POLITECNICO DI TORINO

Master's degree
in Computer Engineering

Master's Degree Thesis

Well-being App: Designing a Secure and Scalable Backend for a Mobile Health and Wellness Solution

Supervisor
prof. Maurizio Morisio

Candidate
Domenico Gagliardo

Academic Year 2024-2025

Abstract

The spread of smartphones and wearable devices across the years have contributed to the development of new technologies. Wearable devices, thanks to the possibility of monitoring the user and collecting data, have made possible technology integration especially in the health sphere. However, sedentary lifestyle remains a common and widespread problem. The health app project aims to address this challenge through the implementation of a mobile application, called Well Being App. The goal is to develop an application capable of providing functionalities which encourages the user to be more active with daily goal to meet, but also by prompting him to insert information periodically, in order to improve his lifestyle and achieve the best health condition possible. All this was done by providing intuitive charts to visualize the data, a user friendly way to insert data, as well as a quiz and a notification system to engage the user. Furthermore, sign-up information together with activity goals are easily accessible and editable, and wearable integration has been made, all coupled with a multi language support, to provide a better user experience and make the application more intuitive and easy to use.

Health guidelines set by leading organizations in the field were firstly considered in framing the application, moving then into how technology can help achieve an healthier lifestyle. By having that in mind, the requirements (both functional and non-functional) were defined. From that point, also the architecture was consequently defined, and the technology stack chosen. Flutter covered the frontend part while Google Firebase the backend, chosen since it provides a comprehensive solution for real-time database, authentication, and cloud storage, as well as an easy integration with flutter. After that follows the implementation of the system architecture, with the contributions made on the application, particularly on the backend part. Finally, the application system performance were assessed and the achieved results were discussed, as well as possible future enhancements and potential new features.

Key Words:

Health, Sedentary Lifestyle, Wearable Devices, Google Firebase SDK, Mobile Application, Flutter, User Engagement, System Architecture, System Performance.

Contents

Introduction

The work that is being presented in the next pages is based on two main arguments: the importance of following a healthy lifestyle and how technology can help in achieving this goal. Infact, our project work along with my colleagues concerned the implementation of a mobile application suited for achieving a good lifestyle. The application, composed both by a frontend part and a backend one, allow the users to track their data regarding metrics like nutrition, as well as physical activity, sleep and emotional state. These data can be either inserted manually in an easy way or collected through a wearable device, and are intuitively displayed through charts. The application still requires the user to insert his basics information (username, age, sex, weight, height), which can be modified later in a dedicated section, together with his activity goals. The application also integrates a quiz gamification approach through a specific section of the app, that allow the users to be more engaged, but also learn and deepen their knowledge about the topic. Finally, a notification system to prompt the user into inserting information is included, as well as multi language support in order to provide a better user experience. In addition, a web application was also conceived to assist the admin into editing some application parameters and to help him into lesson and quizzes management. The thesis is then structured in 5 chapters: the first one deepen the health and well being topic, considering the guidelines that literature has found over the years of studying the topic. It then focuses on how technology can help us in achieving a better lifestyle, by considering which are the main software and hardware tools that can be used, such as smartphone application and wearable devices that allow to collect data and share them with application to create a more complete picture of the user's health. The second chapter covers the system, by firstly consider the requirements related to it (both functional and non-functional) and then moving to the system architecture, designed consequently. The third chapter focuses on the technology stack, covering framework, programming languages, automation tools, the IDE, and backend technlogies and integrations. The fourth chapter then talks about the practical implementation of the application system, by focusing on the backend system. Finally, the fifth chapter talks about the performance of the application system, and then concludes the work by considering the achieved results and the future work that can be done to improve the application.

1 Health and Well Being

Health is one of the most important, if not the most important aspect of a person's life. For this reason, over the years, different organisations have established guidelines on how to stay healthy, thus increasing people's life expectancy and quality of life. Among these, the most widely worldwide recognized is the World Health Organization (WHO) [1] that provides several guidelines not only in term of physical activity, covering also other health aspects.

As far as concerns the physical activity, the WHO estimates that 1 in 3 adults and 4 in 5 adolescents do not do enough physical activity, with adolescents girls less active than adolescents boys and with inactivity levels that increases after 60 years of age. This level is expected to rise due to country economic development (more use of technology, change of cultural values and more sedentary behaviour). This trend sadly keep going in the wrong direction, despite the fact that physical activity has countless benefits, like reducing the risk of heart disease, cancer, diabetes, hypertension and depression.

- Children and Adolescents:
  - **Regular physical activity** enhances fitness, cardiometabolic health, bone strength, cognitive and mental health while reducing body fat.
  - **Sedentary behavior** leads to increased adiposity, poorer cardiometabolic health, behavioral issues, and reduced sleep duration.
- Adults and Older Adults:
  - **Active adults** experience lower body fat, risks of all-cause mortality, cardiovascular diseases, hypertension, specific cancers, and type-2 diabetes. They also enjoy improved mental health, cognitive function, and sleep quality.
  - **Sedentary lifestyles** are associated with higher mortality rates and increased incidences of chronic diseases like cardiovascular issues and cancer.
- Pregnant and Post-Partum Women:
  - **Engaging in physical activity** decreases the risks of pre-eclampsia, gestational hypertension, gestational diabetes, excessive weight gain, newborn complications, and postpartum depression, while having no negative effects on birth weight or stillbirth risk.

Active vs Sedentary lifestyle[2].

Food is also crucial in order to be healtier. Having a healthy diet helps to prevent several diseases (like heart disease, diabetes and cancer) and also malnutrition in all its forms. However, care has to be taken in choosing the right food sources that have good quality and avoid processed foods. Eating noble foods like fruits, vegetables, legumes, nuts, and whole grains, while limiting the intake of salt, sugar, and fats, is the key to a healthy diet. For all these reasons, both physical activity and diet are strongly promoted by the WHO through his global action plan, by calling international partners, private sector and also civil society to take action in order to support them.

1.1 Guidelines

1.1.1 Physical Activity Guidelines

As far as concerns the physical activity, the WHO gives some recommendation based on the age group [3]:

- 5-17 years:
  - Should do at least 60 minutes of physical activity with moderate/vigorous-intensity daily (of course more than 60 minutes provides additional benefits), as well as bone-strengthening and muscle-strengthening activities.
- 18-64 years:
  - Should do at least 150 minutes of physical activity with moderate-intensity in a week or at least 75 minutes of physical activity with vigorous-intensity in a week or an equivalent combination of both (increasing moderate-intensity will provide additional benefits), but also muscle-strengthening activities by involving major muscle groups.
- 65 years and above:
  - Should do at least 150 minutes of physical activity with moderate-intensity in a week or at least 75 minutes of physical activity with vigorous-intensity in a week or an equivalent combination of both (increasing moderate-intensity will provide additional benefits), recruiting major muscle groups with muscle-strengthening activities but also including exercises to enhance balance and prevent falls in case of poor mobility.

1.1.2 Healthy Diet Guidelines

Regarding having an healthy diet, also here the WHO gives some guidelines, emphasizing that a good diet includes legumes, fruit, vegetables, animal sources foods (like meat, fish, eggs, and milk), cereals (like wheat and barley) and also tubers (like potato and yam). It also gives some further recommendations[4]:

- Babies and young children breastfeeding:
  - Breastfeeding promotes healthy growth, as well as having long-term benefit, like reducing the risk of developing noncommunicable diseases, overweight, obesity. From birth until 6 months of life is important to feed the baby only with breastmilk, while from 6 months to 2 years of age is important to introduce also additional complementary foods, while still breastfeeding.
- Eat lots of vegetables and fruit:
  - These foods are rich in vitamins, minerals, dietary fiber, antioxidants and plant protein, which help to prevent heart disease, stroke, diabetes, obesity and some cancers.
- Eat less fat:
  - Fats and oils are concentrated source of energy, so it is important to limit them, especially saturated and industrially-produced trans-fat that can increase the risk of stroke and heart disease. To avoid gaining weight in an unhealthy way because of them, care has to be taken in using unsaturated vegetable oils (like olive oil) instead of animal fats or oils high in saturated fats (like butter or palm) and in any case fat consumption should not exceed 30% of total energy intake.
- Limit sugars:
  - Sugar consumption should be the 10% of total energy intake. This should be achieved by limiting soft drinks, soda and other drinks high in sugars (fruit juices or yogurt drinks) and also by avoiding the consumption of processed foods high in sugars (like cookies, cakes, chocolate). Better to choose fresh fruits instead of them.

1.2 Technology Role in Health

Having clear in mind the importance of an healthy lifestyle and a good dieting, it is also important the role that technology can have in this. Even though it is still possible to achieve a good lifestyle without technology, it has to be said that using technology sure makes it easier across several aspects. Several researches in this aspect have been performed by the National Institutes of Health (NIH) [5], an american health organization driven by the U.S. Department of Health and Human Services. The NIH found notable improvement in diet and activity habits with usage of mobile technology.

They took 204 adult people that met those constraints:

- Being obese or at least overweight.
- Having a diet high in saturated fat and low in fruits and vegetables.
- Perform a small amount if daily physical activity.
- Having lots of sedentary time.

then they divided these people into four groups, where each one had a specific diet. Furthermore, a mobile device was given to them and they had to enter their diet and activity data into the device for a 20-week follow-up period. Coaches would then receive the data during this period to monitor these people, as well as contacting them in order to provide encouragement and support towards an healthy change. The results found overall improvements in all four groups, emphasizing how technology can improve a fitness journey, also as a means to provide support and motivation [6].

Another aspect in which technology surely can help is about measurements: during physical activity and dieting, several aspects require measurements: the amount of calories burned, the heart rate during physical activity, the amount of calories taken, as well as the types of food consumed, their macronutrients (carbohydrates, proteins and fats) and so on. On this matter, technology can provide several tools to help, like smartphone applications or wearable devices.

In the first place, technology helps in easing the process of performing measurements and gather those data (both for physical activity and dieting) that can be boring to do repetitively for us humans. In the second place, technology can provide a more accurate measurement of those data, more difficult to achieve manually. Related to this aspect, a study of the NIH showed how physical activity measurements taken by devices proved to be more punctual compared to the one taken manually with a diary [7].

1.2.1 Smartphone Applications

Moving to the technological tools that can be employed, smartphones surely are one of the most used devices and they allow to exploit several aspects related both to dieting and physical activity. Also here a study of the NIH [8] showed that users were more stimulated into following a healthy diet, particularly liking applications that were quick and easy to administer, and those that increase awareness of food intake and weight management. Even though work has to be made to increase food awareness, the study recognizes the importance of smartphone applications in this aspect. Dually another study has been done also on physical activity [9], showing that smartphone apps can be efficacious in promoting physical activity. Also in this case users tend to prefer applications that are user-friendly, thanks to their capability of automatically track physical activity (e.g., steps taken), track progress toward physical activity goals, as well as be flexible enough to be used with different types of physical activity. Countless of these smartphone applications are available to support an healthy lifestyle. They are cross-platform, so they can be used on both Android and IOS devices, in order to reach the largest audience possible. Here are some of the most popular applications, where for each the main features and the feature that distinguishes the application from other on the market are listed:

App Name	**Features (Distinguishing Features In Bold)**
MyFitnessPal	Food logging with a large database, barcode scanning, calorie and macro tracking, personalized insights, exercise logging, and integration with other apps. **Share and View Your Diary with Others.**
Fitbit	Step tracking, heart rate monitoring, sleep analysis, GPS tracking, food logging, and activity reminders. **Comprehensive health metrics tracking, including stress levels and Active Zone Minutes.**

Google Fit	Step counting, activity tracking, heart points, integration with health apps, and customizable fitness goals. **Collaborates with the American Heart Association and WHO for heart health insights.**
Nike Training Club	Guided workout programs, personalized fitness plans, workout tracking. **Free access to a variety of workouts, from yoga to high-intensity interval training.**
Strava	GPS tracking for running, cycling, performance metrics, social sharing, and route planning. **Community-focused with support for sharing routes and competing with others.**
Noom	Weight loss coaching, food logging, biometric tracking, and habit-building tools. **Focus on the psychological aspects of diet and health for long-term results.**
JEFIT	Exercise logging, workout planning, personalized workout, and performance tracking. **Training optimization with advanced analytics.**
Cronometer	Nutrition tracking, biometrics tracking, diary and diary groups, and micronutrient breakdown. **Advanced nutrient tracking suitable for specific diets.**
Lifesum	Food tracking, calorie counting, diet plans, water tracking, and nutrient breakdown. **Visual and user-friendly meal planning tailored to dietary preferences.**
Yazio	Calorie counting, meal planning, recipes, nutrition tracking, and progress reports. **Extensive recipe database and meal planning features.**

Table 2: Overview of popular diet and fitness apps with distinguishing features highlighted

1.2.2 Wearable Devices

Considering the Wearable Devices, even though they are less used compared to smartphones, their usage is growing more and more. Furthermore they are a good tool in order to track physical activity and dieting. Their main advantage is that they can be worn on the body, like a watch or a bracelet, but they are equipped with sensors allowing to collect data about the body, like the heart rate, the number of steps taken, the calories burned, the quality of sleep, and so on. They can also be connected to a smartphone in order to share the collected data, so that the user can have a more detailed view of his health status. Given their diffusion, applications have been introducing a way to connect to these devices, in order to exploit their data. This has been done by carefully considering their diffusion [10].

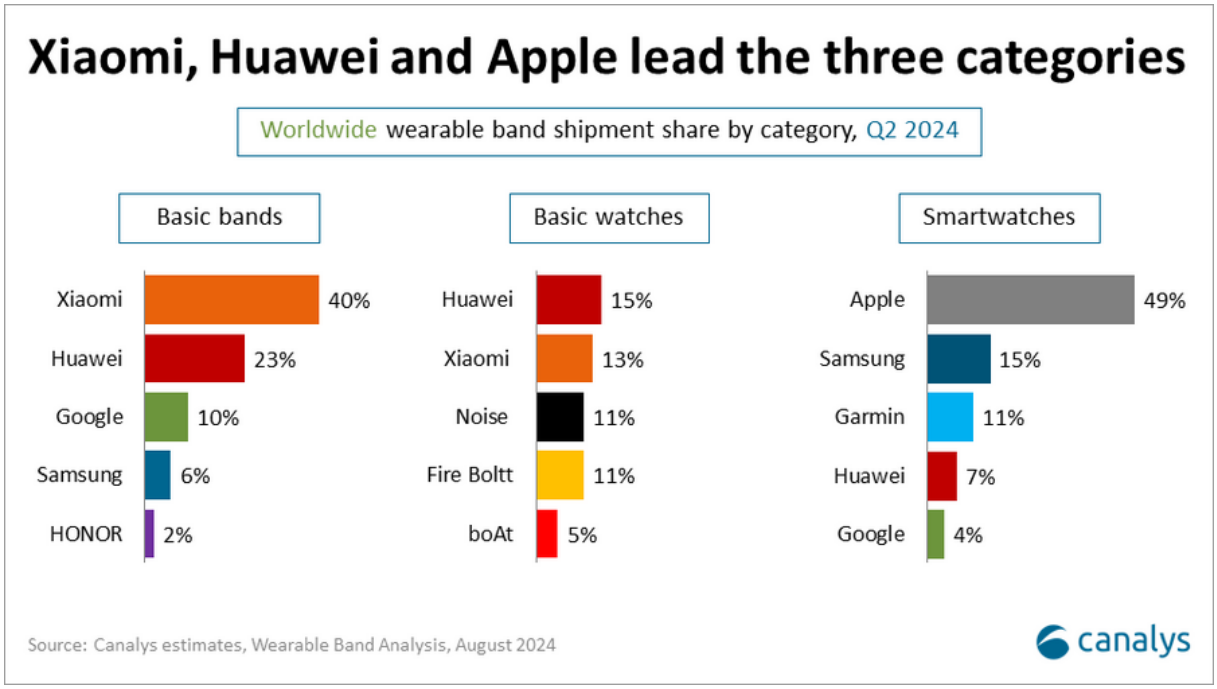


Figure 1: Wearable diffusion by major brands [10].

2 System Specifications

2.1 System Requirements

By defining the system functional requirements, the application key functionalities to implement were identified, by considering also existing applications (see table 1) that represent standards on the health and fitness market. Security, usability as well as other important aspects were considered, leading to the definition of the non-functional requirements. The requirements can be classified as follows:

FR1	**User Management**	
FR 1.1	Registration	Allow a user to register through google or by defining his credentials. Additionally, allow an user to enter his demographics (age,sex,...) and body (height,weight,...) data when the registration is in progress.
FR 1.2	Login Management	Allow a user to log in into his account with the methods that he had set up and allow also to change the password, including the possibility to recover it in case he forgot it.
FR 1.3	Account Management	Allow a user to log out from his account and delete it. Allow to add either google or classic credential method as login method, in case he hadn't used before.
FR 1.4	Home Page	Allow a user to visualize data regarding his steps, food, sleep and emotional state. Allow also to visualize data that exceeds the user goals differently (they visually differ from the other ones).

FR1	**User Management**	
FR 1.5	Learn Page	Allow the user to take lessons and then related quizzes to assess his preparation on the topic through the learn page. The user can browse the lesson and his topics, then take the corresponding test if he wants.
FR 1.6	Health Measures Page	Allow a user to view his vital metrics by using charts organized in different tabs instead of raw values, for a better understanding.
FR 1.7	Personal Information Page	Allow a user to edit some of his personal measures (provided at registration time) and add his activity goals, used as upper bound into the charts of the Home Page.

Table 3: Overview of Functional Requirements related to the User Management.

FR2	**Admin Management**	
FR 2.1	Admin Application	Allow the admin to use a web application in order to modify system metrics.
FR 2.2	Login/Logout	Allow the admin to login into the web application, as well as to logout.
FR 2.3	Parameters Edit	Allow the admin to edit the application parameters set up and shown to users.
FR 2.4	Application Lessons	Allow the admin to manage the lessons that are provided to the users inside the application.
FR 2.5	Application Quizzes	Allow the admin to manage the quizzes that corresponds to the lessons and are provided to the users inside the application.

Table 4: Overview of Functional Requirements related to the Web Application provided to the Admin.

FR3	**Notification System**	
FR 3.1	Notification System	Allow a user to receive notification with different frequencies to prompt him into inserting data regarding food, emotional aspect, balance capability, strength capability.
FR 3.2	Notification Parameters	Allow a user to receive notification based on common users parameter, but also personalized based on some personal parameter.
FR 3.3	Body Balance Notification	Allow a user to receive notification to prompt him to insert data regarding his balance capability, such as balance test on one leg at a time or tandem walk.
FR 3.4	Body Strength Notification	Allow a user to receive notification to prompt him to insert data regarding his strength capability, such as number of squats, abdominals and push ups, as well as a grip strength test.
FR 3.5	Emotional Notification	Allow a user to receive notification to prompt him to insert data regarding his emotional aspect by following the panas guidelines.
FR 3.6	Food Notification	Allow a user to receive notification to prompt him into inserting data regarding his consumed food, with a different frequency, compared to the others.
FR 3.7	Assessment	Allow a user to visualize a periodic assessment produced thanks to the data that were previously inserted.

Table 5: Overview of Functional Requirements related to the Notification System provided to the User.

FR4	**Data Management**	
FR 4.1	Data Sources	Employing as main app data source Health Connect (Health data management and integration platform developed by Google), while still keeping Google Firebase as additional data source where needed.
FR 4.2	Health Data Source	The system must retrieve and display users' health data relating to steps and sleep on the home page, as well as their heart and lung data in health measures page.
FR 4.3	Google Firebase Data Source	The system must retrieve and display users' health data related to emotional state and food on the home page. Also weight, waist circumference, grip, balance and strength data are retrieved from this data sources and showed in the health measures page. Finally, lessons and quizzes are fetched and showed in the learn page, as well as profile and goals data are also fetched and showed into personal information page.
FR 4.4	Health Data Backup	The system must add the necessary logic to perform a backup of the users's health data.

Table 6: Overview of Functional Requirements related to the Data Management.

As far as concerns the non-functional requirements, we can classify them as follows:

NFR	**Type**	**Description**
NFR1	Reliability	The system should ensure at least 80% accuracy and functionality over the course of a year.
NFR2	Portability	The application must be capable of running on Android devices.
NFR3	Security	Robust login mechanisms should be implemented to protect user data and limit access only to authorized individuals.
NFR4	Usability	The application should be intuitive enough for users of all ages and skill levels, requiring minimal training.
NFR5	Data Privacy	User data must adhere to OAUTH for secure data handling.
NFR6	Performance	The app should smoothly load and handle user interactions within two seconds under typical conditions in order to achieve an optimal user experience.
NFR7	Interoperability	The application should seamlessly connect with third-party platforms like Health Connect, maintaining data accuracy and improving functionality.
NFR8	Localization	The app should support at least English and Italian languages, adapting content and formats (e.g., date, currency) accordingly.
NFR9	Modularity	The app's architecture should support modular development to facilitate future updates without impacting the entire codebase.

Table 7: Overview of Non-Functional Requirements related to the system.

2.2 System Design and Architecture

2.2.1 System Design

While designing the system, the main components have been chosen in order to provide a clear and intuitive application system, as well as a robust and secure backend.
For the frontend application part, cross-platform frameworks have been analyzed, having in mind the possibility to extend the application also to IOS platform. Flutter, React Native and Xamarin were considered, but Flutter has been chosen, thanks to the possibility to develop a single codebase for both Android and IOS, and also for his strong support by Google who created it, that makes it a stable and reliable system, as well as a big community and an official package repository where tons of libraries can be found. Considering the web application, among several frameworks available (like React, Angular, Vue.js) React has been chosen, given his diffusion, simplicity and modularity.
Regarding the backend part, different alternatives were considered, also based on the frontend and our requirements:

- **AWS Amplify**, set of tools and services provided by Amazon Web Services (AWS) to build secure and scalable mobile applications. It comes with a suite of services that streamline the development process, like authentication, storage, GraphQL and REST APIs as well as analytics.
- **Express**, fast and minimalist web framework built on top of Node.js, which is a javascript runtime.
- **Microsoft Azure**, cloud computing service that provides a wide range of cloud services, like analytics, storage, and networking. It allows developers to build, deploy, and manage applications through Microsoft-managed data centers.
- **Spring Boot**, a Java-based framework used to create stand-alone, production-grade applications, which simplifies the development of new applications by providing code and configuration through annotations.
- **Firebase**, comprehensive app development platform by Google that provides a variety of tools and services to help developers build high-quality apps quickly.

Among them, Express was immediately discarded despite its simplicity, because it mainly focuses an APIs, does not provide authentication and it is not as rich of services as the others, so it does not fit our needs. Spring Boot was also discarded, given the need to setup external service for authentication (like keycloak IAM) and maintain a server where the backend could run. AWS Amplify and Microsoft Azure were also closer to our needs, but Firebase was chosen, thanks to his Flutter support and seamless integration (since both made by Google) and thanks to the variety of services it provides: Cloud Firestore as a data source, the Authentication system, Cloud Storage system as well as a suite of services to monitor the application (such as Performance and Crashlytics).

2.2.2 System Core Architecture

Focusing on the system core architecture, it can consequently be summarized as follows:

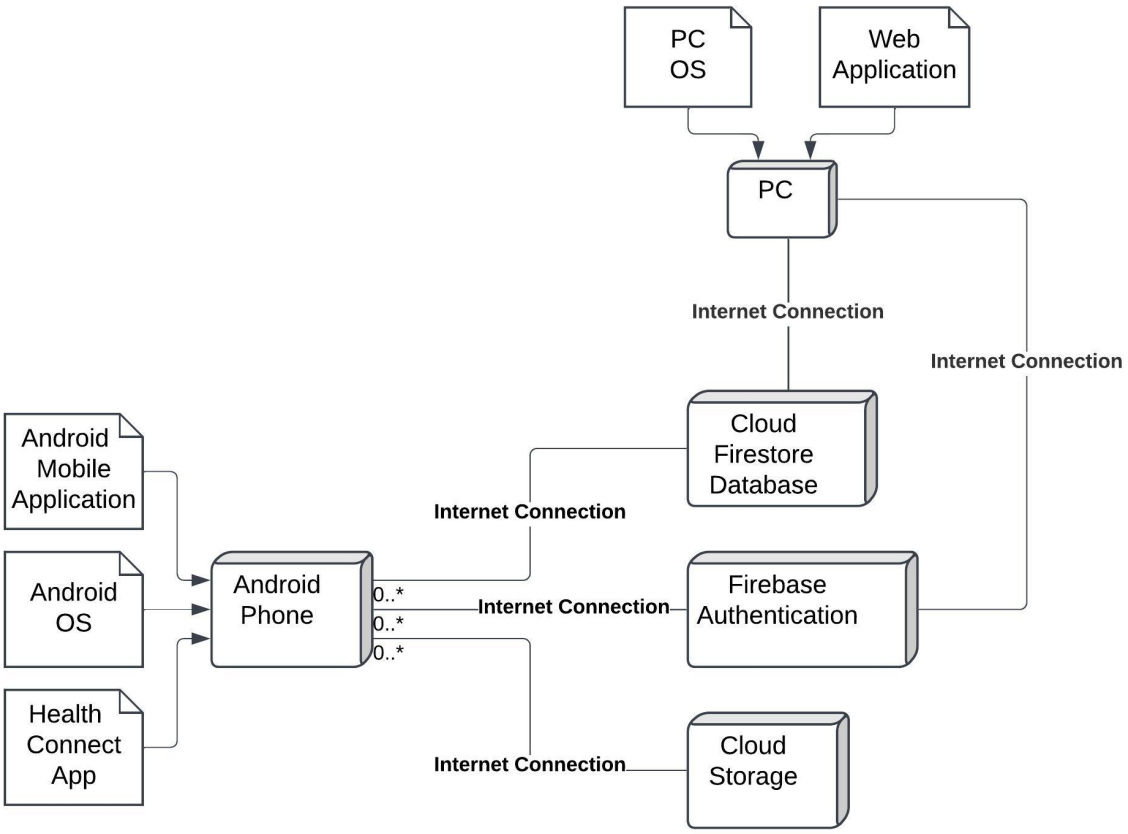


Figure 2: Overview of the System Core Architecture.

We can identify the core architecture with the following components:

- **Android Phone** that represent the physical smartphone, along with his operating system, our application installed and health connect installed, used to manage in an unified way the health data on Android.
- **Cloud Firestore Database** server, firebase service employed as a data source for the application logic, containing needed data, such as profile information.
- **Firebase Authentication** server, firebase service employed in order to perform and enforce authentication.
- **Cloud Storage** server, firebase service employed to store the backup of the health data for each user (see table 6 FR 4.4).
- A **Personal Computer** along with his operating system and a web application that allows the admin to authenticate and to interact with the Cloud Firestore Database, in order to modify the application parameters for users, as well as manage lessons and quizzes that are then showed to them.

In addition to this architecture, the system can be further enriched by integrating a wearable device, that can be a very useful tool to gather data, as previously explained in section 1.2.2. In this scenario the wearable device will be connected to the smartphone:

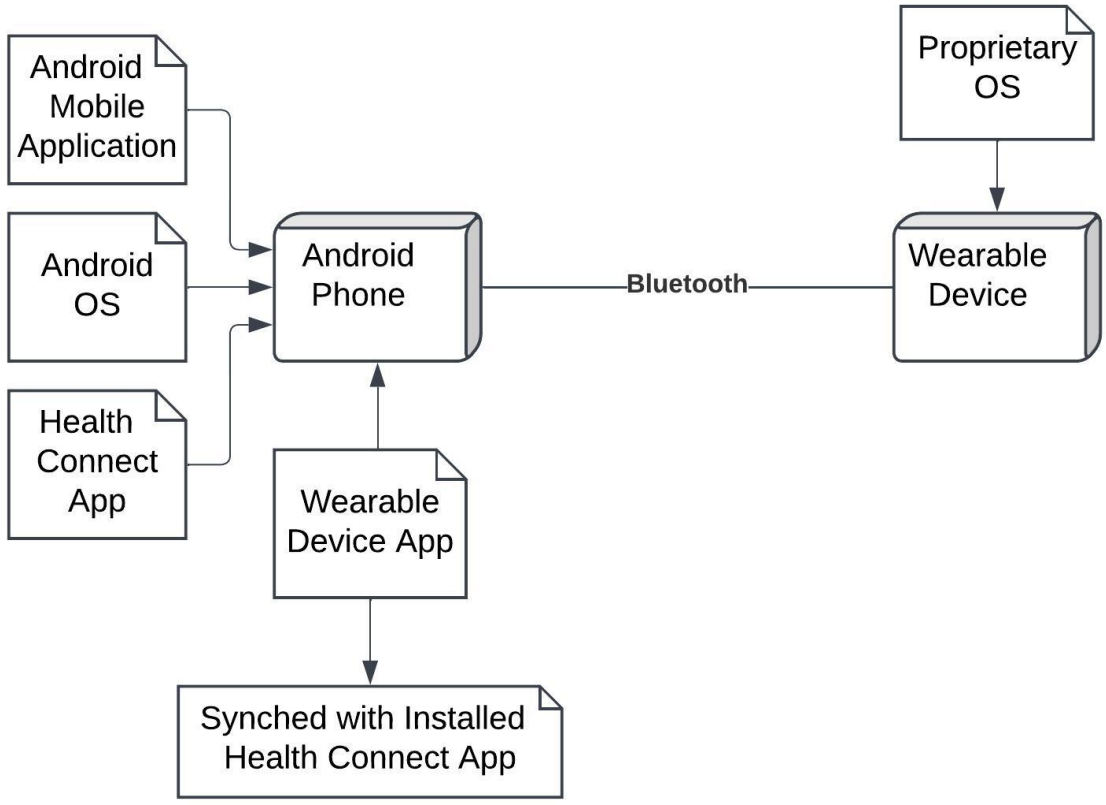


Figure 3: Overview of the Mobile System Architecture, enriched with a wearable.

In this case the architecture has been extended with the wearable device, along with his operating system, that through bluetooth connection can communicate with the smartphone. On the smartphone side, the wearable application will be able to retrieve the data from the wearable device to then sync with the Health Connect data management service.

3 Technology Stack

3.1 Frameworks

3.1.1 Flutter

Figure 4: Logo of the Flutter Framework.

In order to develop the mobile application, the flutter framework was employed. Flutter is an open-source framework created by Google. It allows to build applications that are natively compiled for mobile, web, embedded and desktop from a single codebase. The flutter code compiles into ARM, Intel machine code or JavaScript, depending on the machine for better performance on any device. It gives also the possibility to fine control the layout to be able to create a customized, adaptive design that look and feel great on any screen. It is also very productive from the developer point of view, thanks to the hot reload feature, that allows to see the changes in real-time without losing any state. The developer experience is also enhanced by automated testing and developer tools that further allow to build production-quality apps. Among these the most relevant are widget and layout inspectors, network and memory profilers, extensive docs, the pub package manager (see section 3.3.1) as well as lots of pre-built widgets and layouts in the SDK. The framework is widely known for its stability and reliability: infact, it is used by Google who made it but also trusted by other well-known brands around the world, and maintained by a community of developers, giving the possibility to collaborate on the open source framework, contribute to the package ecosystem on pub.dev, and find help whenever it is needed. The framework also offers a seamless integration with google services, allowing to streamline development and reach a wider audience. Among these services Firebase, Google Ads, Google Play, Google Pay, Google Wallet, Google Maps stand out. The framework is based upon Dart (see section 3.2.1) as programming language [11]. To practically use Flutter, just the Flutter SDK is needed, which includes the full Dart SDK, and then any text-editor or integrated development environment (IDE), combined with Flutter's command-line tools. However, most popular options that include also a guided setup are Android Studio, IntelliJ IDEA, and Visual Studio Code [12].

3.1.2 React

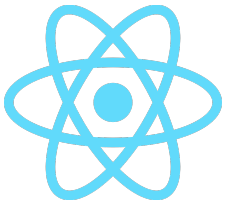


Figure 5: Logo of the React Framework.

In order to develop the web application, the react framework was employed. React is an open-source javascript library that aims to build user interfaces based on components. Maintained by facebook, his main advantage lies on the fact that it only re-renders those parts of the page that have changed, avoiding unnecessary re-rendering of unchanged DOM elements through the usage of the Virtual DOM, used as an in-memory data-structure cache to compute the resulting differences. The framework adheres to the declarative programming paradigm: the developer design views for each state of an application, and react will handle update and render of the components when some data changes. Among react main characteristics we have the components, which are modular and can be reused: a react application comes with many layers of components, rendered to a root element in the DOM using the react DOM library. These components are tipically written in JSX (an extension to the JavaScript language syntax). Also react hooks are a core feature, which are essentially functions that let developers "hook into" react state and lifecycle features from function components. They have specific rules (like they can be called only at the top level and from react function) and some built-in hooks are already provided, like `useState, useEffect`. Since react does not come with built-in support for routing, third-party libraries can be used to handle routing (define routes, manage navigation, and handle URL changes) as well as other client-side functionalities [13].

3.2 Programming Languages

3.2.1 Dart

Figure 6: Logo of the Dart Programming Language.

The programming language on which Flutter is based upon and that makes possible most of his features is dart. Dart is a client-optimized language for making fast apps on any platform. It offers the most productive programming language for multi-platform development, along with a flexible execution runtime platform. Dart is the foundation of Flutter, but also helps in several core delevoper tasks like formatting, analyzing and testing code. Among his features, the most interesting surely is the instant hot reload, that thanks to the Dart VM reflect immediately any code change, and the possibility to build application for different devices but from a single codebase.

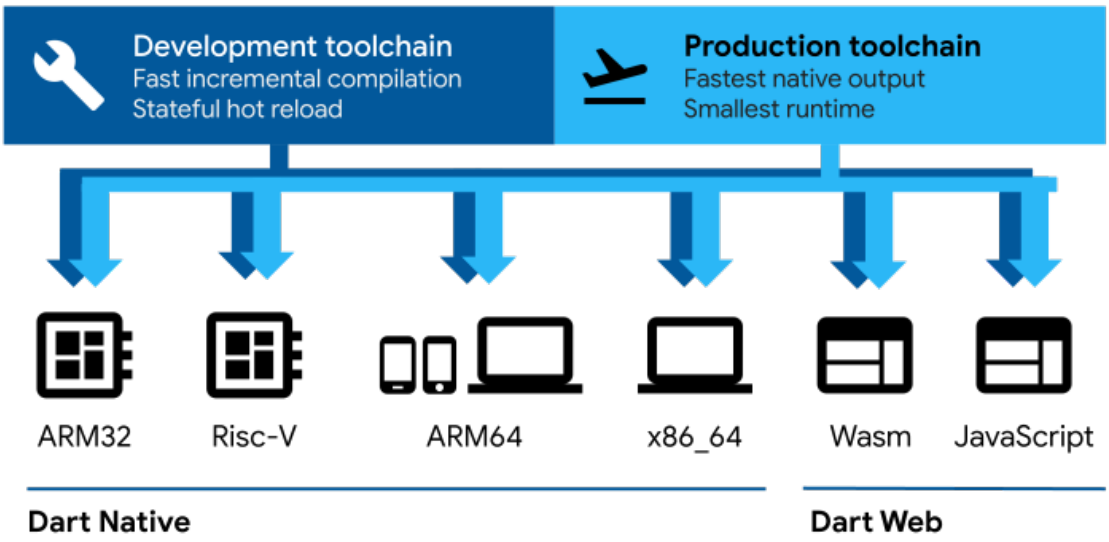


Figure 7: Overview of all the devices that dart is able to reach.

As we can see from fig. 7, the devices covered by a single code base range from embedded devices, to mobile and laptop devices (by using Dart VM with just-in-time (JIT) compilation and an ahead-of-time (AOT) compiler) and to web applications, with his web compiler that translate Dart into JavaScript or WebAssembly. Additionally, the language is type safe, so it uses static type checking to ensure that a variable's value always matches the variable's static type. This is done without sacrificing flexibility, since the language

still permits the use of a dynamic type combined with runtime checks, which can be useful during experimentation or for code that needs to be especially dynamic, through the usage of the `dynamic` keyword. The language has also built-in null safety, so values can't be null unless the programmer explicitly say they can be. In this way dart can protect from null exceptions at runtime through static code analysis. Unlike other null-safe languages, this non-nullability is also retained at runtime, so if dart determines that a variable is non-nullable, that variable can never be null. The language also comes with a mature and complete `async-await` syntax for UI with event-driven code, all paired with a concurrency system based on dart isolates (separate memory-isolated threads of execution used to achieve concurrency). Dart comes with a rich set of libraries, ranging from core libraries (`dart:core`) to the ones to parse json (`dart:convert`), for concurrency (`dart:isolate`), web (`package:web`) and so on [14].

3.2.2 Javascript

Figure 8: Logo of the Javascript Programming Language.

Javascript is a high-level, often just-in-time compiled language that represents the core technology to build web applications. Among his characteristics has dynamic typing, prototype-based object-orientation, and first-class functions. It is multi-paradigm, supporting event-driven, functional, and imperative programming styles. It comes with APIs to work with text, dates, regular expressions, standard data structures, and the Document Object Model (DOM). JavaScript is a single-threaded language, based on the event loop approach. The runtime processes messages from a queue one at a time, and it calls a function associated with each new message, creating a call stack frame with the function's arguments and local variables, which will shrink and grow based on the function's needs. When the call stack is empty and the function completed, javascript proceeds to the next message in the queue. In order to execute javascript code, a software component that executes the code called javascript engine is needed. These engines are tipically developed by web browser vendors: in fact, they tipically provide a runtime system, to enable interaction with a broader environment. The runtime system includes the necessary APIs for input/output operations, such as networking, storage, and graphics, and provides the ability to import scripts [15].

3.2.3 Groovy

Figure 9: Logo of the Groovy Programming Language.

Groovy is an object-oriented programming language, with dynamic typing for the Java Platform, alternative to the Java language. It can also be employed as a scripting language, intended to manage application automations for the Java Platform. The language also has features similar to languages such as Python, Ruby, Perl, and Smalltalk. Based on the Java platform, the language uses a Java-like syntax, based on curly brackets and is dynamically compiled in bytecode via the Java Virtual Machine. The Groovy compiler can be used to generate standard Java bytecode to interoperate seamlessly on any Java project. Among his features, the interoperability with java (java files are valid groovy files) makes him very versatile, and groovy code can also be more compact. It is characterized by object-oriented features (like operator overloading and polymorphic iteration) as well as safe navigation operator `?.` to check automatically for null pointers. The latest versions add the support to newer features like modularity, static compilation, type checking and multicatch blocks. Groovy has also native support of markup languages such as XML and HTML by using an inline Document Object Model (DOM) syntax. A Groovy script is fully parsed, compiled, and generated all before execution (similarly to Python and Ruby) and differently from Java, a groovy source code file can be executed as an uncompiled script under some circumstances[16]. Groovy is also used in the Gradle build system (see section 3.3.2), which is the official build system for Android applications, and it is used to define the build configuration and dependencies for the android part of the project.

3.2.4 Yaml for flutter pub package manager

Figure 10: Logo of the Yaml Language.

YAML is a data serialization language that is human readable. His common usage regards configuration files and applications where data is being stored or transmitted. The language targets many of the same communication applications as Extensible Markup Language (XML) but his minimal syntax differs from Standard Generalized Markup Language (SGML). The language also supports JSON style inside the same file. In Yaml custom data types are allowed but tipically they are not needed, since the language natively encodes scalars (strings, integers, and floats), lists, and associative arrays (so maps, dictionaries or hashes) all based on the Perl language. In particular, there is the possibility for both lists and associative arrays to form nested structures. It reuses escape sequences like in C and uses whitespace wrapping for multi-line strings like in HTML. Read and write support of Yaml is available for many programming languages and editors, where their extension is tipically `.yaml` or `.yml`. Regarding the syntax, whitespace indentation is used for denoting structure like in Python and typically uses UTF-32 encoding in order to have JSON compatibility. It is possible to comment a line with the `#` character, use strings with single or double quotes, and specify lists and arrociative arrays respectively through square brackets (`[...]`) or curly braces (`{...}`)[17]. In the project, Yaml is used to define the dependencies in the `pubspec.yaml` file, which is the configuration file for the Flutter pub package manager (see section 3.3.1).

3.3 Automation Dependencies Tools

3.3.1 Pub Package Manager

For managing flutter dependencies in our project, the pub package manager has been employed. It uses a `.yaml` file (pubspec.yaml) to list the dependencies and has a command-line interface that works both for flutter framework and dart programming language (in our case we exploited it with flutter). The syntax is relatively simple and it works by using the `pub` command followed by a subcommand.
There are several subcommands that are divided into three main categories, based on functionalities[18]:

- **Managing Package Dependencies:** in this category the most relevant are the `get` or `upgrade` commands, that respectively fetch or upgrade the dependencies that are used by a package. Also other commands are available, like `downgrade` to downgrade the dependencies at their lowest version for testing, or dually `upgrade` that upgrades the dependencies to their highest version.
- **Running command-line apps:** in this category fall the `global` command, which allows to make a package globally available, so that is possible to run scripts from his bin directory (the directory must also be added to the PATH variables).
- **Deploying packages and apps:** the `publish` command here is used to share developed dart packages with the community. This is done by uploading the package to the pub.dev website, acting as a package repository where all developer can download the published packages.

3.3.2 Gradle

Figure 11: Logo of the Gradle Automation Tool.

For managing dependencies/plugins in our project on the android side, gradle has been employed. Gradle is a build automation tool for software development which uses either Groovy or Java/Kotlin as language (in our case we leveraged on Groovy). It offers support for all phases of a build process, from compilation to verification, dependency resolving but also test execution, source code generation, packaging and publishing. Gradle is a multi-language tool, supporting languages like Java, Kotlin, Groovy, Scala, C, C++ and Javascript. Gradle operates with his own domain-specific language in contrast with the maven approach with XML. It has been designed to handle multi-project builds, that can be very large. Infact, it supports a series of build tasks that can run serially or in parallel, and build components can be also cached. Regarding his main features, gradle follows a convention (folder structure of the project, standard tasks and their order as well as dependency repositories) over configuration approach and all the build phases can be described in short configuration files. All conventions can still be overriden if it is necessary. One crucial gradle component is the plugin, that allow to integrate a set of configurations and tasks into a project, and can be either downloaded from a central plugin repository or custom-developed[19].

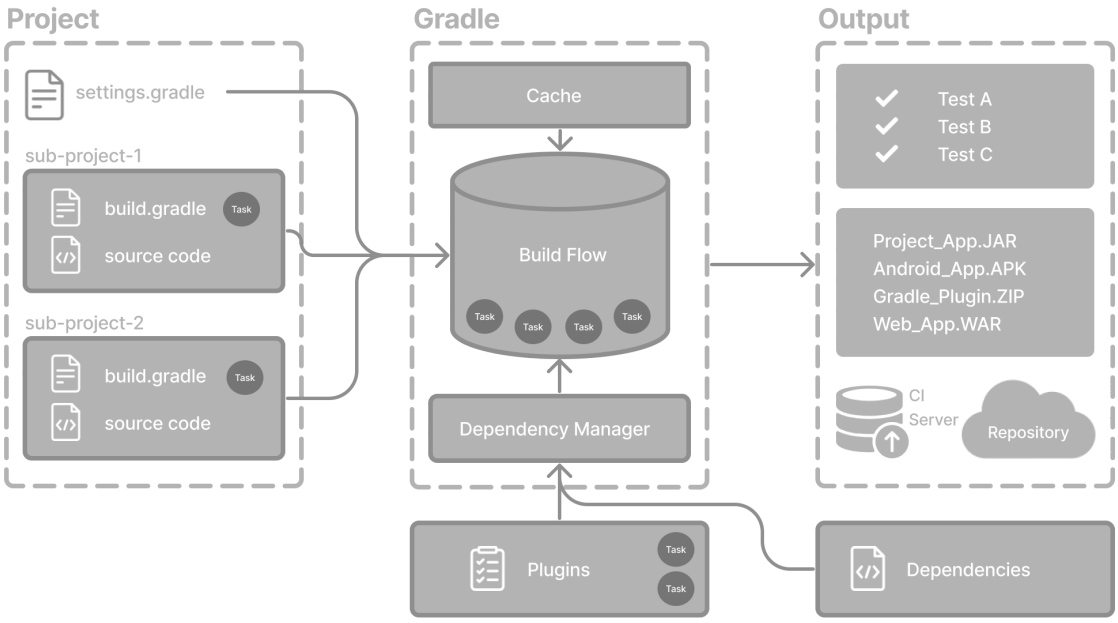


Figure 12: Gradle operating flow [20].

3.3.3 Npm

Figure 13: Logo of the NPM Automation Tool.

For managing dependencies/plugins in our project on the web side, npm has been employed. Npm is a package manager for the JavaScript programming language and the default package manager for the JavaScript runtime environment Node.js. It consists of a command line client interface and an online database of public and paid-for private packages, called the npm registry. The command line client interface allows users to consume and distribute JavaScript modules that are available in the registry and the available packages can be browsed and searched via the npm website. When used as a dependency manager for a local project, npm can install, in one command, all the dependencies of a project through the `package.json` file. Inside this file each dependency can specify a range of valid versions: this to allow developers to auto-update their packages and at the same time avoiding unwanted breaking changes. Npm also provides the `package-lock.json` file which has the entry of the exact version used by the project after evaluating semantic versioning in package.json. [21].

3.4 Integrated Development Environment

Figure 14: Logo of the Android Studio IDE.

As Integrated Development Environment for the mobile application, Android Studio has been employed. Android Studio is the official IDE for Android app development, but it seamlessly supports flutter through the flutter plugin. Based upon JetBrains' IntelliJ IDEA software, it inherites most of his features (like code completion, refactoring, debugging but also built-in tools and a plugin ecosystem). These features, combined with the fact that it is available for download for windows, macOs and linux, has helped to make it one of the most used IDE (together with VsCode) on this field. Android Studio is licensed under the Apache license but it ships with some SDK updates that are under a non-free license, so it is not completely open source. The supported languages are Java, Kotlin (the actual Google's preferred language for Android app development), C++ and more with extensions, such as Go and Dart[22].

3.4.1 Core features

Several are the core features, based on IntelliJ Idea and then further extended and enriched, that made this IDE a de-facto standard on mobile development:

The **Intelligent code editor feature** have been further extended to enhance the developers productivity. Particularly focused on the Android side, there is a clean *Project Structure*, where each project contains one or more modules (Android App, Library and Google App Engine Modules), each one with source code files and resource files. The Android App module in particular contains the manifest, the source code and then all the non-code resources in the `res` folder. Then, regardless of the mobile platform, Android Studio includes *Debug and profile tools* that help in debugging and improving the performance of code, including inline debugging and performance analysis tools. It is possible to leverage on inline debugging to enhance code walkthroughs in the debugger view with inline verification of references, expressions, and variable values. Also the Performance Profiler allows to easily track memory and CPU usage, find deallocated objects, optimize graphics performance, locate memory leaks and analyze network requests. The Memory Profiler can be used instead to track memory allocation and watch where objects are being allocated when you perform certain actions, since can be useful to optimize the app performance and memory use by adjusting the method calls. Also heap dump is allowed in a specific format, to analyze memory usage and find memory leaks. A solid Code Inspection system is also provided. At compile time, the IDE automatically runs configured lint checks and other IDE inspections to help you easily identify and correct problems with the code quality. The lint tool checks the project source files for potential bugs and optimization improvements for correctness, security, performance, usability, accessibility, and internationalization. In addition to that, IntelliJ code inspections validates annotations to streamline coding workflow. Finally, there is a Log System to view adb output and device log messages when building or running the app and also an Annotation System is supported to annotate variables, parameters, and return values to help find bugs, such as null pointer exceptions and resource type conflicts [23].

The **Flexible Gradle Build System** is employed, with specific Android capabilities provided by the Android Gradle Plugin. Leveraging on the Gradle flexibility allows us to easily manage dependencies, then also customize, configure, and extend the build process, as well as create multiple APKs. All this without modifying the app core source files, only

the gradle files (either using groovy or kotlin). By going deeper into the build system it is possible to clearly distinguish some of the main aspects: the *build types*, that define certain properties that Gradle uses when building and packaging your app (for example, the debug build type enables debug options and signs the app with the debug key, while the release build type signs your app with a release key for distribution). At least one build types must be defined and the IDE creates debug and release build types by default; The *product flavors* that are different versions of your app (like free and paid one) that can be released to users: they can be customized to use different code and resources while sharing the common parts. The *dependencies* are managed through the local file system and remote repositories: this eliminates the need to manually search, download, and copy binary packages into the project. Related to the APKs, the *Code and resource shrinking tool* allow to shrink code and resources by using its built-in shrinking tools and applying the appropriate set of rules: as result the APK size may be reduced significantly. Finally, the *multiple APK support* allows to automatically build different APKs that contains only the code and resources needed for a specific screen density or Application Binary Interface (ABI)[24].

The **Android Emulator** simulates Android devices on the computer so that it is easy to test an application on a variety of devices and Android API levels without needing to have each physical device. In most cases, the emulator is the best option to test an application on Android (alternatively it is possible to deploy the application on a physical device). Using the Emulator offers *flexibility*, since it is able to simulate lots of devices and comes also with predefined configurations for Android phone, tablet, Wear OS, Android Automotive OS, and Android TV devices. It also offers *high fidelity* because it offers almost all capabilities of a real android device, such as phone calls, messages, location, play store, networks, sensors and much more. Finally, on some aspects (like data trasfer) it is possible to gain *speed*, since data tranfer is higher to the emulator compared with a real device speed on USB. Each instance of the Android Emulator uses an Android virtual device (AVD) to specify the Android version and hardware characteristics of the simulated device and each AVD work as a separate device (it is treated independently from the others). Each AVD's data is stored inside a specific directory (directory then used to load the data when booting it up). It is also possible to test an application with WearOs

Devices by using the Wear OS pairing assistant that provides a step-by-step guide through pairing Wear OS emulators with physical or virtual phones directly in the IDE. Use the emulator is pretty simple: it is possible to simulate the touch with the mouse in the same way and use the provided buttons on the emulator panel for additional functionalities[25].

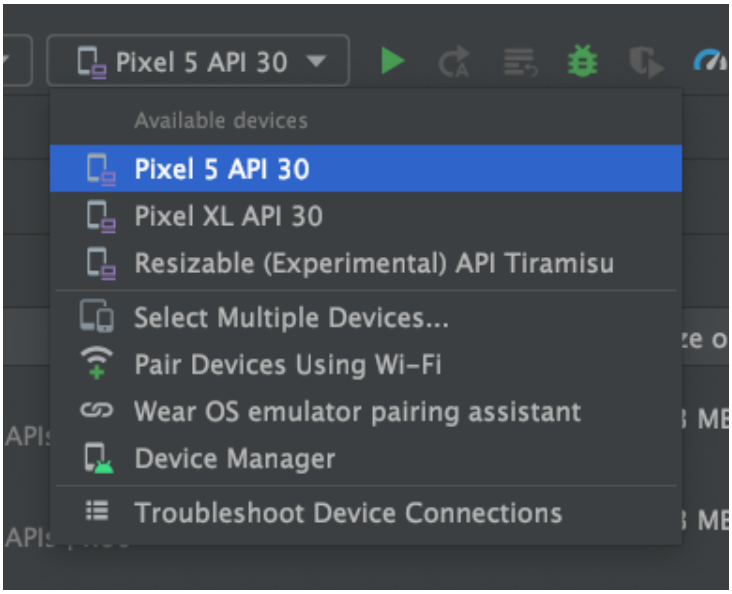


Figure 15: The target device menu[25].

The **APK Analyzer** gives more insight into the APK/Android Bundle composition once the build has been completed. By leveraging on this tool it is possible to reduce the debugging time related to DEX files and resources in the app and help reduce the APK size easily. APKs are files that follow the ZIP file format, and the APK Analyzer *displays each file or folder* as an expandable entity that can be used to to navigate into folders. For each entity three metrics are shown: Raw File Size (entity contribution to the total APK size), Download Size (estimated compressed size of the entity as it would be delivered by Google Play) and the percentage of Total Download Size (how much that entity impact on the overall APK size). The tool also allow to *view the AndroidManifest.xml* file, to understand any changes that might have been made to your app during the build. For example, since product flavors and libraries have their own manifest file, this is then merged with the application and the tool helps in visualizing it. It also provides some lint capabilities, with warnings or errors that appear in the top-right corner. The APK Analyzer also provides a *DEX file viewer* that allows to see underlying information in the DEX files (class, package, total reference, and declaration counts), which can assist in

deciding whether to use multidex or how to remove dependencies to get below the 64K DEX limit (Also the possibility to Filter the DEX file tree view is provided). Through the APK Analyzer there is also the possibility to *see code, resource entities and textual/binary files* inside the final APK, since during the build process shrinking rules can alter the code, and image resources can be overridden by resources in a product flavor. Finally, the tool allows to *compare different files*: specifically, the tool allows to compare the size of the entities in two different APK or app bundle files, and this is particularly useful to understand why the app increased in size compared to a previous release[26].

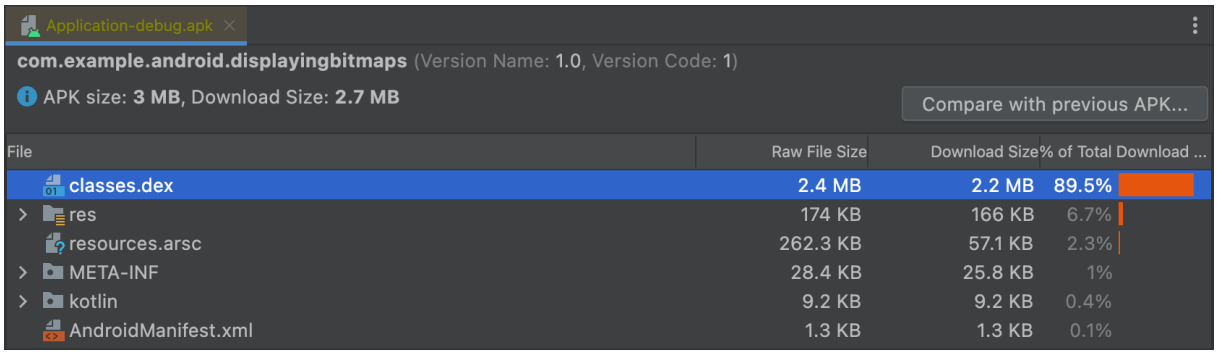


Figure 16: File sizes in the APK Analyzer[26].

3.4.2 New features

As a result of the massive involvement of artificial intelligence and cloud computing, by moving all into the cloud, also Android Studio has adapted by integrating two new major features into the IDE[27]:

- **Gemini**, coding assistant powered by artificial intelligence, that can understand natural language. Helps into achieving more productivity by generating code, finding relevant resources, learning best practices. However it has to be said that like any AI model, it might provide inaccurate, misleading, or false informations while presenting them confidently.
- **A Web Version** of the IDE (in early access preview), that leverages on IDX (web-based workspace for full-stack application development made by google itself). It could be used as a convenient way to open up samples but also to open an existing project on Github inside the browser.

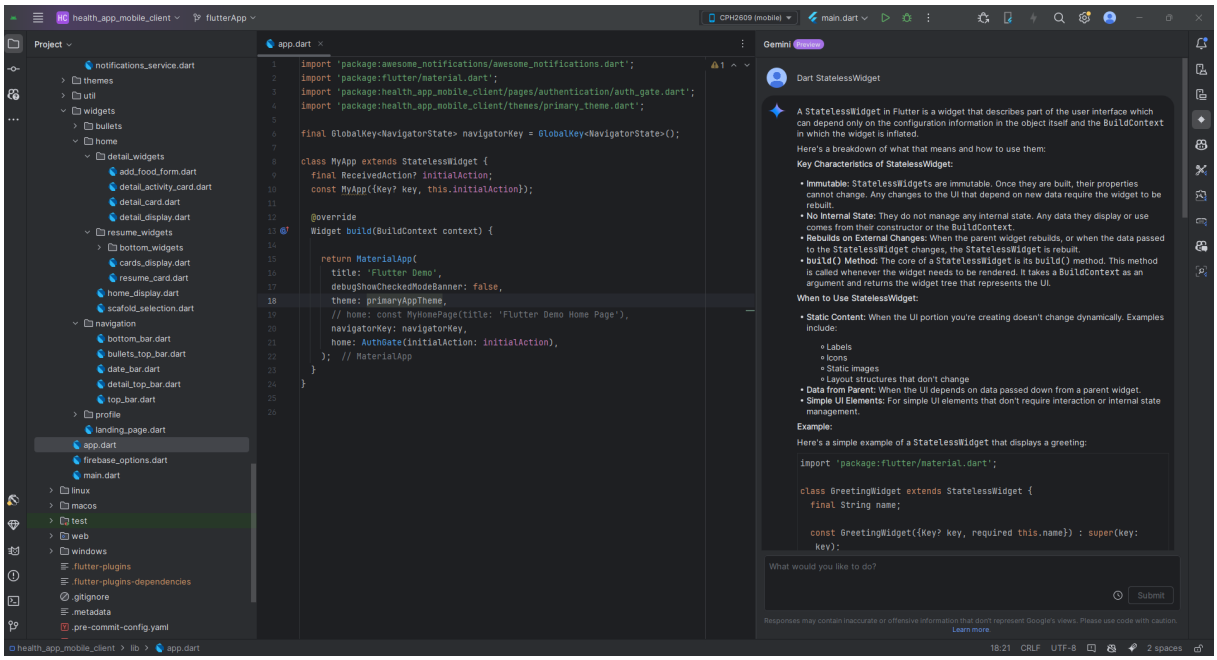


Figure 17: Example of the Gemini feature usage through the tab in android studio.

3.5 Code Editor

Figure 18: Logo of the VsCode Code Editor.

In order to develop the web application part of the system, VsCode was employed. VsCode is a fully-customizable code editor that streamline the development experience with features like IntelliSense, debugging and web version. It allows to install extensions to add additional services and customize itself. VsCode is fully customizable, with the possibility to change theme colors, defining customization profiles and easily switch between them. Also sync settings across VsCode instances on different devices is possible. It supports almost every major programming language and given to his lightweight nature and portability, can run also on the web browser. It comes also with a built-in terminal and git version control support, and more recently also artificial intelligence support was added through the copilot extension, that allows to edit multiple file through copilot, given a description of what we want to build, as well as giving code suggestions, that can be either accepted or rejected [28].

3.6 Google Firebase

Google Firebase is a backend comprehensive solution that allows to build, deploy and run an application. It comes with already-made extensions for common use cases, and it is easy to integrate on IOS, Android, and Web. It is a platform developed by Google, composed of several services, each one with a specific purpose: the employed services used for satisfy the system requirements will be covered.

3.6.1 Authentication Service

The Firebase Authentication service allows to perform a simple, multi-platform sign-in by providing backend services, easy-to-use SDKs, and ready-made UI libraries to authenticate users. It provides an easy authentication system by supporting several platforms and authentication methods (email and password accounts, phone auth, as well as Google, Apple, Facebook and more). It also simplifies the sign-in and onboarding experience for end users thanks to FirebaseUI Auth component, that implements the best practices for authentication on mobile devices and websites. It allows to setup a working authentication system for an application in under 10 lines of code, including account merging, and applies Google's internal expertise into adopting a comprehensive security system [29].

3.6.2 Cloud Firestore Database

The Cloud Firestore Database service is a NoSQL document database that allows to store, sync, and query data for an application on a global scale. Given the fact that is a NoSQL database, it lets structure the data freely with collections and documents and easily retrieve them by using expressive queries. It allows to build serverless app through a mobile and web SDKs and a comprehensive set of security rules, but it still provides support for traditional client libraries like Node and Python. It comes with a notification system that communicates data changes to easily build collaborative experiences and realtime apps, and also with an offline mode so that users can change their data at any time. The system (powered by Google's storage infrastructure) is fully scalable, allowing to focus on the application without worrying about servers and consistency. Finally, the declarative security language and the integration with firebase authentication allows to setup a strong user-based security [30].

3.6.3 Cloud Storage

The Cloud Storage is a powerful, simple, and cost-effective service that allows through the client SDKs to store images, audio, video, or other user-generated content. Is designed to store and serve contents quickly and easily. It is a scalable service, allowing to effortlessly grow from prototype to production and automatically scale up computing resources with cloud functions. Also with this service, a strong user-based security is enforced with an integration with firebase authentication through the SDK APIs for Cloud Storage [31].

3.6.4 Performance Monitoring

The Performance Monitoring service, available both for mobile and web applications, allows to measure app network requests, screen rendering times, as well as custom and automated traces to gain insights of the app performance. This tool allows to keep track of the performance on different app versions as new features are delivered or configuration changes are made, with a dashboard that makes it easy to focus on the most important metrics. A monitoring of data like app version and device allows to understand how the app performs from the users' point of view. There is the possibility to closely monitor HTTP/S requests by analyzing response time, success rate as well as payload size, particularly useful for critical requests. Thanks to that, understand where performance issues take place and address them is easier [32].

3.6.5 Crashlytics

The Crashlytics service, integrable with mobile and unity apps, allows to obtain a real time crash, error report and analysis to help keep the apps running flawlessly. The tool starts capturing crashes and groups them into manageable issues based on the impact on real users,so that it is easier to prioritize what to fix first. AI-powered insights helps understand why a crash happened and help to get to the root cause faster. Crashlytics works seamlessly with industry-standard tools including Jira, Slack, BigQuery, and with integrations for Android Studio it is possible to view the data directly in the IDE, making it easy to debug crashes. Contextual informations about the crashes are displayed, like the timeline of events leading up to app crashes to quickly reproduce bugs and uncover the root cause [33].

3.7 Dependencies

This section will cover the major dependencies that have been used in the project, with a focus on background features and firebase.

3.7.1 health

The health library was used to fetch health data from the devices to successfully use them. This library is a wrapper for Apple's HealthKit on iOS and Google's Health Connect on Android, and enables reading and writing operations from/to Apple Health and Google Health Connect. All the library functionalities are managed through a singleton `Health` instance, which supports lots of operations.
However, for our purposes, we mainly exploited:

- Permission handling operations to access health data, through the `hasPermissions` method to check the permissions and the `requestAuthorization` method to request them.
- Read operations (through the `getHealthDataFromTypes` method) to read the needed health data.

No write operations have been adopted, as the wearables or applications adopted by the users are synchronized and write data directly to Google's Health Connect, since it constitute the standard: from it health will then retrieve the data. The employed version is the 11.1.0 [34].

3.7.2 workmanager

The workmanager library, used to perform the health data backup (see table 6 FR 4.4), is a powerful solution for managing background tasks in Flutter applications. It allows to schedule and execute tasks even when the app is not actively running, useful for tasks such as periodic data synchronization, notifications, and other background operations. The package provides a simple API to schedule background tasks at specific intervals or under certain conditions, with the possibility to specify retry policies for failed tasks, ensuring robustness in the face of temporary issues. These tasks are executed efficiently, having minimal impact on the device resources and battery life. The employed version is the 0.5.2 [35].

3.7.3 awesome_notifications

The awesome notifications library, used to implement the notification system for the users (see table 5), allows to implement custom local and push notifications on Flutter with real-time events. It allows to keep the user engaged, thanks to the possibility of creating notifications with different layouts, images, sounds and so on. Real-time events are delivered at Flutter level code, leaving the possibility to the programmer to handle them. It gives the possibility of scheduling periodic or single notifications, is easy-to-use and highly customizable, including also the possibility to translation (see table 7 NFR8). The employed version is the 0.10.0 [36].

3.7.4 l10n

The l10n library, used to implement the localization of the application (see table 7 NFR8), allows to easily localize the application in multiple languages with a straightforward setup, through the definition of some configuration files and the `.arb` files, where the translations content are stored (one file per language).

3.7.5 firebase

Regarding firebase, given the variety of services that it offers, several dependencies have been employed in the project to cover all the functionalities needed:

- **firebase_core:** the base dependency for the firebase services, and it is required to use the Firebase Core API, which enables connection to multiple Firebase apps. The employed version is the 3.6.0.
- **firebase_auth:** dependency enabling classic authentication with email/password as well as identity providers like Google, Facebook and Twitter through the Firebase Authentication API. The employed version is the 5.3.1.
- **firebase_ui_auth**, a dependency that provides pre-built widgets, already integrated with the variety of the Firebase Auth providers. The employed version is the 1.10.0.
- **firebase_ui_oauth_google**, a dependency that allows to authenticate with Google through Oauth protocol, by specifying it as authentication provider. The employed version is the 1.12.14.
- **firebase_storage:** dependency for Firebase Cloud Storage service, allowing to easily store and retrieve files from it through the Firebase Cloud Storage API. The employed version is the 12.3.4.
- **firebase_crashlytics:** dependency for Firebase Crashlytics that allows, through the Firebase Crashlytics API usage, to report uncaught errors to the Firebase console. The employed version is the 4.3.3.
- **firebase_performance:** dependency for Google Performance Monitoring for Firebase, allowing to monitor traces and HTTP/S network requests and their parameters (like response time) through the Firebase Performance API. The employed version is the 0.10.1+3.

Regarding the **web application dependencies**, only the **firebase** [37] dependency has been employed, giving access to both authentication and cloud firestore database, in order to perform the operations. The employed version is the 11.3.1.

4 System Implementation

This section will discuss about the system implementation by firstly focusing on the backend part of the mobile application, moving then to consider also parameters editing, so the web application part that allows to do that.

4.1 Firebase

4.1.1 Firebase Authentication

Setting up the Authentication was pretty straightforward, as also previously explained (see section 3.6.1). All it took was going to the firebase console in the project settings and register both the mobile application and the web one. After that, in the authentication tab the desired Sign-in providers have been manually enabled (in our case email/password and Google). The firebase platform and the underlying google structure handled the rest by generating both the API keys to access the firebase services and the OAuth 2.0 clients to authenticate the users.
In case of the email/password authentication approach, also email templates have been employed:

- **Email address verification:** once a user signs up, a confirmation email is sent to verify his registered email address.
- **Password reset:** in case a user forgets his password, he can request a password reset email.
- **Email address change:** in case a user change his email address, a confirmation email is sent to the original address, so that the user can review the change.

At this point for the **mobile client** side the firebase dependencies (in particular firebase_ui_auth and firebase_ui_oauth_google) have been employed to perform the authentication and the sign-in process. The already well made components provide the possibility to sign-in or eventually register in case the user is not already registered, as well as manage the user profile once logged in. They are also highly customizable, allowing us to change the UI to match the app's theme. As additional logic, only the authentication state had to be handled in order to redirect the user to the home page once the authentication is successful. This has been done through a `StreamBuilder` widget, having as source stream `FirebaseAuth.instance.authStateChanges()`, that notifies about the user's sign-in state. In this way, any change is listened and if the user signs in or out the interface is updated coherently.

About the **web application client** instead, no already-made components were available. The react `createContext` and `useContext` hooks have been employed to handle the authentication state: it was done by using a context in the react application and providing it at the root level so that the whole application could access the authentication state. This state is updated after performing authentication operations and the components that depend on it are updated accordingly, allowing to display the edit parameters screens once authentication is performed. A profile management screen like the mobile one has not been implemented, since it is an admin interface.

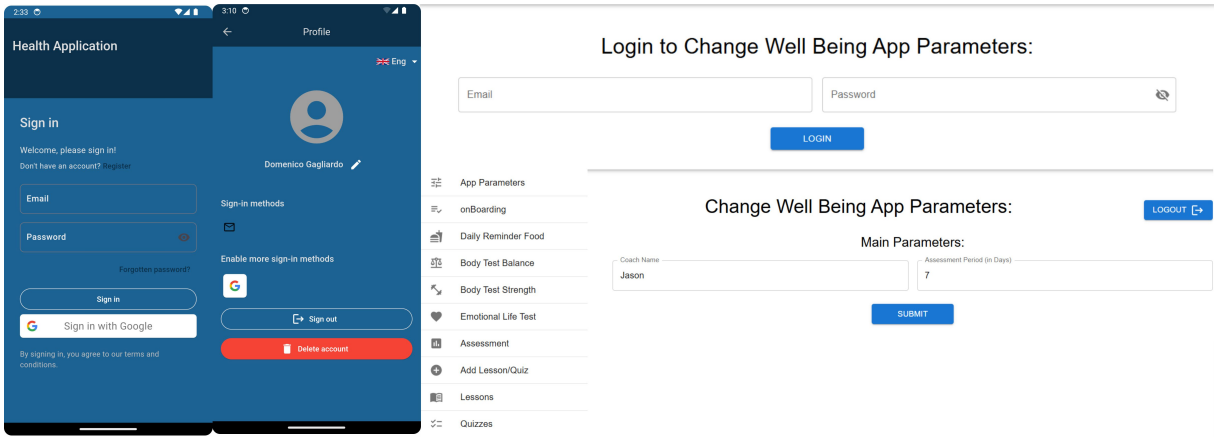


Figure 19: Authentication Screens of the mobile client (sign in and profile management) on the left and of the web client on the right.

4.1.2 Cloud Firestore Database

Taking into account the data-related requirements (see table 6), the database has been implemented accordingly by storing all the data in the most efficient way possible.
The database structure is as follows:

- The **users** collection, used to manage main users informations, with each document having the following fields:
  - The `UID` field, that uniquely identifies the user.
  - The `goals` field (composed of `calories`, `sleep` and `steps` as inner fields) that represents the user's goals.
  - The `language` field, that represents the app language set and preferred by the user.
  - The `metrics` field (composed of `birthDate, height, nickname, sex, waist circumference, weight` and `wearable` as inner fields) that represents the user's personal information.
- The **user_data** collection, used to manage additional users informations, with each document having the following fields:
  - The `userId` field, that links those data with the corresponding user.
  - The `current_notification` field that used in combination with the `notification_counter` field allows to implement the notification logic (see section 4.10 for more details).
  - The `lastBackupDate` field, that saves the last time that the user performed a backup of his data.
  - The `weightData` array field that stores the user's weight information, implementing an history of this metric, with the most recent present in the `users` collection for each user. Each element of the array has the `date` and `weight` fields.
  - The `waistCircumferenceData` array field that stores the user's waist circumference information, implementing an history of this metric, with the most recent present in the `users` collection for each user. Each element of the array has the `date` and `waist circumference` fields.
  - The `emotionalData` array field that stores the user's emotional information. Each element of the array has the `date`, `negative` and `positive` fields.

  - The `bodyTestBalanceData` array field that stores the user's body balance information. Each element of the array has the `date`, `leftLeg`, `rightLeg` and `tandemWalk` fields.
  - The `bodyTestStrengthData` array field that stores the user's body strength information. Each element of the array has the `date`, `absCount`, `gripTest`, `pushUpCount` and `squatCount` fields.
  - The `completedLessons` array field that stores the user's completed lessons information. Each element of the array is the `lessonId` to which it refers to and a `completedPills` boolean array field used to store how many pills of the lesson have been read (true) or not (false).
  - The `completedQuizzes` array field that stores the user's completed quizzes information. Each element of the array is the `quizId`: if present means that user has completed that quiz.
- The **foodRecords** collection, used to manage the food informations for the users, with each document having the following fields:
  - The `userID` field, that links those data with the corresponding user.
  - The `date` field, storing the food entry recording date.
  - The `foodGroup` field, that identifies the type of food (like water, vegetables and so on).
  - The `amount` field, that represents the food quantity (e.g. 250 ml if a liquid food or 250 gr if a solid food).
- The **lessons** collection, used to manage the available lessons for the users, with each document having the following fields:
  - The `id` field, that uniquely identifies the lesson.
  - The `quizId` field, that links the quiz related to that lesson.
  - The `title` field, that represents the lesson title.
  - The `titleIta` field, that represents the italian lesson title.
  - The `pills` array field, that represents the list of pills that the lesson is made up of, where each element is a single pill.
  - The `pillsIta` array field, that essentially is the italian version of the above `pills` field.

- The **quizzes** collection, used to manage the available quizzes for the users, with each document having the following fields:
  - The `id` field, that uniquely identifies the quiz.
  - The `questions` array field, that represents the questions of that quiz. Each questions has the `questionText`, `correctAnswer` fields and `possibleAnswers` array field, that respectively represents the question, the correct answer and the list of possible answers, where each element is an answer.
  - The `questionsIta` array field, that essentially is the italian version of the above `questions` field.
- The **notifications_text** collection, used to manage the mobile application parameters displayed for the users and editable by the admin through the web application. It is composed of 3 documents:
  - The first document, that contains two fields called `coach_name` and `assessment_period` that represents the name of the app assistant and the length of the assessment perios (in days) used for the notification cycle (see section 4.10 for more details).
  - The second document, that contains several strings fields used to customize the onBoarding experience of the user as well as the data insertion operations. We have up to three fields for the onBoarding (`onBoarding1,onBoarding2,onBoarding3`), 7 fields for the body test balance data insertion (`body_test_balance1` up to 7), 6 fields for the body test strength data insertion (`body_test_strength1` up to 6), one field for the emotional data insertion (`emotional_life_test`), one field for the assessment (`assessment`) and one field for the food insertion (`daily_reminder_food`).
  - The third document, that similarly to what has been done with the lessons and quizzes, contains the italian version of the second document.

4.2 User OnBoarding and Registration

As explained before (see section 4.1.1), the user registration and sign-in process is mainly managed by the firebase authentication service. However, some additional initialization steps are required, in order to setup all the documents needed in the database collections for the user, once is registered for the first time. When the user registers for the first time and it interact with ui authentication components of firebase, instead of displaying the main page of the application, the user is redirected to the onBoarding page. Here essentially almost all the information needed to initialize the user document inside the users collection (see section 4.1.2) are asked to be inserted. The proper validation is enforced in case of input errors, preventing the user from proceeding and when the user inserted all the data, all the needed documents and fields for that user are created and the home page is displayed.

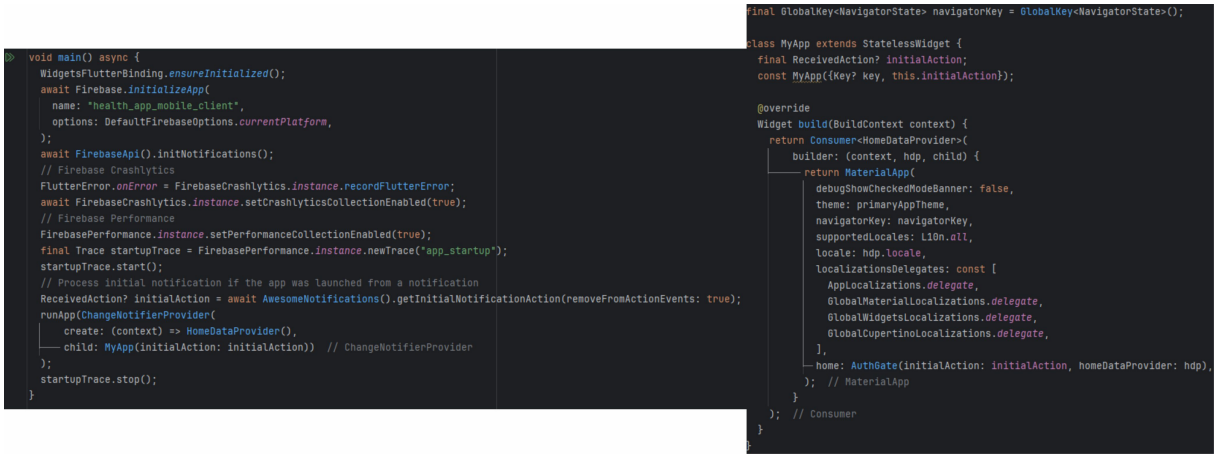
```
void setUserMetrics(Map<String, dynamic> metrics) async {
  final db = FirebaseFirestore.instance;
  final user = FirebaseAuth.instance.currentUser;
  final goals = {"sleep": "400", "calories": "1500", "steps": "8000"};
  await db.collection("users").doc(user!.uid).set({"language" : locale.languageCode}, SetOptions(merge: true));
  await db.collection("users").doc(user.uid).set({"metrics" : metrics}, SetOptions(merge: true));
  await db.collection("users").doc(user.uid).set({"goals" : goals}, SetOptions(merge: true));
  final userData = {
    "current_notification": "body_test_balance",
    "bodyTestStrengthData": [],
    "bodyTestBalanceData": [],
    "completedLessons": [],
    "completedQuizzes": [],
    "weightData": [{"weight": metrics["weight"], "date": DateFormat('yyyy-MM-dd').format(DateTime.now())}],
    "waistCircumferenceData": [{"waist circumference": metrics["waist circumference"], "date": DateFormat('yyyy-MM-dd').format(DateTime.now())}],
    "emotionalData": [],
    "lastBackupDate": "",
    "userId": user.uid
  };
  await db.collection("user_data").add(userData);
}
```

Figure 20: Function Called at the end of the onBoarding process to setup all the needed documents/fields for the user.

Only the goals are not asked to be inserted in the onBoarding process to avoid make it too long and boring. They have a default value and can be edited in the personal information page at any time.

4.3 State Handling

All the mobile application states are kept in a class called `HomeDataProvider` that extends the `ChangeNotifier`, implementing the best practises regerding the flutter state management and allowing the interface to be updated accordingly whenever some state changes. It is instantiated at the root level of the application with the `ChangeNotifierProvider` widget that creates a `ChangeNotifier` and passed down to all the widgets that need to access the data. These widgets can easily access the shared instance by using a `Consumer` widget, that consumes the above `ChangeNotifierProvider` allowing to access the instance as well as the context and child. All the relevant states, including the ones to populate the charts (both coming from health as data source or the database) were kept in this class, ensuring a clean state management and an organized code.



```dart
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    name: "health_app_mobile_client",
    options: DefaultFirebaseOptions.currentPlatform,
  );
  await FirebaseApi().initNotifications();
  // Firebase Crashlytics
  FlutterError.onError = FirebaseCrashlytics.instance.recordFlutterError;
  await FirebaseCrashlytics.instance.setCrashlyticsCollectionEnabled(true);
  // Firebase Performance
  FirebasePerformance.instance.setPerformanceCollectionEnabled(true);
  final Trace startupTrace = FirebasePerformance.instance.newTrace("app_startup");
  startupTrace.start();
  // Process initial notification if the app was launched from a notification
  ReceivedAction? initialAction = await AwesomeNotifications().getInitialNotificationAction(removeFromActionEvents: true);
  runApp(ChangeNotifierProvider(
      create: (context) => HomeDataProvider(),
      child: MyApp(initialAction: initialAction))  // ChangeNotifierProvider
  );
  startupTrace.stop();
}
```

```dart
final GlobalKey<NavigatorState> navigatorKey = GlobalKey<NavigatorState>();

class MyApp extends StatelessWidget {
  final ReceivedAction? initialAction;
  const MyApp({Key? key, this.initialAction});

  @override
  Widget build(BuildContext context) {
    return Consumer<HomeDataProvider>(
        builder: (context, hdp, child) {
          return MaterialApp(
            debugShowCheckedModeBanner: false,
            theme: primaryAppTheme,
            navigatorKey: navigatorKey,
            supportedLocales: L10n.all,
            locale: hdp.locale,
            localizationsDelegates: const [
              AppLocalizations.delegate,
              GlobalMaterialLocalizations.delegate,
              GlobalWidgetsLocalizations.delegate,
              GlobalCupertinoLocalizations.delegate,
            ],
            home: AuthGate(initialAction: initialAction, homeDataProvider: hdp),
          );  // MaterialApp
        }
    );  // Consumer
  }
}
```

Figure 21: HomeDataProvider instantiation at the root level of the app (left) and example of accessing the instance (right).

4.4 Home Page

The home page of the application was designed by allowing the user to have access to his steps, food, sleep and mood data, in which both the database and the health data sources have been employed. While health data are already gathered by the phone/wearable device and provided (is the case of steps and sleep), for food and mood there is also the possibility to let the user insert the data. Considering this aspect, for food and mood the database has been employed as data source instead. Also a `DateBar` component is present in this page as an utility component, easily allowing to change the selected date, as well as the time period (day,week,month) and view the corresponding data.

4.4.1 Health Data Source

As said before, for steps and sleep data, the health data source has been employed. For steps and sleep data, two different states are kept as lists (currentActivityDataPoints and currentSleepDataPoints). When, through the health package, the data are retrieved based on the selected date these states are set, and the charts are updated and populated accordingly. For these two metrics, also a chart threshold is set through the users goals, so that the user can see if he is reaching his goals or not. In case the goal is met or also surpassed, the bar appears yellow and with a dotted border, so that the user can easily see the difference.

4.4.2 Database Data Source

For food and mood data instead the database has been employed. For food the approach is still the same, except for the data source: a list state is kept (currentNutritionDataPoints), data are fetched also based on the date and chart are updated. For mood instead simply the two positive and negative mood values are kept, computed as the average on the selected timespan (positiveMoodPoint and negativeMoodPoint). In case of food data the threshold like steps and sleep is available. For both food and mood, the user can insert the data through the app with dedicated forms. In case of food, the user can insert the food group and the amount, while for mood the user can insert the positive and negative mood values, according to the PANAS scale. The employed collections are `foodRecords` for food and `user_data` for mood (the `emotionalData` array field).

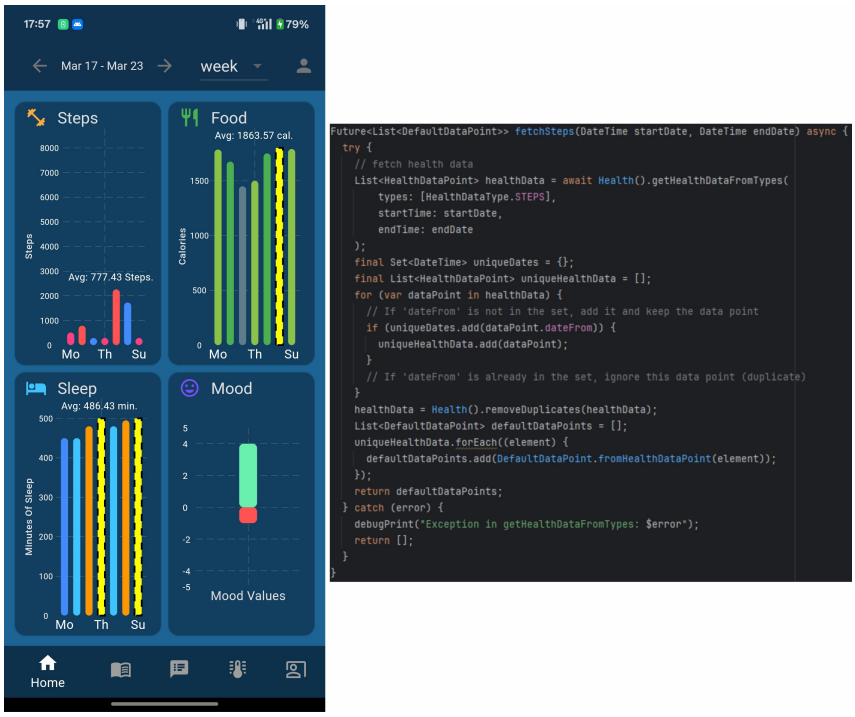


Figure 22: Home Page charts (left) and example of health data fetch for the steps (right).

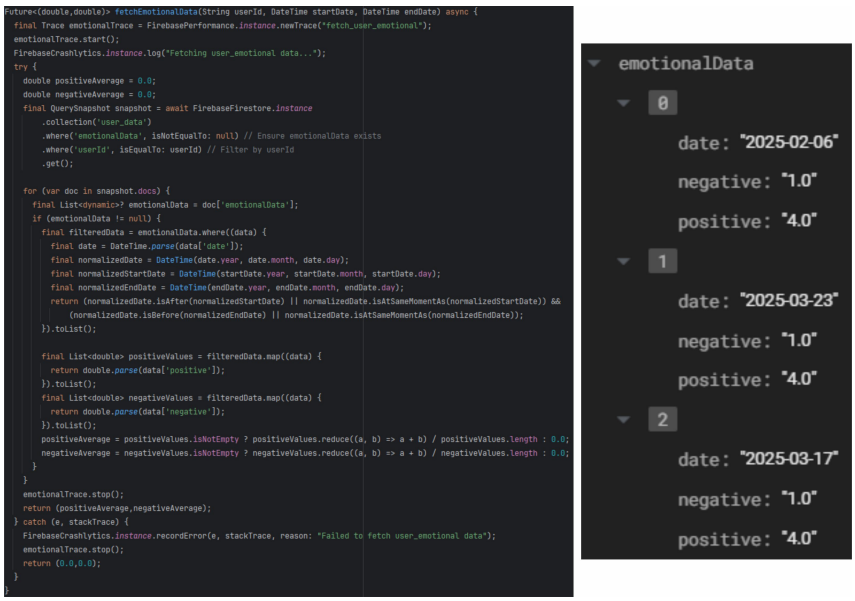


Figure 23: Example of database data fetch for the emotional data (left) and corresponding database fields (right).

In the food and sleep charts we can also see the overflow of the related goals in one particular day that makes the bar appear differently.

4.5 Health Measures Page

In the health measures page the user can see additional data about his health.
A tab subdivision has been made to obtain a clearer view of the data:

- **Lungs-related data tab**, composed of:
  - Oxygen Saturation chart, representing the oxygen saturation in the blood.
  - Respiratory Rate chart, representing the number of breaths per minute.
- **Heart-related data tab**, composed of:
  - Resting Heart Rate chart, representing the resting heart rate, measured in breaths per minute.
  - Heart Rate Variability chart, representing the variation in time between heartbeats.
- **Body-related data tab**, composed of:
  - Weight chart, measured in kilograms.
  - Waist Circumference chart, mesured in centimeters.
  - Body Balance chart, composed of the left leg, right leg and tandem walk tests, all measured in seconds.
- **Strength-related data tab**, composed of:
  - Grip Strength chart, measured in kilograms.
  - Waist Circumference chart, mesured in centimeters.
  - Body Strength chart, composed of the abs count, push up count and squat count, all measured in repetitions.

Also here the `DateBar` component is present to change the selected date, as well as the time period (day,week,month) and view the corresponding data.

4.5.1 Health Data Source

For the first two tabs (lungs and heart), the health data source has been employed. All those data are directly retrieved through the health package, similarly to the home page steps data. We have four list states (currentOxygenDataPoints, currentBreathRateDataPoints, currentHeartRateVariabilityDataPoints and currentRestingHeartRateDataPoints) that are set when the data are retrieved based on the selected date. Also in this case, the charts who depend on these states are updated and populated accordingly.

4.5.2 Database Data Source

For the last two tabs instead (body and strength), the database has been employed, particularly the `user_data` collection, that contains the array fields `weightData`, `bodyTestBalanceData`, `waistCircumferenceData` and `bodyTestStrengthData`. Except for the data source the approach is still the same: four list states (currentWeightDataPoints, currentWaistCircumferenceDataPoints, currentBodyBalanceDataPoints and currentBodyStrengthDataPoints) are kept, data are fetched based on the date and charts are updated. The user has also the possibility to insert the data with dedicated forms, giving the possibility to have an hystorical view of metrics like weight and understand eventual improvement or worsening.

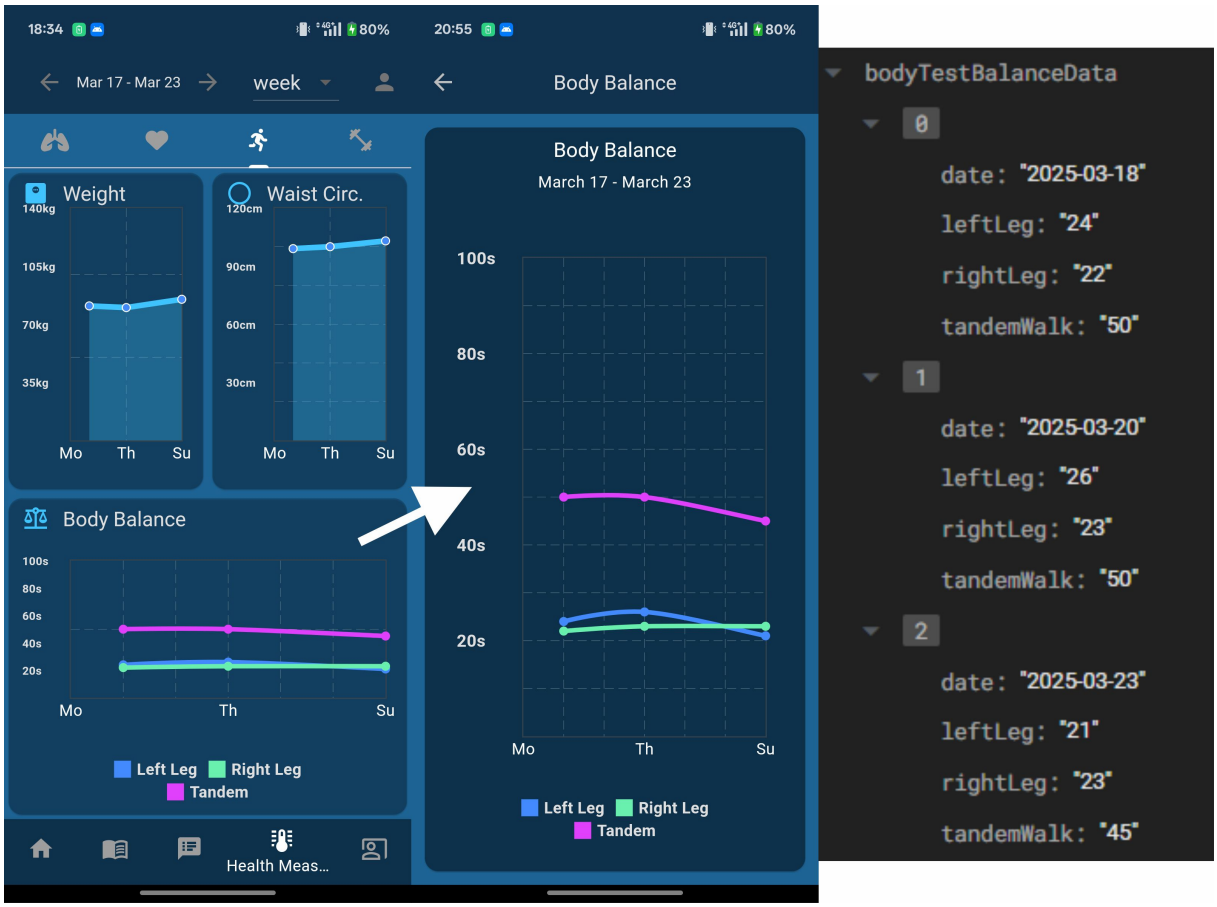


Figure 24: Health Measures Page on Body-related data tab (left), detailed view of the Body Balance chart (center) and his corresponding database fields (right).

4.6 Personal Information Page

In this page the user has the possibility to edit part of his metrics as well as editing his goals. Since those metrics are acquired in the onBoarding phase, they are all handled through the database and stored in the user-generated document inside the `users` collection, in particular in the `metrics` field.

Also in this case, in order to have a clearer view of the data, there is a tab split into:

- **Personal Measures tab**, composed of almost all the users metrics. Metrics not present in this page are the weight and the waistCircumference, since they can be added in the healthMeasures page, where in addition the historical view of them is displayed. The present metrics are freely editable at any time after user registration, except for the birthDate, only displayed for obvious reasons.
- **Goals tab**, composed of the users goals, freely editable at any time after user registration. The goals are used to set the threshold of the charts in the home page, so that the user can see if he is reaching his goals or not.

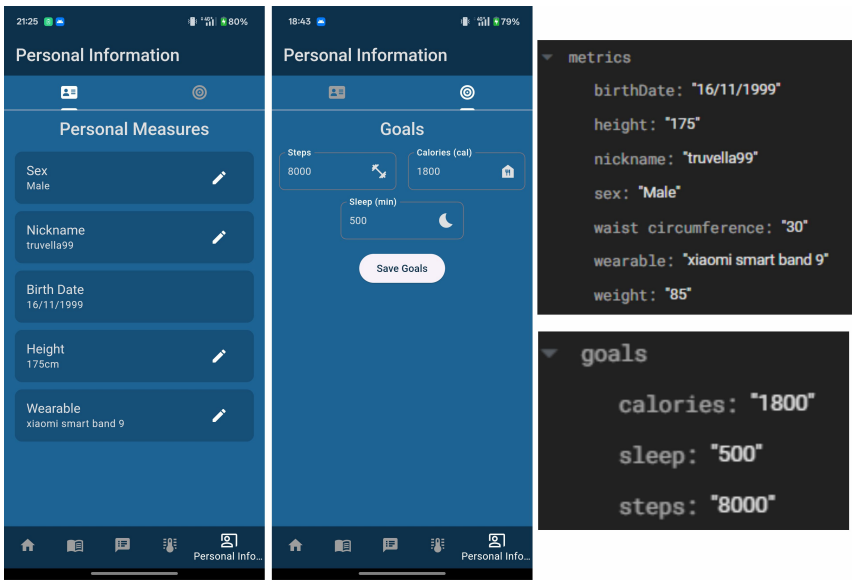


Figure 25: Personal Information Page with Personal Measures tab (left), Goals tab (center) and corresponding database field (right).

Both the metrics and goals are updated in the database when the user changes them by using the `SetOptions(merge: true)` method, so that only the fields that are changed are updated without affecting the whole document.

4.7 Learn Page

In the learn page the user has the possibility to browse the available lessons and learn new things. The lessons and quizzes are managed through the `lessons` and the `quizzes` collections, where the single lesson references the quiz through the `quizId` field. Other additional fields are also used to store information of the single user. Infact, each lesson is composed of several pills, and the `completedPills` field is employed (see section 4.1.2): when the user reads a certain pill of a lesson, the corresponding array field is updated with a true value, with an index correspondance between the pill and the boolean field in the array. In this way the user can see how many pills of a lesson he has read and how many are still to read, and only when all the pills have been read the whole lesson is marked as read. Regarding the quiz instead, to leave the user with more freedom, it can be taken at any time, but the quiz is marked as completed (the quizId is added to the `completedQuizzes` array field still in `user_data`) only if the user completes it with success. Both lessons and quizzes supports localization and have their italian traslation stored in the `pillsIta` and `questionsIta` fields respectively. The user will be able to see this version by changing the default language of the app.

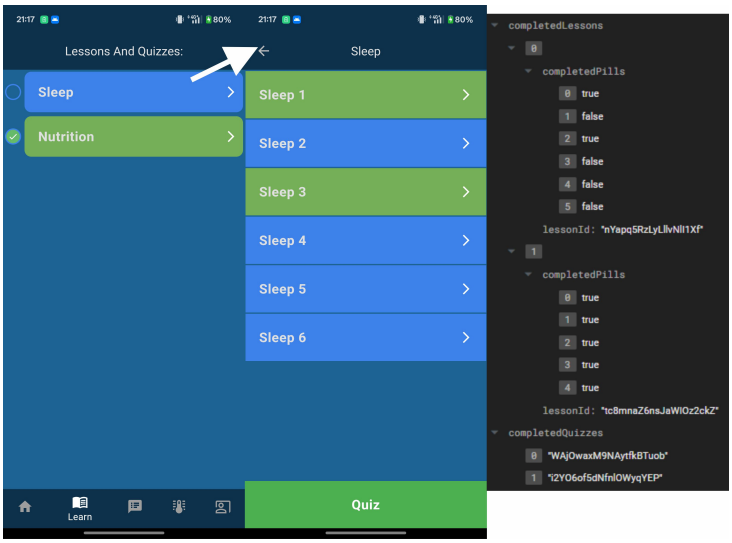


Figure 26: User View of the Lessons and Quizzes (left), of a lesson in detail (center) and the related database fields to handle pills reading (right).

In this example we can see the pills handling, since for the sleep lesson only pills 1 and 3 are read (same array fields set to true), while for the nutrition lesson all the pills are read. Both quiz however are completed with success and added to the `completedQuizzes` field.

4.8 MultiLanguage

Also the multilanguage feature has been implemented, in order to satisfy the localization requirement (see table 7 NFR8). Implementing this feature required additional fields on the database, as well as additional management on the application side. As explained previously (see section 4.1.2) an additional field was added for each user document inside the `user_data` collection called `language`, used to store the current language selected by the user (english by default). Also other fields were added to handle multilanguage: for the lessons and quizzes the `pillsIta` and `questionsIta` fields were added, while for the notifications an additional document in the `notifications_text` collection was employed, that contains the italian version of the parameters. The user will be able to see either the english or italian version by changing the default language of the app through a dropdown on the top right of the screen (see fig. 19 middle image), and changing his `language` value will imply a change of the whole app language.

As application side setup, the `l10n.yaml` file and `l10n` folder were created. The first file was used to configure the default directory where the `.arb` files (traslation files) will be stored (the `l10n` directory), as well as the default traslation file to pick (`app_en.arb`) and the output `.dart` file that contains the localization code. This because running the build with all the files configured will produce auto-generated locale output `.dart` files for all supported locales, that can be imported and then used. Inside the `l10n` folder the `l10n.dart` file was defined, which contains a class with an array field named `all`, containing all the supported locales (only en for english and it for italian), as well as the traslation files (`app_en.arb` and `app_it.arb`) that contains the traslation of the app strings in the two languages.

The multilanguage support was handled from the `MyApp` root component by setting the locale, defined as a HomeDataProvider state that can be changed at any time (see fig. 21 right side): this allowed to change the language also during the onBoarding process, so that the user can see the app in his preferred language from the beginning, and the language setting persists also after registration.

Once implemented that, all it took was defining the traslation strings in the `.arb` files with a key/value format, and referring them throughout the whole application. The `AppLocalizations.of(context)` method was employed to access the traslation strings, allowing to traslate the whole application in the selected language.

For example, given the `learn` key, having value Learn in english and Impara in italian, the `AppLocalizations.of(context)!.learn` will return Learn or Impara based on the selected language.

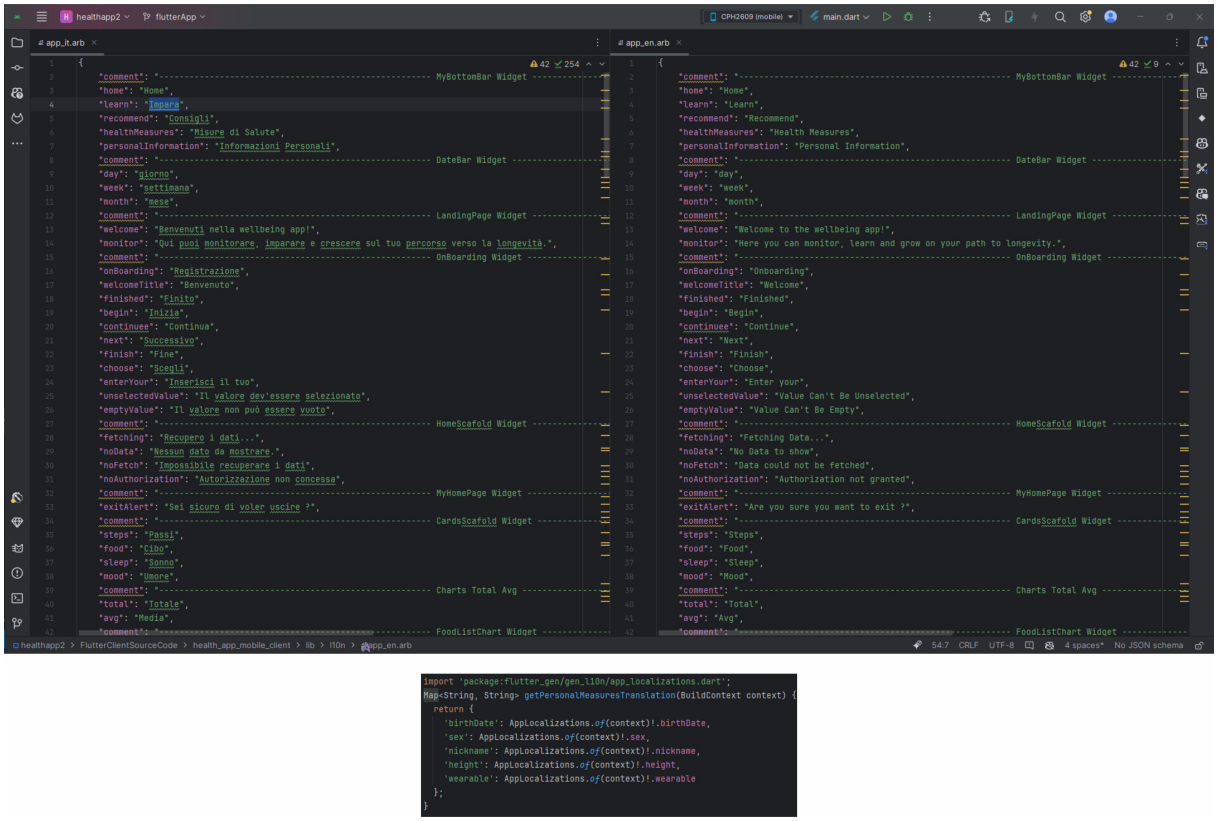


Figure 27: `.arb` files structure definition (top) and usage example of the multilanguage feature (bottom).

4.9 Health Data Backup with Cloud Storage

In order to perform the backup of the health data and satisfy the related requirement (see table 6 FR 4.4), the cloud storage service has been employed. The main reason lies behind the fact that health data can be pretty large, so using the Cloud Firestore Database to store them would be inefficient and expensive. For this reason the health data if the users are stored on the cloud storage into JSON files, one for each backup. The backup task is performed thanks to the flutter workmanager library by running a background task that allows to execute the backup even if the mobile app is not opened or actively used in that moment. However, since these tasks run on a separate background isolate, retrieving the health data directly there is not possible due to permission issues. For this reason, the backup implementation is structured into three phases:

1. In the first phase, that takes place at each application startup, the `performLocalBackup` function is called. This function retrieves the health data in the main isolate (that has the permission) through the health library. The fetch timespan goes from the `lastBackupDate` field (taken from the `user_data` collection for each user) to `Datetime.now()` and the data are stored into a JSON file in the local storage of the device, named with `Datetime.now().toIso8601String()` to guarantee univocity. After that the `lastBackupDate` field is coherently updated to `Datetime.now()`. This local backup is performed with a daily frequency, so if the app is opened more than once a day or simply no data are available the function returns immediately, avoiding to increase the frequency or to create an empty file.
2. In the second phase, that takes place immediately after the first one, the workmanager background task is scheduled through the `scheduleBackupTask` function. The task is a periodic task with the same frequency of the local backup (daily). Also in this case, if the app is opened more than once a day, even if the function is called multiple times, the task id guarantees that the periodic task is still scheduled once.
3. In the third phase, that takes place when the workmanager background task triggers and is executed, Firebase is initialized in the background isolate and the `performStorageBackup` function is called. This function loops through the JSON files saved in the local storage of the device and it uploads them to the storage. In case of successful upload, these files are marked for deletion and a second loop proceeds to delete them from the local storage of the phone. In case of failure, the files are not deleted and they will be uploaded when the task will be triggered again. These files are saved on the cloud storage in a folder named with the userID to guarantee univocity and to avoid conflicts between different users' backups, while the filename is also unique, since it is the same that was defined in the `performLocalBackup` function.

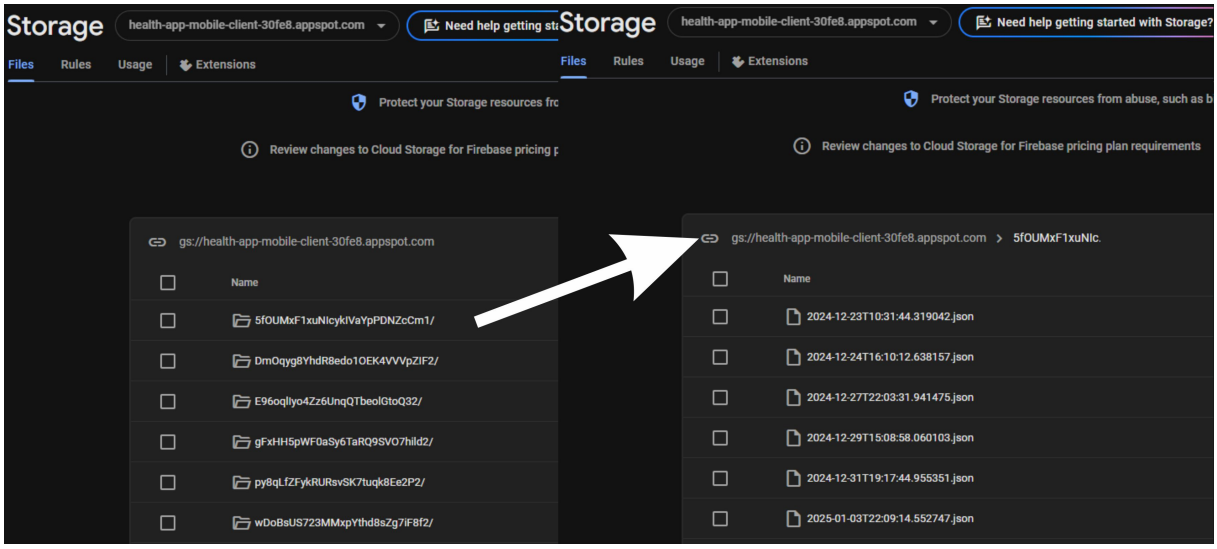


Figure 28: Cloud Storage view of the users folder, uniquely named with userIds(left) and view of a single user folder containing the JSON files of the backup, where each file is uniquely named with the timestamp (right).


```
{
    "dateFrom": "2024-12-23 10:00:12.473",
    "type": "HealthDataType.STEPS",
    "value": "NumericHealthValue - numericValue: 7"
},
{
    "dateFrom": "2024-12-23 10:01:12.473",
    "type": "HealthDataType.STEPS",
    "value": "NumericHealthValue - numericValue: 5"
},
{
    "dateFrom": "2024-11-25 16:45:28.000",
    "type": "HealthDataType.WORKOUT",
    "value": "WorkoutHealthValue - workoutActivityType: STRENGTH_TRAINING,\n          totalEnergyBurned: 543,\n          totalEnergyBurnedUnit: KILOCALORIE,\n    totalDistance: 69,\n          totalDistanceUnit: METER\n          totalSteps: 259,\n          totalStepsUnit: COUNT"
},
{
    "dateFrom": "2024-12-02 16:47:30.000",
    "type": "HealthDataType.WORKOUT",
    "value": "WorkoutHealthValue - workoutActivityType: STRENGTH_TRAINING,\n          totalEnergyBurned: 746,\n          totalEnergyBurnedUnit: KILOCALORIE,\n    totalDistance: 168,\n          totalDistanceUnit: METER\n          totalSteps: 576,\n          totalStepsUnit: COUNT"
},
{
    "dateFrom": "2024-12-04 16:14:36.000",
    "type": "HealthDataType.WORKOUT",
    "value": "WorkoutHealthValue - workoutActivityType: STRENGTH_TRAINING,\n          totalEnergyBurned: 684,\n          totalEnergyBurnedUnit: KILOCALORIE,\n    totalDistance: 182,\n          totalDistanceUnit: METER\n          totalSteps: 354,\n          totalStepsUnit: COUNT"
},
{
    "dateFrom": "2024-12-06 16:18:30.000",
    "type": "HealthDataType.WORKOUT",
    "value": "WorkoutHealthValue - workoutActivityType: STRENGTH_TRAINING,\n          totalEnergyBurned: 1055,\n          totalEnergyBurnedUnit: KILOCALORIE,\n    totalDistance: 235,\n          totalDistanceUnit: METER\n          totalSteps: 426,\n          totalStepsUnit: COUNT"
},
```


Figure 29: Sample content of a json file of the backup, containing the health data of the user.

4.10 Notifications

In order to satisfy the notifications requirements (see table 5), a notification system was designed and implemented. The notification system was designed with a cyclical behaviour. The number of days that a full cycle of notification takes to go across all phases depends on the `assessment_period` parameter inside the `notifications_text` collection, editable by the admin. Infact, the scheduling of each notification during the different phases is not immediate but depends on the `assessment_period` field and it is calculated based on that (`assessment_period`/4 for each phase).

The main cycle is composed of four phases, with the `current_notification` field initialized to the first phase, and employed to identify the current one. The phases are the following:

1. In the first phase, the first notification is sent and prompts the user to complete a test in order to assess his balance capabilities. In this case, the `current_notification` field has as value body_test_balance.
2. Once the user inserted his body balance data, the `current_notification` field is updated to body_test_strength and the second phase begins. Another notification will be sent, in this case to assess the physical strength of the user.
3. Once the user inserted his body strength data, the `current_notification` field is updated to emotional_life_test and the third phase begins. Another notification will be sent, in this case to assess the mood and emotional status of the user.
4. Once the user inserted his emotional data, the `current_notification` field is updated to assessment and the last phase begins. Another notification will be sent, in this case opening an assessment based on the data provided by the user instead of prompting him to insert data, so that the user can see a period assessment each `assessment_period` days. At this point the cycle restarts from the first phase.

For all those phases there is of course the possibility to ignore the notification by discarding it. In this case to further prompt the user into an active participation, the same notification is rescheduled with an higher frequency. Howewer, this is performed for a limited amount of times, and that is where the `notification_counter` field comes into: each time an user discard a notification the counter is incremeted for a maximum of three times (from 0 to 2 included). When the counter reaches the maximum value, the notification is not rescheduled anymore and that specific phase is skipped, moving to the next one and resetting the counter to 0.

A separate lifecycle is used to manage the food interaction instead: in this case the frequency is daily, and in case the user discards the notification, it is simply rescheduled again.

The `NotificationService` class was employed to handle this logic, with two main methods to handle the two separate lifecycles: `scheduleNotification` for the main cycle and `scheduleFoodNotification` for the food cycle. These methods are called ad each application startup to schedule the notification, handling the case where the notifications have already been scheduled. Before these two methods the `initializeNotification` method is called to initialize the notifications and setup the listeners to handle notification creation, opening and discarding. To practically schedule the notifications and use the listeners the `awesome_notifications` API have been employed.

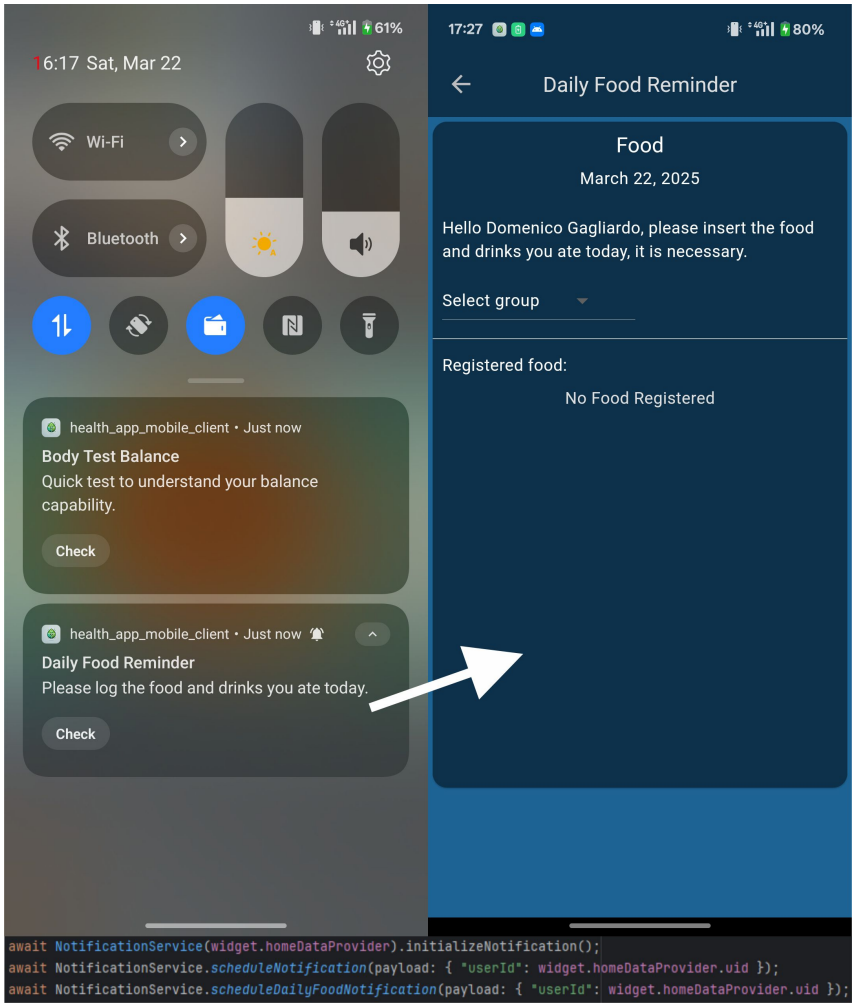


Figure 30: Example of the notifications sent (left), of the food form opened from the notification (right) and function call at application startup (bottom).

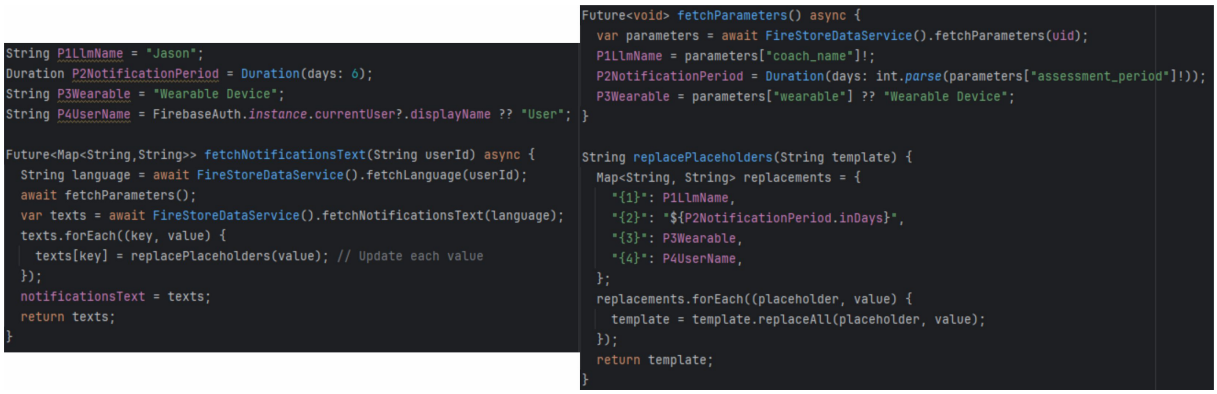
4.11 Application Parameters

The application parameters were employed to customize the app behaviour and the user experience. These parameters were used to perform the onBoarding, but also for the input forms (food form, body balance form, emotional forn and so on), present both in the graphs (giving the possibility to the user to enter data) and in the notifications, since the same widgets were used in both cases.

Coming to their usage, it is possible to make a practical distinction by using the `notifications_text` collection documents to understand how they were used (see section 4.1.2):

1. Firstly the `parameters` document and the user metrics were considered: the `assessment_period` and `coach_name` parameters were firstly fetched, together with the user wearable device (`wearable` field, available for each user inside the `users` collection `metrics` field, among user personal information) and the username, directly fetched from the authentication session inside the application.
2. Then, considering the other two documents of the `notifications_text`, one of the two was fetched, depending on the language. At this point, since each string field contains placeholders for the parameter mentioned at step 1, a pre-processing step was performed to replace those placeholders with the actual values, in order to obtain fields customized with the admin parameters (coach_name and assessment_period) but also user-tailored with the specific username and wearable device.

The whole parameter logic is based upon the `fetchNotificationsText` function. Here the user language is fetched. Then the four parameters, initialized with default values, are fetched and overriden when the `fetchParameters` function is called. After that, the `fetchNotificationsText` retrieves the `notifications_text` parameters based on the language and the `replacePlaceholders` function is executed for each string field to replace the fields correctly. All those function are defined and present in the `HomeDataProvider` class in order to be easily accessible in the whole application, wherever these parameters are needed.



```
String P1LlmName = "Jason";
Duration P2NotificationPeriod = Duration(days: 6);
String P3Wearable = "Wearable Device";
String P4UserName = FirebaseAuth.instance.currentUser?.displayName ?? "User";

Future<Map<String,String>> fetchNotificationsText(String userId) async {
  String language = await FireStoreDataService().fetchLanguage(userId);
  await fetchParameters();
  var texts = await FireStoreDataService().fetchNotificationsText(language);
  texts.forEach((key, value) {
    texts[key] = replacePlaceholders(value); // Update each value
  });
  notificationsText = texts;
  return texts;
}
```

```
Future<void> fetchParameters() async {
  var parameters = await FireStoreDataService().fetchParameters(uid);
  P1LlmName = parameters["coach_name"]!;
  P2NotificationPeriod = Duration(days: int.parse(parameters["assessment_period"]!));
  P3Wearable = parameters["wearable"] ?? "Wearable Device";
}

String replacePlaceholders(String template) {
  Map<String, String> replacements = {
    "{1}": P1LlmName,
    "{2}": "${P2NotificationPeriod.inDays}",
    "{3}": P3Wearable,
    "{4}": P4UserName,
  };
  replacements.forEach((placeholder, value) {
    template = template.replaceAll(placeholder, value);
  });
  return template;
}
```

Figure 31: Parameter fetch flow.

4.12 Web Application

To cover the admin requirements (see table 4) and allow the edit of the application parameters and the lessons/quizzes management, a react web application was employed. The application leverages on the firebase dependency [37] to enforce both authentication (see section 4.1.1) and parameters editing.

The application easily allows the admin to find which parameter to edit through a sidebar, that highlights the different parameters so that the admin can easily find them. The application also allows to change the language of the app through a dropdown, so that the admin can see the italian version of the parameters when it is needed, to edit them accordingly. In that case, speaking of parameters, only editing is possible and no deletion can be performed through the application.

Considering lessons and quizzes management instead, there is the possibility to add, edit and delete them. The addition and deletion operations have been done atomically in a transaction, in order to add lesson and quiz together, as well as deleting them together. This in order to link the lesson with the quiz through the `quizId` field and avoid inconsistencies, like a quiz with no lesson or viceversa, which would have required at least a dedicated screen to let the admin link them together. Additional sections have been added to the sidebar to allow to perform these operations. In particular there are the `Add Lesson/Quiz` section, to add a lesson with the corresponding quiz. Then the `Lessons` section to view, edit and delete the lessons along with the quizzes, and the `Quizzes` section to view and edit the quizzes (see fig. 19, the logged web client).

The edit operations were performed with the `updateDoc` method, that allows to update only the fields provided without affecting the rest of the document, while the add and delete operations were performed transactionally, respectively with `transaction.set` and `transaction.delete` methods.

```
const handleSubmit = async (e) : Promise<void> => {  Show usages
    e.preventDefault();
    if ( Object.values(mainParameters).some(value : string => value === '')) {
        setError( value: "You must provide all fields to save the changes.");
        return;
    } else {
        setError( value: '');
    }
    // NOTIFICATIONS TEXT
    const documentId : string = language === "English" ? "XpefscNNTLdybzJxOuwq" : "SCNwoZ6ACYiRSNYeXnZD";
    const docRef : DocumentReference<DocumentData, DocumentData> = doc(db, path: 'notifications_text', documentId);
    await updateDoc(docRef, data: {
        body_test_balance1: mainParameters.body_test_balance1,
        body_test_balance2: mainParameters.body_test_balance2,
        body_test_balance3: mainParameters.body_test_balance3,
        body_test_balance4: mainParameters.body_test_balance4,
        body_test_balance5: mainParameters.body_test_balance5,
        body_test_balance6: mainParameters.body_test_balance6,
        body_test_balance7: mainParameters.body_test_balance7
    });
};
```

Figure 32: Example usage of parameters update (body_test_balance).

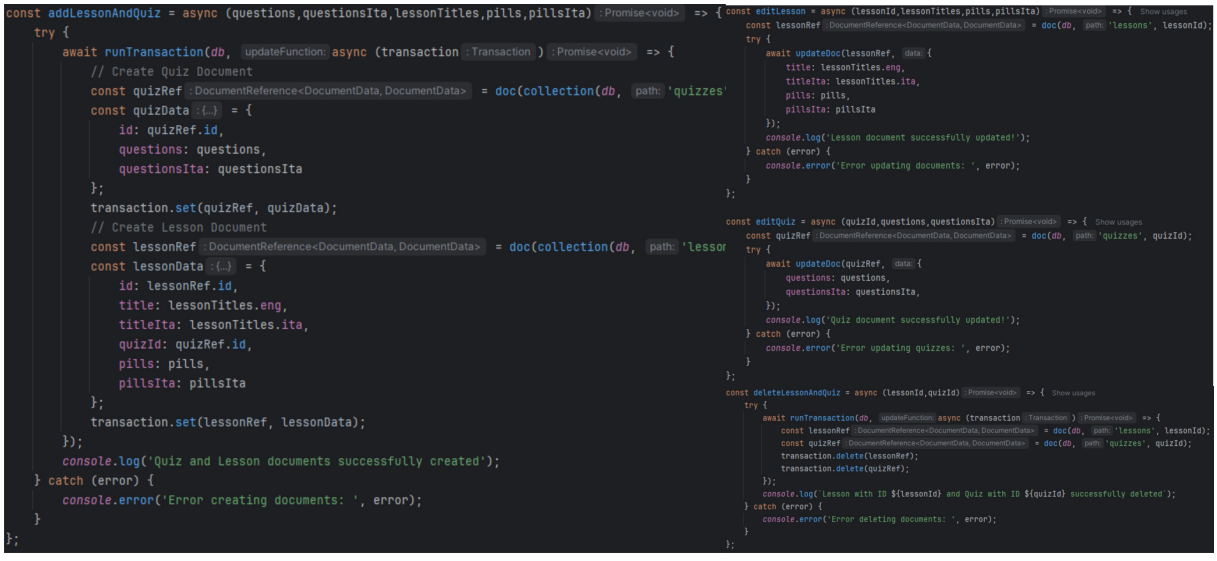


Figure 33: Lessons And Quizzes database operations.

5 System Outcomes and Enhancements

5.1 Achieved Performances

Once the system features were delivered, a particular focus was made on the performances in order to meet the NFR6 (see table 7). This was possible through the usage of the Performance Monitoring tool provided by the Firebase platform.

5.1.1 Startup Time

The performance tool allowed to monitor the application startup time as main metric of relevance. The library allowed an easy monitoring implementation by creating a `trace` instance, call his `start` method, perform the application startup and then call the `stop` method. The timespan considered was 60 days, the largest one allowed. The trend is positive, since the data fetch operations are called only for the shown data, and not for the whole dataset. However, it has to be said that peak values found were over 2 seconds: that was the case when the backup task severely impacted on the metric performances (`performLocalBackup` method, see section 4.9). In any case, the 90th percentile response time relative to the metric is 971 ms, so well under 2 seconds.

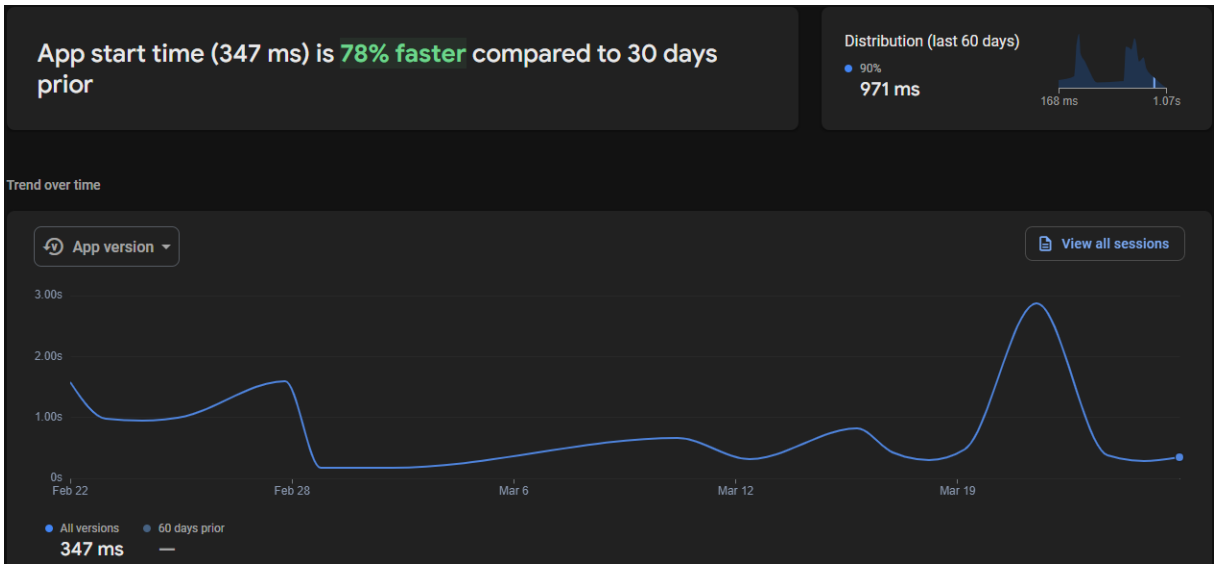


Figure 34: Performance analysis of the startup time metric, with very good results.

5.1.2 API Calls

Also the API calls were monitored, in order to understand the impact of the data fetching operations on the overall performances. Also in this case the timespan considered was 60 days, and each API call was monitored with a `trace` object, like the application startup. Without focusing on a specific API, on average the results were pretty good, since we had an average response time of 176 ms and an increasing performance trend of the 25.48%.

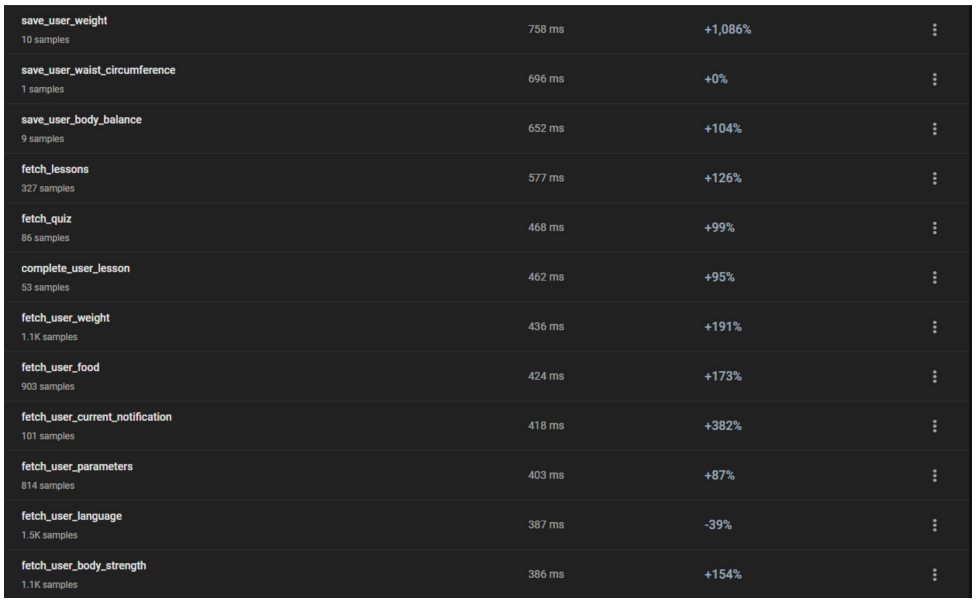


Figure 35: Performance of the API calls.

```
Future<void> deleteFood(String documentId) async {
  final Trace foodTrace = FirebasePerformance.instance.newTrace("delete_user_food");
  foodTrace.start();
  FirebaseCrashlytics.instance.log("Deleting user_food record...");
  final CollectionReference foodRecords = db.collection('foodRecords');
  try {
    await foodRecords.doc(documentId).delete();
    print("Food record deleted successfully.");
  } catch (e, stackTrace) {
    print("Error deleting food record: $e");
    FirebaseCrashlytics.instance.recordError(e, stackTrace, reason: "Failed to delete user_food data");
    foodTrace.stop();
    rethrow;
  }
  foodTrace.stop();
}
```

Figure 36: Monitoring example on the deleteFood API call.

5.2 Crashlytics

Also the `Crashlytics` tool has been helpful, thanks to the crash monitoring and the possibility to deeply analyze the log related to detected errors. Thanks to this tool provided by the Firebase platform, a bug in the Health Data Backup (see section 4.9) was detected and fixed consequently. This allowed an improvement of the overall application stability, passing from a 93.7% crash-free users to a 100% crash-free users. Even though is not a big improvement and the percentage was already high, it is still worth to mention.

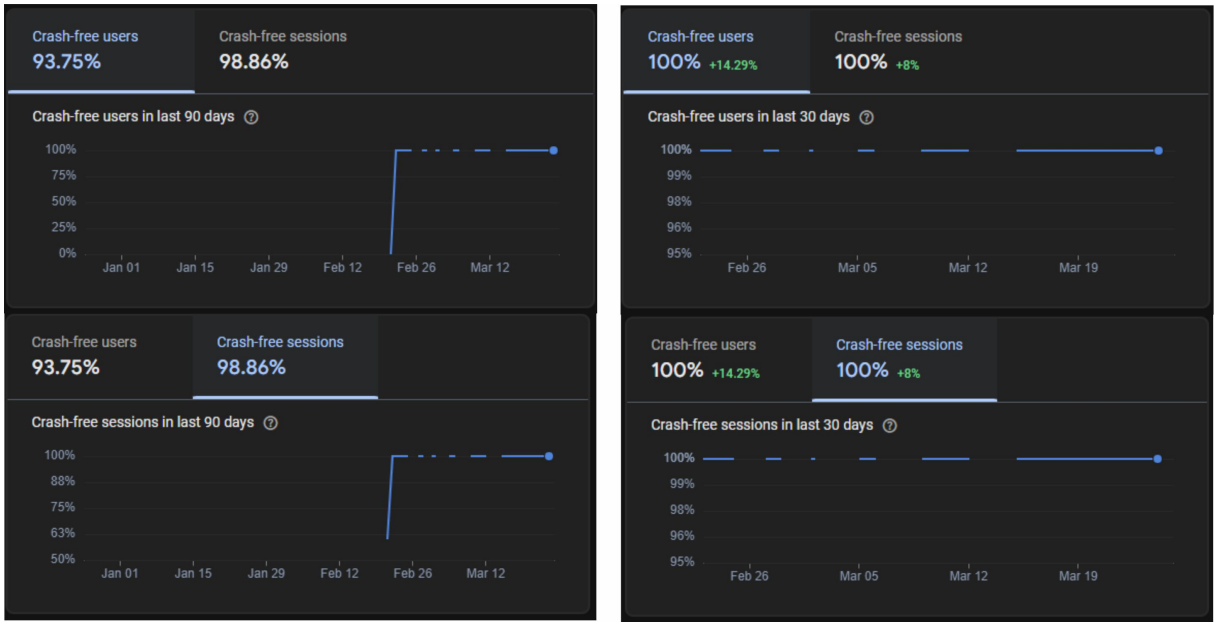


Figure 37: Crash-free users and session in the last 90 days, with the bug present (left) and in the last 30 days, after the bug fix (right).

5.3 Future Developments

Rarely a system can be considered perfect. There is always room for improvement in several aspects, so of course a future development can be done on this side.
Having said that, other than improvement on current features, other possible new features have been considered in this system, that can definitely make it different, allowing to stand out from the competition:

- **Ios Integration:** the system is already designed to be intrinsically cross-platform from a smartphone point of view (since flutter is cross-platform and supports IOS). For this reason, from a theoretically point of view, the IOS support should already be present in the system. However, the system has not been tested practically on IOS devices. This because a mac, an IOS device and an apple watch are needed to test the system in its entirety, and these devices were not in our possession. For this reason IOS was not covered in the elaborate and should be tested to assess the effective cross-platform support.
- **Personal Coach:** integrate another component in the system architecture acting as a personal coach by generating personalized health and fitness recommendations. The goal would be to provide data-driven, actionable, and behavior-changing insights tailored to the user. This would be possible thanks to the integration of a Large Language Model (LLM) that can generate personalized recommendations based on the user's data. In fact, the backup feature already present and integrated into the system has been developed precisely to allow this data to be used in the LLM model, in order to provide user-tailored insights and guaranteee the best user experience possible.
- **Personal Coach App Integration:** once developed and deployed the personal coach, the next step would be integrating it inside the application: infact, this should be the purpose of the recommend page (reachable through the central button in the bottom navigation bar), that has been added to the app, in addition to the home, health measures, personal information and learn pages. This page will be used to integrate the LLM model in the future, by inserting some recommendations generated by the model. It would also be useful to have the possibility to chat with the personal coach, in order to ask for clarifications or to have more information about the recommendations provided.

Bibliography

[1] World Health Organization (WHO), https://www.who.int/ [9].

[2] WHO Physical Activity Benefits, https://www.who.int/en/news-room/fact-sheets/detail/physical-activity [9].

[3] WHO Physical Activity Guidelines, https://www.who.int/publications/i/item/9789240015128 [10].

[4] WHO Healthy Diet Guidelines, https://www.who.int/initiatives/behealthy/healthy-diet [11].

[5] National Institutes of Health (NIH), https://www.nih.gov/ [12].

[6] NIH mobile technology studies, https://www.nih.gov/news-events/news-releases/nih-funded-study-examines-use-mobile-technology-improve-diet-activity-behavior [12].

[7] NIH Comparison of Self-Reported and Device-Based Measurements, https://pubmed.ncbi.nlm.nih.gov/33920145/ [12].

[8] NIH Smartphone Applications for Promoting Healthy Diet and Nutrition, https://pubmed.ncbi.nlm.nih.gov/26819969/ [13].

[9] NIH Smartphone Applications for Promoting Physical Activity, https://pubmed.ncbi.nlm.nih.gov/27034992/ [13].

[10] Global wearable band market in Q2 2024, https://www.canalys.com/newsroom/worldwide-wearable-band-market-Q2-2024 [15].

[11] Flutter Framework, https://flutter.dev/ [27].

[12] Flutter Framework, https://docs.flutter.dev/get-started/install/windows/mobile [27].

[13] React Framework, https://en.wikipedia.org/wiki/React_(software) [28].

[14] Dart Programming Language, https://dart.dev/overview [30].

[15] Javascript Programming Language, https://en.wikipedia.org/wiki/JavaScript [30].

[16] Groovy Programming Language, https://en.wikipedia.org/wiki/Apache_Groovy [31].

[17] Yaml Language, https://en.wikipedia.org/wiki/YAML [32].

[18] Pub Package Manager, https://dart.dev/tools/pub/cmd [33].

[19] Gradle Build Tool, https://en.wikipedia.org/wiki/Gradle [34].

[20] Gradle Basics, https://docs.gradle.org/current/userguide/gradle_basics.html [34].

[21] NPM, https://en.wikipedia.org/wiki/Npm [35].

[22] AndroidStudio IDE, https://en.wikipedia.org/wiki/Android_Studio [35].

[23] AndroidStudio Code Editor Feature, https://developer.android.com/studio/intro [36].

[24] AndroidStudio Build System Feature, https://developer.android.com/build [37].

[25] AndroidStudio Emulator Feature, https://developer.android.com/studio/run/emulator [38].

[26] AndroidStudio APK Analyzer Feature, https://developer.android.com/studio/debug/apk-analyzer [39].

[27] AndroidStudio New Features, https://developer.android.com/studio [39].

[28] VsCode Code Editor, https://code.visualstudio.com/ [40].

[29] Firebase Authentication, https://firebase.google.com/products/auth?_gl=1*1ifvla4*_up*MQ..&gclid=Cj0KCQjwpf2IBhDkARIsAGVo0D3BsyWTPDfP4LLkcA0rcwgJYQ97HunsvJtSWmV9fQUoolB5HNwdz8QaAjU8EALw_wcB&gclsrc=aw.ds [41].

[30] Cloud Firestore, https://firebase.google.com/products/firestore?_gl=1*c76szm*_up*MQ..&gclid=Cj0KCQjwpf2IBhDkARIsAGVo0D3BsyWTPDfP4LLkcA0rcwgJYQ97HunsvJtSWmV9fQUoolB5HNwdz8QaAjU8EALw_wcB&gclsrc=aw.ds [41].

[31] Cloud Storage, https://firebase.google.com/products/storage?_gl=1*1ifvla4*_up*MQ..&gclid=Cj0KCQjwpf2IBhDkARIsAGVo0D3BsyWTPDfP4LLkcA0rcwgJYQ97HunsvJtSWmV9fQUoolB5HNwdz8QaAjU8EALw_wcB&gclsrc=aw.ds [42].

[32] Performance Monitoring, https://firebase.google.com/products/performance?_gl=1*1hhc75t*_up*MQ..&gclid=Cj0KCQjwpf2IBhDkARIsAGVo0D3BsyWTPDfP4LLkcA0rcwgJYQ97HunsvJtSWmV9fQUoolB5HNwdz8QaAjU8EALw_wcB&gclsrc=aw.ds [42].

[33] Crashlytics, https://firebase.google.com/products/crashlytics?_gl=1*1hhc75t*_up*MQ..&gclid=Cj0KCQjwpf2IBhDkARIsAGVo0D3BsyWTPDfP4LLkcA0rcwgJYQ97HunsvJtSWmV9fQUoolB5HNwdz8QaAjU8EALw_wcB&gclsrc=aw.ds [42].

[34] Health Library, https://pub.dev/packages/health [43].

[35] Workmanager Library, https://medium.com/@nandhuraj/background-task-automation-in-flutter-unleashing-the-power-of-workmanager-20c5e610cfbf [43].

[36] Awesome Notifications Library, https://pub.dev/packages/awesome_notifications [44].

[37] React Firebase Dependency, https://www.npmjs.com/package/firebase [45, 67].

List of Figures

Ringraziamenti

Pensare al percorso che ho fatto in questi anni, così come tutte le persone che hanno contribuito a renderlo tale, mi rende felice e un pizzico nostalgico. Se sono arrivato qui, oltre che a me stesso, lo devo a tutte queste persone.

Ringrazio il mio relatore, Prof. Maurizio Morisio, per la sua disponibilità e il suo sostegno durante il tirocinio e la tesi.

Ringrazio i miei genitori, Gaetano e Maria, per tutti i sacrifici che hanno fatto, e che mi hanno permesso di arrivare dove sono. Senza il loro aiuto non sarei certamente qui ora.

Ringrazio i miei nonni Vincenza, Domenico, Sofia e Santino, per il loro supporto e amore incondizionato.

Ringrazio i miei zii e cugini per il loro supporto e affetto dimostratomi.

Anche se non coinvolti con l'università, vorrei ringraziare i miei amici più stretti per la loro preziosa compagnia: Vincenzo Petrillo, amico d'infanzia di una vita, ma anche Alberto Contaldi per la sua follia e Giuseppe Romano.

Ringrazio i miei colleghi e amici:

- Gaetano, noto anche come Tanucc/TanoDev, per tutte le risate e i momenti di studio e divertimento condivisi.
- Alessandro, noto anche come la Regina Regale, per la sua presenza regale durante i momenti di studio e divertimento, e tutte le risate condivise.
- Davide, noto anche come DaveBreaks, per i tanti momenti di divertimento condivisi insieme (non di studio dal momento che mi ha torturato con la grammatica e il padding [`187.92 px` numero perfetto], ma lo voglio bene lo stesso).
- Vorrei anche ringraziare Giorgio e Giuseppe (noto anche come LoHacker) per la loro compagnia e le tante risate condivise.

Insieme abbiamo condiviso la maggior parte dei giorni di studio, e li ringrazio per averli resi meno faticosi e certamente più divertenti, tra una risata e l'altra. Abbiamo condiviso tutta la difficoltà di questo corso, e siamo riusciti a superarla insieme.

Ringrazio tutti voi con i quali ho condiviso una parte di me e del mio percorso.