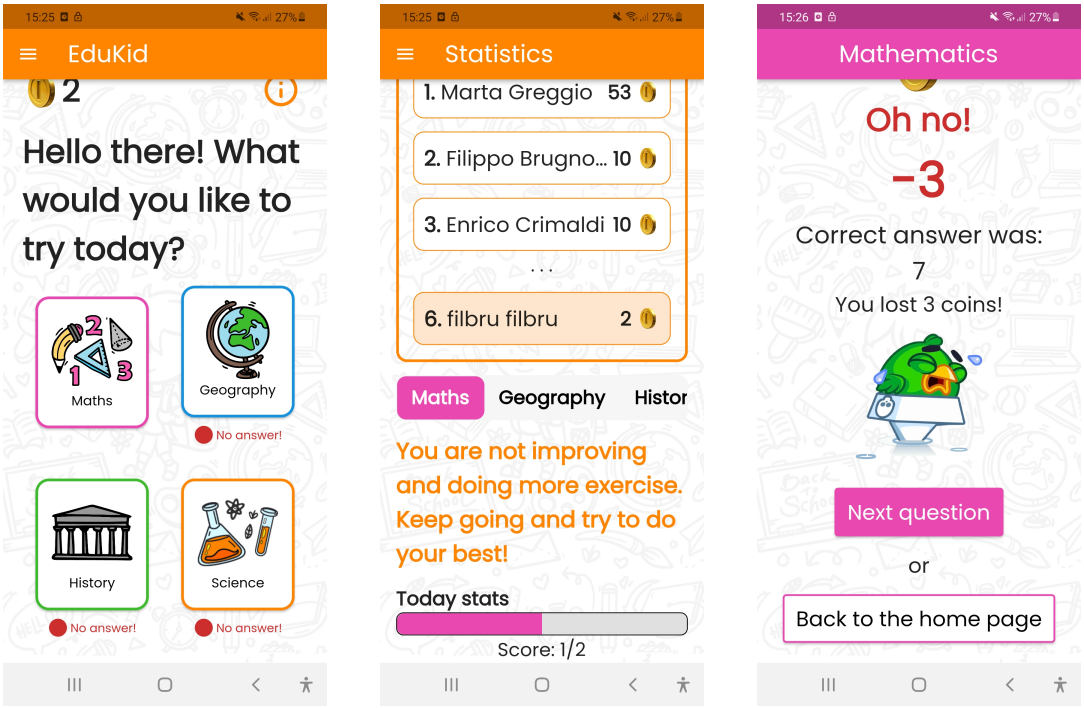


Figure 12: Font size ≈ x1.5 user settings in Android

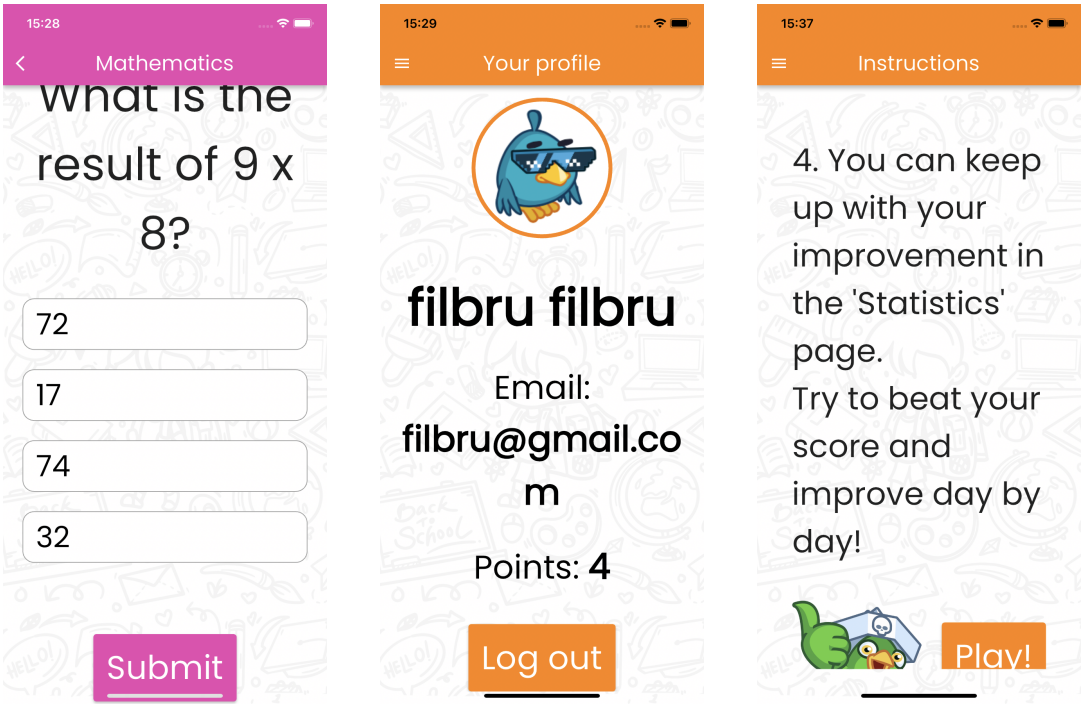


Figure 13: Font size ≈ x3.0 user settings in iOS

4.2.4 Images

To ensure efficient storage and faster loading times, we used the PNG format. This was preferred to JPEG to preserve transparency in icons and illustrations.
For what concerns copyright and licences:

- illustrations in the homepage have been realized by ourselves with Adobe Illustrator, taking some inspiration from the web;
- Jack the Parrot is a famous stickers pack from *Telegram* in which you can find the source we used for the different images;
- avatars in the Personal Data page has been taken from the same sticker pack above from *Telegram*.

4.3 Testing

In this section we will report the devices we used to test the UI of our application. For the Android environment, we used both a virtual emulator and a physical device:

- Virtual emulator: *Pixel 3a XL*, screen size 6", SDK level API 30 and Android 11;
- Physical device: *Samsung Galaxy A50*, screen size 6,4" and Android 11.

Meanwhile for the iOS environment, we only used the physical device provided by the Mobile Lab:

- Physical device: *iPhone XR*, screen size of 6.1", iOS 16.6.

4.4 Extensions

In this section we will explain all the features we did not implement for time issues and they can be developed in the future.

4.4.1 Push notifications

Initially push notifications were a feature which had to be implemented. They were important in order to increase the engagement of the users with the application. In fact they would have remembered them if they did not answer to questions of a particular category or if they did not access to the application for a long period of time.
However a problem arises. In fact, in order to manage the notifications from server side, *Firebase Cloud Functions* were required and they would have cost us money after the use of a low free number of calls. For that reason, it was preferred not to implement them.

There would have been also the possibility to implement them with local push notifications using the corresponding Flutter_local_notifications *Flutter* package. However the notifications would be triggered only if the call to the local notification manager were done. This limitation would have made the project more complex and it would have required a lot of time to manage particular behaviors of the application.

For the future, developers can implement this feature by using *Firebase Cloud Functions* or by developing and integrating a *custom backend*, which is possible to substitute with the *Firebase* thanks to the *Clean Architecture* (Section §3.1.1).

4.4.2 Statistics scaling

As you can see, the application displays a limited amount of statistics which are the *Today* and *Latest* (2 days in total).
However it could be a good idea to scale the display of data statistics to a larger amount in order to create more specific graphs. In this way the user would be more conscious and would understand better his/her strengths and weaknesses, seeing his/her trend in a larger amount of time.

4.4.3 Category's leaderboard

At the moment the leaderboard only reports the global ranking among all different categories. It could be an interesting idea to create also a leaderboard for each subject: a competitive kid may want to excel everywhere and thus this could be an encouragement to master all the subjects.

4.4.4 Virtual rooms

Another thing than came up to our minds was to implement some *virtual rooms* such that kids of the same school or class can have their own leaderboard, along with the global one. This would be also helpful for the teacher, that could monitor her/his students' progress.

5 Default configuration

5.1 Accounts

The application starts with the login page and it is required to sign in to use the application. So, there are 3 preconfigured accounts which are the following:

- User User:
  - **Email:** user@user.com
  - **Password:** useruser
- Marta Greggio:
  - **Email:** marta.greggio@studenti.unipd.it
  - **Password:** martagreggio
- Filippo Brugnolaro:
  - **Email:** filippo.brugnolaro.1@studenti.unipd.it
  - **Password:** filippobrugnolaro

However it is possible for anyone who wants to use the application to create a new account.

5.2 Database

The application uses *Firebase Real-time Database* to store the information which are necessary to guarantee its proper functioning. It is a NoSQL database and the information are organised in a hierachical way. Now we briefly describe the structure of the tree:

- **subject:**
  it contains the 4 categories:
  - **Mathematics**
  - **Geography**
  - **History**
  - **Science**

  Each of them contains a number of **question** which have been generated randomly. Each question contains:
  - **question text**
  - **options** (it contains 4 choices)
  - **answer**

- **users**: it contains all the user **UUID** in order to distinguish them unequivocally. Each of them contains:
  - **name**
  - **surname**
  - **email**
  - **number of points**
  - **statistics**
  - **reset statistics** (used to know when to reset the statistics of the user)
  - **wizard display** (used to display the wizard only the first time)

  The statistics are divided into the 4+1 categories :
  - **Mathematics**
  - **Geography**
  - **History**
  - **Science**
  - **Global** (it collects the global statistics considering all the categories)

  Each one of the 4 categories have:
  - **current** day's statistic
  - **latest** day's statistic
  - **total** (i.e. global counter for that category)

  current and **latest** day's statistic have:
  - **correct** answers
  - **done** answers
  - **timestamp**

  While **total** have:
  - **correct** answers
  - **done** answers

  There is also the additional category **Global** which have:
  - **correct** answers
  - **done** answers
  - **timestamp**