AY 2022 / 2023

POLITECNICO
MILANO 1863

SkiSage

Design and Implementation of Mobile Applications

Design Document

Francesco Zanella Stefano Chiodini

Professor
Luciano Baresi

September 5, 2023

Introduction

Project overview

SkiSage is an app that lets you train and discover Italy's skiing resorts. Select and review top resorts, gain insights from fellow skiers' rankings, and save your favorite spots to share across platforms.
Enhance your skiing experience by initiating real-time training sessions to record your performance, delve deep into post-session statistics, and track your progress over time with a complete training history.

This document outlines the key design decisions we made to realize our app.

Implementation Choices

While implementing the app functionalities, the choice was to focus more on the usability and on making the app usage as much intuitive as possible, more than implementing a lot of features. Thus the key feature of the whole application is to share knowledge gathered from students' experience in such a way that it won't get lost into the chaos generated by a complex app. A lot of importance has been given to making a responsive and glad to see interface. To perform this we have chosen Flutter as developing framework to deal with few but effective widgets that optimize the User Interface. We also need fast access to data that should be stored as they are, without being normalized; to do this we have chosen Firebase as database and backend manager.

Requirements

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

Requirement ID	Requirement Description
R1	Users should be able to sign in/sign up in the app
R2	Users should be able to logout from the app
R3	The system shall authenticate users before allowing access to specific features.
R4	The system shall display all Italian ski resorts on a map.
R5	The icon's size of a resort on the map shall be proportional to the average rating of that specific resort.

R6	Users shall have the ability to review each resort directly from the map interface.
R7	Users should be able to save any resort as a favorite from the map interface.
R8	The system shall maintain a dynamic ranking of the top 10 Italian resorts based on user reviews.
R9	Users shall be able to save their favorite resorts within the app.
R10	The system shall allow users to convert resort information into a shareable picture format.
R11	Users shall be able to share these pictures across multiple platforms such as Telegram, WhatsApp, Instagram, Email, Bluetooth, etc.
R12	Users shall be able to start a real-time training session while skiing.
R13	The system shall track and record multiple metrics during a training session
R14	Upon completion of a training session, the system shall provide detailed statistics of the session.
R15	The system shall maintain a history of all the user's training sessions
R16	Users shall be able to modify personal details such as first name, last name, and username.
R17	The system shall offer a theme choice to users: dark or light mode.
R18	The system shall provide an option for connecting to a smartwatch.

Features implemented

These are the implemented functionalities that meet the requirements:

Login and Registration

We have developed a custom classic sign in/sign up through email address and we have added also the possibility to sign in/sign up using the Single-Sign-On (SSO) functionality with Google identity providers

Map of Resorts

Users can see all the Italian resorts on the map; each resort is indicated on the map with a specific icon and the size of the icon is directly proportional to the average of the reviews of that resort. The user can either review each resort or save it as a favorite. So if the reviews

average changes a lot this can influence a lot the visibility of a resort on the map; indeed the resort icon will be bigger or smaller.

User-Ranked Best Resorts

The app offers insights into the most highly-rated resorts based on feedback from other users, giving a crowd-sourced ranking. In particular in the “most popular resort screen” the 10 Italian resorts with the best reviews are shown. Obviously this is a dynamic ranking and based on the reviews of other users the resorts will change.

Save and Share Resorts

Users can save their favorite resorts and effortlessly share them across multiple platforms, enhancing the social experience of the app. The container that contains every information about a resort is transformed into a picture and that picture is shareable everywhere: telegram, whatsapp, instagram, email, bluetooth ecc. In this way you can tell your friends where you are or where you want to go to ski.

Training Session Recording

While skiing, users can initiate a training session within the app to track and record their skiing performance in real-time. A variety of data is measured including user location, average speed, max speed, training time, distance travelled etc.

Comprehensive Training Statistics

Post-training, users can access detailed statistics from their session, providing insights into their performance and areas of improvement. For example, a graph is shown that clearly represents the speed trend over time.

Historical Training Overview and Best Results Highlight

Users can view their entire training history, allowing them to track progress over time; each workout is saved by recording the date, time and location; the list of all workouts is visible on a dedicated screen. The app also highlights the best performance of a user, indeed, the best performance is highlighted in the home page.

Manage profile settings table

A user can manage its profile settings:

- by changing first name, last name or username;
- by updating the profile picture;
- by changing the email address associated to the account;

- by changing the phone number associated to the account;

Manage app settings

- theme (dark or light);
- bug report through email channel;
- app ratings;
- smartwatch connection;

Application architecture

Overview

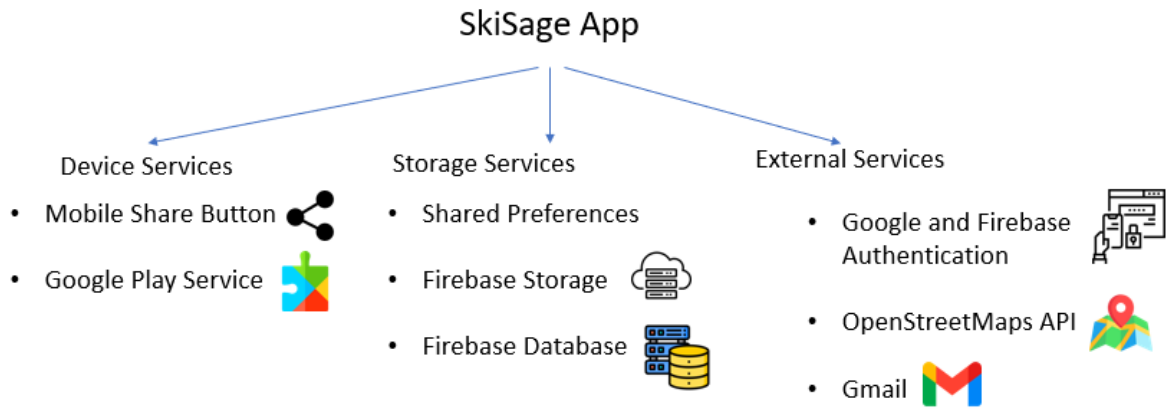
We have decided to realize our app using Flutter; Flutter is an open source framework developed and supported by Google.With Flutter it is possible to build, test, and deploy beautiful mobile, web, desktop, and embedded apps from a single codebase.
Flutter is powered by Dart, a language optimized for fast apps on any platform.

In creating the application, we utilize various services that can be categorized into three groups:

- Device Services: These APIs correspond to the device's sensors and innate features.
- Storage Services: APIs focused on managing and saving the application's data.
- External Services: APIs that integrate functionalities and data sourced from third-party companies.

By harnessing asynchronous communication methods where appropriate we've ensured the app remains fluid and responsive, independent of any variations in the performance of external services.

In summary:



Now let's see in detail how we used these three categories.

Device Services

Google Play Services for Location Tracking

To enhance the app's training features, we leverage Google Play Services, specifically its location capabilities, to track a user's real-time position while skiing. By doing so, the app can consistently monitor and calculate the user's speed throughout their training session. This live tracking offers users a dynamic and interactive experience, enabling them to gauge their performance accurately as they ski.

Mobile Share Button Integration

Aiming for a seamless and integrated user experience, our app incorporates the standard mobile phone share API. When users wish to share information about a particular ski resort, they can simply click the share button. This action generates an image encapsulating the most pivotal details about the resort. Users can then effortlessly distribute this image, accompanied by a caption, across various platforms including Telegram, Instagram, WhatsApp, Gmail, Bluetooth, and more.

Storage Services

In our app to save information related to a user we have used both local and cloud storage services. We will explain the cloud one first

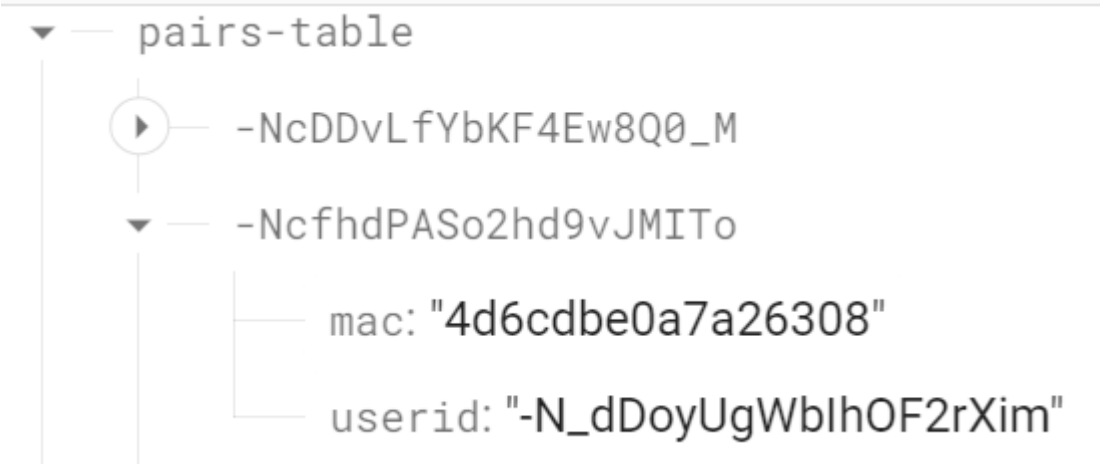
Cloud Firestore

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

favorites-resort-table

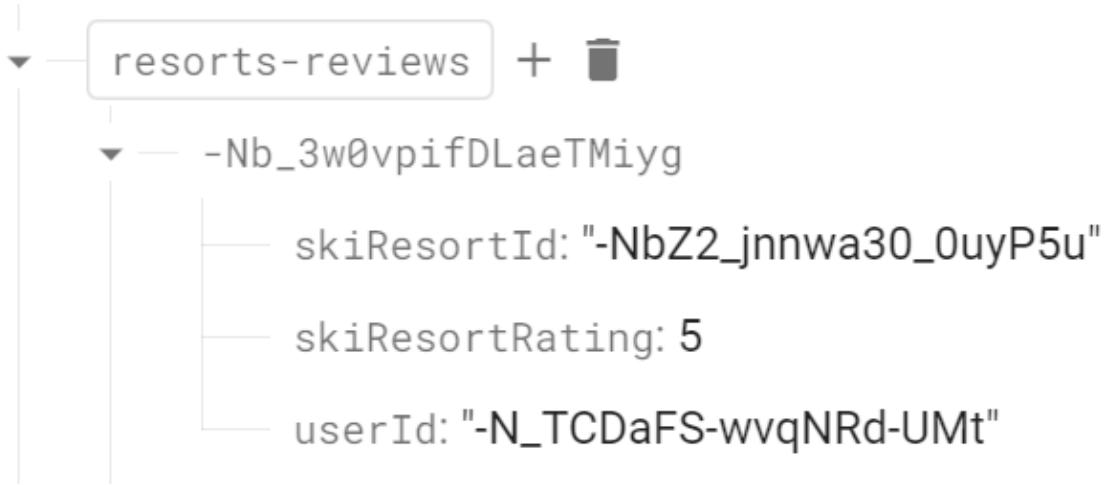
This table retains the resort information associated with a specific user ID. By doing so, it enables the swift retrieval of all favorite resorts for a user upon login. Each entry essentially creates a link between a user and their preferred resort, ensuring that personalization is maintained.

pairs-table



Dedicated to storing established connections between individual mobile phones and smartwatches, this table ensures that devices are paired correctly and efficiently. The pairing phase is made by a qr code scanner

resorts-reviews



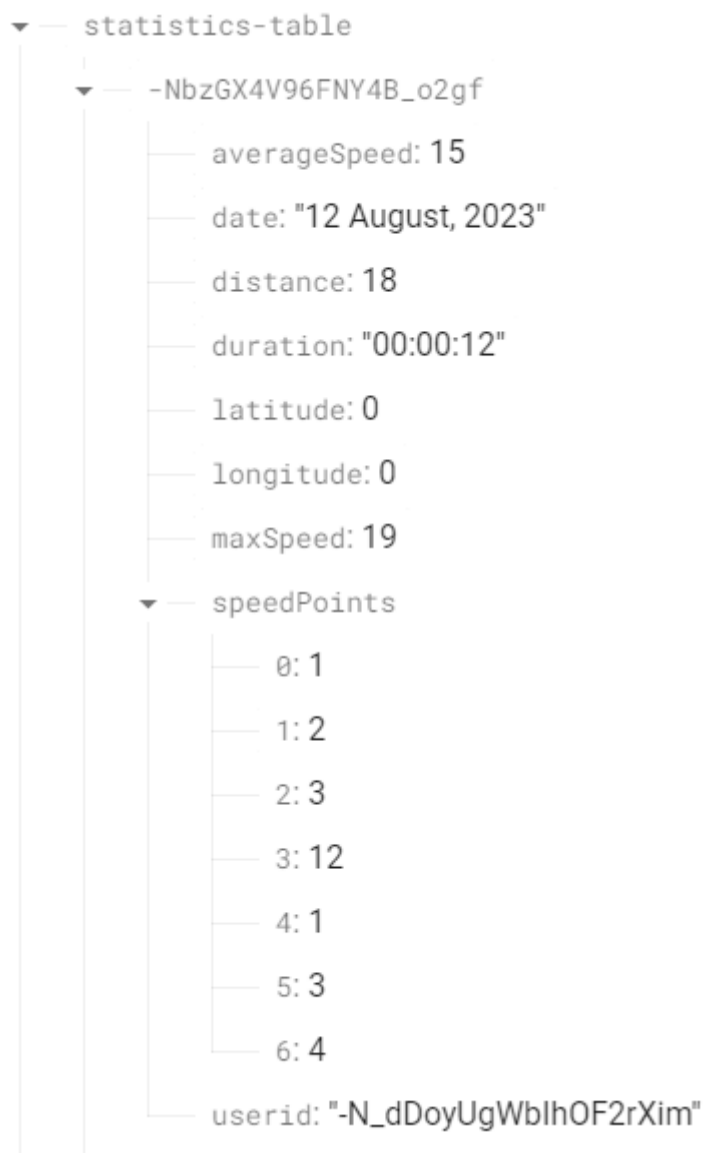
This table is designed to maintain user reviews specific to each resort. Whenever a user submits feedback or rates a resort, the data is logged here, enabling an ongoing assessment of resorts based on user experiences and feedback.

resorts-table

In the process of our data acquisition, we developed a web crawler using Python. This crawler was designed to extract the most salient information from this website. Suddenly for every extracted resort we took every respective coordinate(latitude and longitude) from this website.
Once we obtained the data, it was restructured to align with our specific requirements. Subsequently, the refined data was pushed into our database, populating the resorts-table.

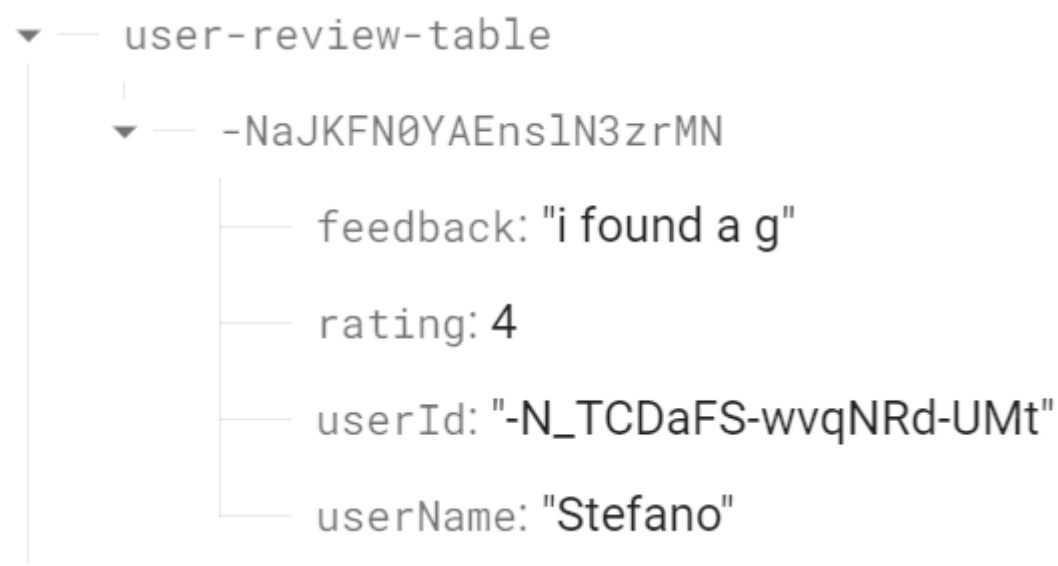
This table houses all essential information pertaining to each ski resort. Details such as the number and type of ski slopes, the cost of the ski pass, geographical coordinates, and other relevant resort details are stored here. It serves as a central repository for all resort-specific data.

statistics-table



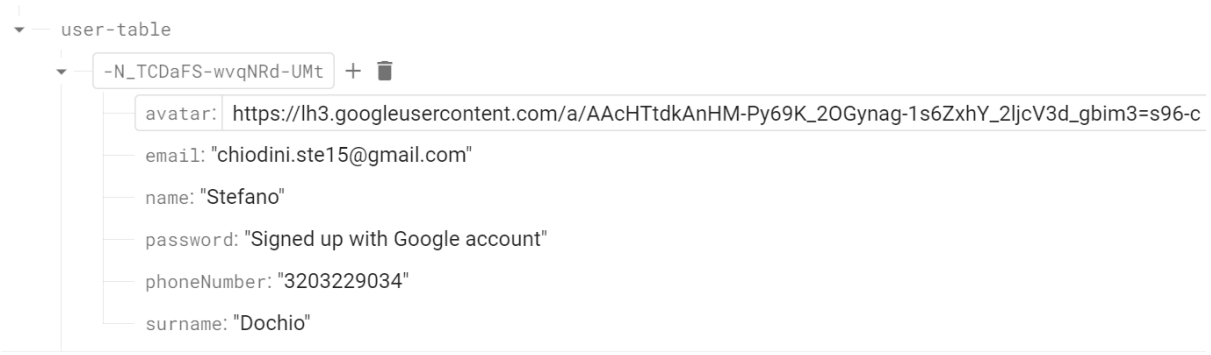
As the name implies, this table is instrumental in recording a wide range of data related to training statistics. This data aids in generating comprehensive analysis, graphs, and insights into the skiing performance of users over time, allowing them to gauge and enhance their skiing proficiency.

user-review-table



Unlike the resorts-reviews table which focuses on specific resorts, this table archives all the reviews made by users about the app itself. This feedback mechanism helps in continuous app improvement by understanding user sentiments, likes, and areas of concern regarding the application.

user-table



This collection stores information about registered users, i.e. email, username and profile picture (file is stored in firebase storage, while its url is stored here). Every entry of this table can be editable from the settings page.

Cloud Storage

Cloud Storage is built for apps that need to store and serve user-generated content, such as photos or videos. In this scenario, it has been used mainly to store users' profile pictures. Each user has only one profile picture associated with him/her.

Shared Preferences

This is the only type of local storage used; obviously we preferred the use of cloud storage to store any type of data. We have chosen to save only a very small amount of data locally; which are data associated with the user only to allow better performance and usability

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

- choice of theme mode between light and dark (key: darkMode).
- if a user is logged in when closes the app; in this way if he reopen the app he is still logged in and don't have to do login again
- some user's data to improve performances

In our app the shared preferences are kept updated automatically and transparently to the user; the user only has control over the dark mode shared preferences which he can change from the settings at any time: when a user updates one of these options in the dedicated page, a boolean value is assigned to the specific key and stored in memory so that, when the user closes and reopens the application, the chosen setting are always restored.

External Services

Firebase Authentication + Google Sign-In

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

OpenStreetMaps API

OpenStreetMaps is a project that creates and distributes free geographic data for the world. Flutter provides a specific package for building maps (with templates, markers, polylines, etc..) which can be easily integrated with the OSM API. In this scenario, the map shows the position of every Italian ski resort with a little information for every resort. We also use openstreetmaps to show the user's location during training.

Gmail Bug Reporting

To streamline the process of addressing software glitches and improve user experience, our app integrates a bug reporting feature utilizing Gmail. When users encounter issues, they can easily detail the problem within the app. Upon clicking the "send bug report" button, the app pre-populates an email body with the user's description. This email is then dispatched through the user's Gmail account, ensuring swift communication between the user and our technical team. This integration simplifies the bug reporting process.

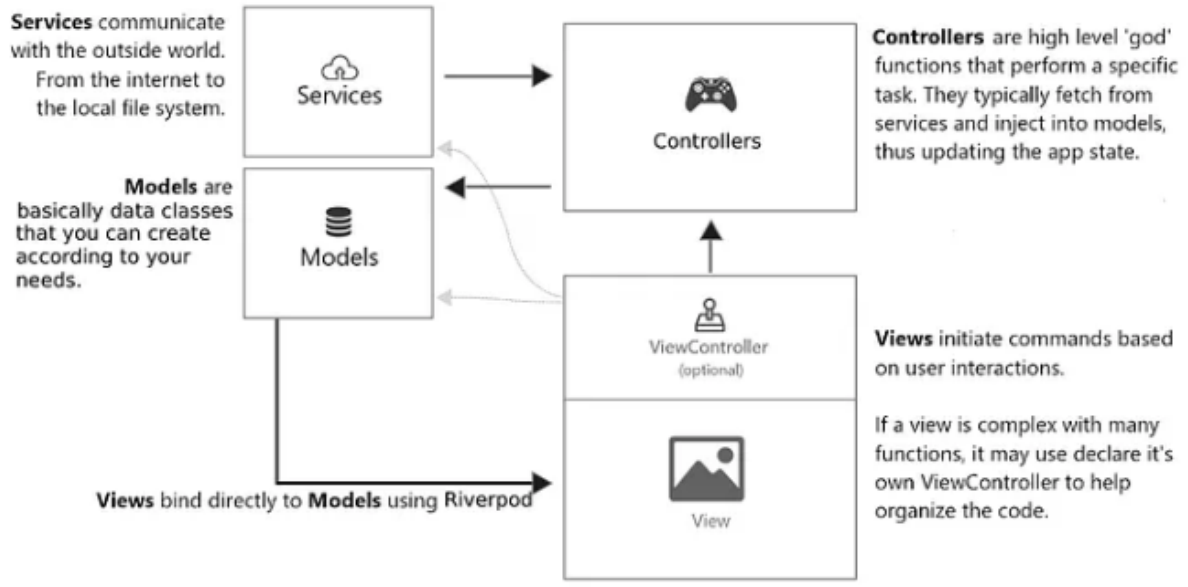
Dependencies

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

firebase_core firebase_auth	Used to handle user data, the registration and the authentication process.
google_sign_in	Used for authentication with Google.
cloud_firestore	Necessary to use the Cloud Firestore API.
firebase_storage	Necessary to use the Firebase Cloud Storage API.
shared_preferences	Used to provide persistent storage for simple data.
flutter_test integration_test mocktail	Used to perform unit tests, widget tests and integration tests.
geolocator geocoding	Used to access location services Used to compute the address from coordinates.
wear	A plugin that offers Flutter support for Wear OS by Google (Android Wear).
image_picker file_picker	Used to load images when updating profile pictures.
share_plus widgets_to_image	Used to make a widget shareable through images.
fl_chart charts_flutter simple_animations	Used to draw charts on training statistics.
qr_code_scanner unique_identifier qr_flutter	Used to link for the first time a mobile device to a smartwatch.

Model View Controller pattern(MVC)

In the development of our Flutter app, we adopted the Model-View-Controller (MVC) architectural pattern, a time-tested design that facilitates modularity, scalability, and maintainability. Here's a breakdown of how MVC is implemented:



- **Model**:

  Description: Represents the data and the business logic of the application. It directly manages the data and the rules for updating or processing it.
  In Our App: Models in our app are responsible for example for fetching resort details, managing user favorites, or handling user authentication data. They interact with databases like Firebase and external APIs such as OpenStreetMaps to fetch, store, and update data.

- **View**:

  Description: Represents the UI of the application. It displays the data to the user and sends user commands to the controller. Essentially, it's all about presentation and how things are displayed.
  In Our App: This is what the user interacts with directly. For instance, the favorite ski resort details page, the login page, and the settings page are all part of the View. It's built using Flutter widgets and doesn't contain any direct data manipulation logic.

- **Controller**:

  Description: Acts as an interface between the Model and the View. It receives user input from the View, processes it (with potential updates to the Model), and returns the display output to the View.
  In Our App: When a user decides to share a favorite ski resort, the controller decides what should happen. It communicates the user's decision to the relevant model, captures the relevant data, and then updates the view to show a "share successful" message.

User Interface(UI)

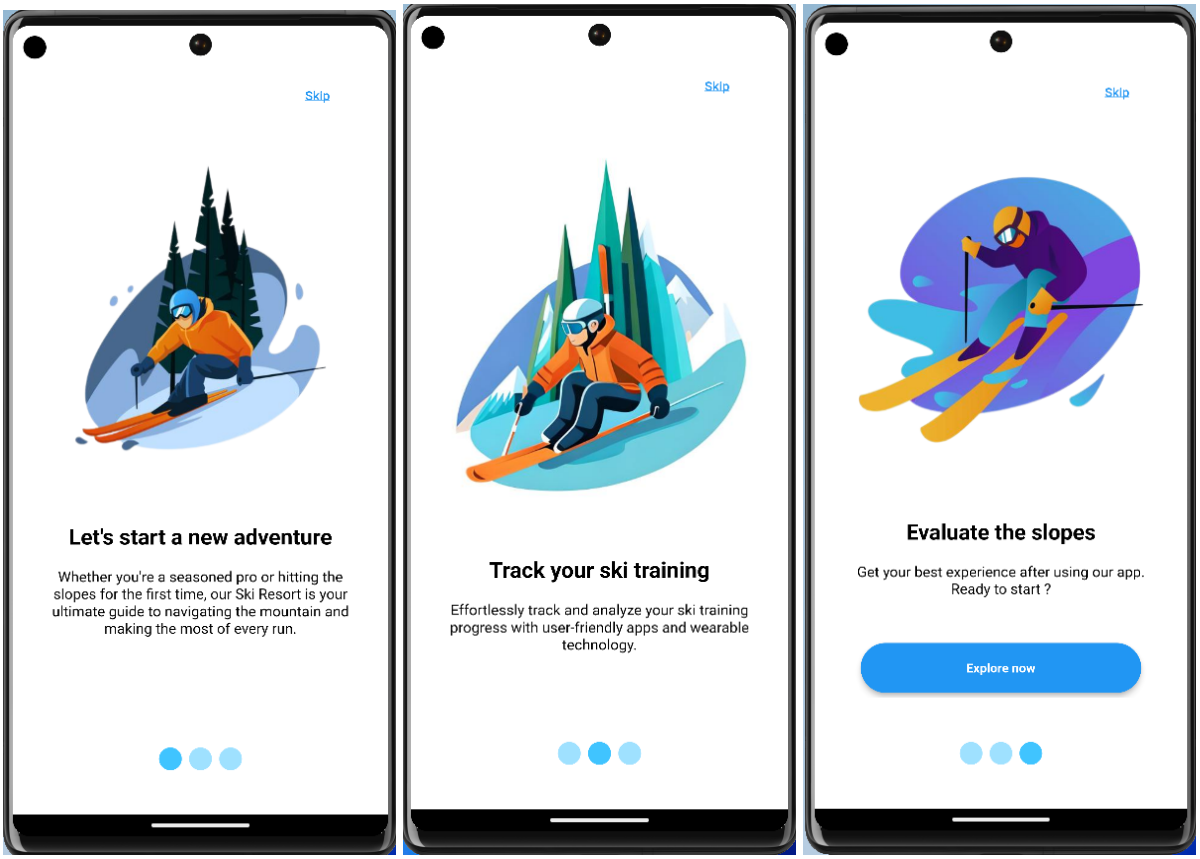
The use of Flutter creates a User Interface where everything works together to create beautiful and attractive layouts and renders. Our implementation is easy to understand but still attractive and not tedious for the user. This is because we want to guarantee a painless experience that doesn’t bother the customer during the usage of the app.

Most of the screens are shown either in light mode or in dark mode for the sake of brevity but of course every screen has been implemented in dark and light mode.

Smartphone UI

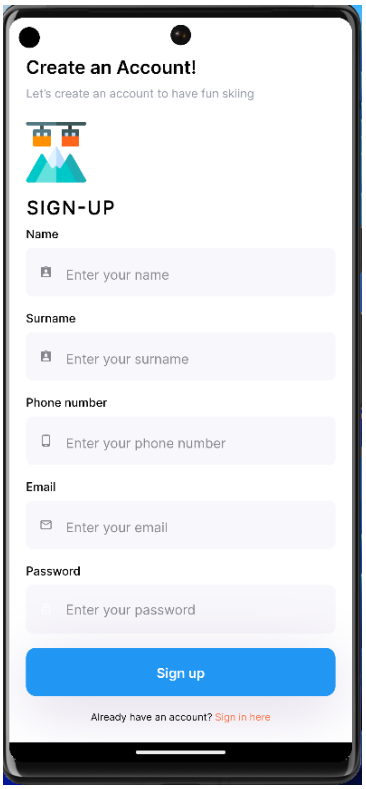
The App was initially designed for smartphones. Later, we decided to add support for an alternative layout: the smartwatch.

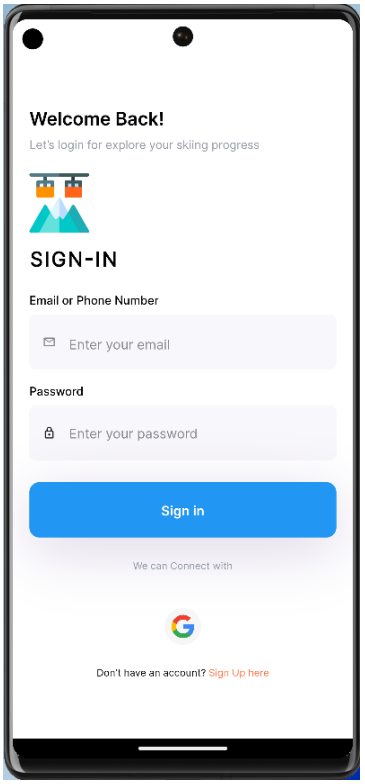
Pre Login Page



Displayed when the app is first opened or a user executes the logout procedure, it introduces the app to users and guides them to either log in or register, providing a glimpse of the app's features.

Login and Registration Page

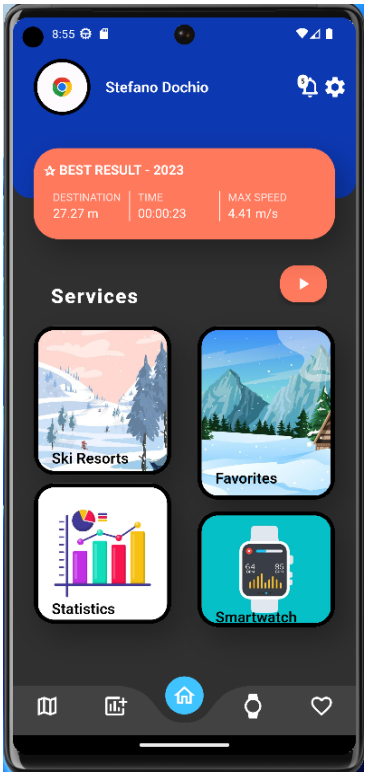


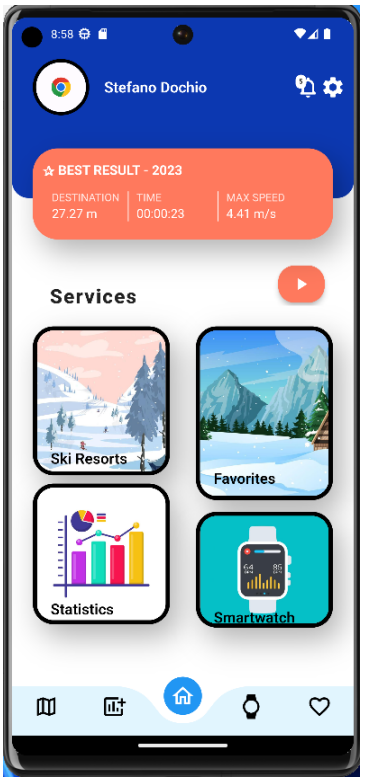


This screen is dedicated to user authentication. It allows new users to register and returning users to log in, offering both traditional email/password and Single-Sign-On methods.

A text button is used to switch from the login to the registration form and vice versa. Google Authentication is also available.

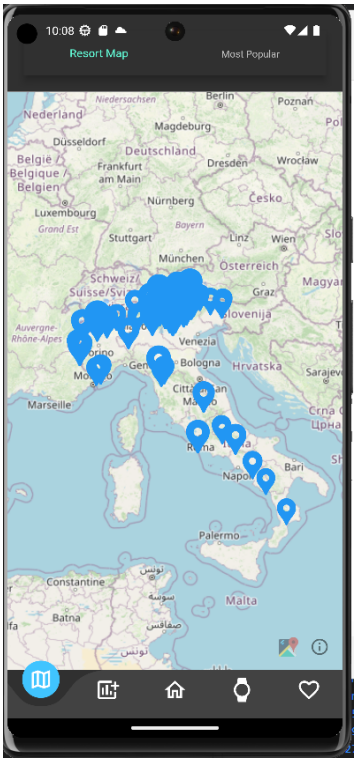
Home Page

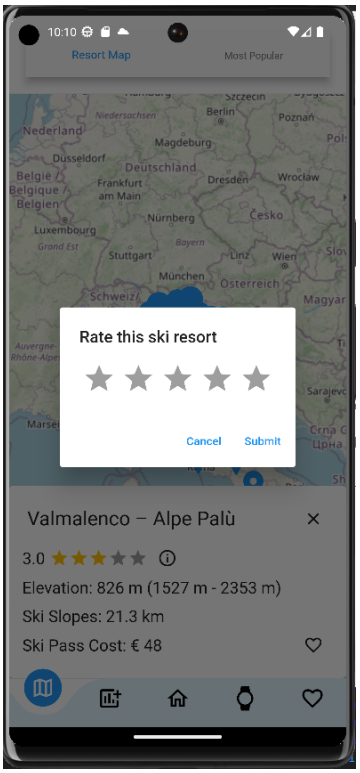


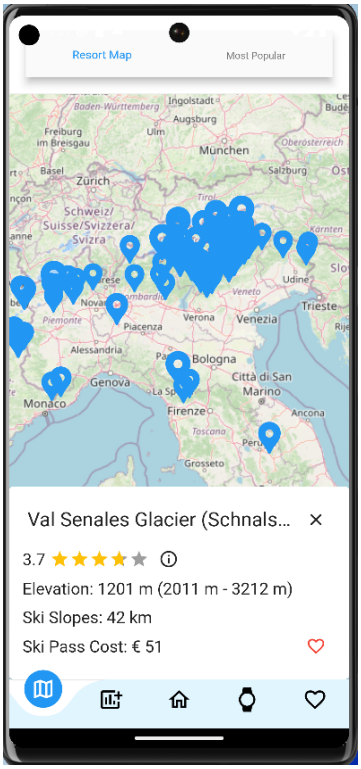


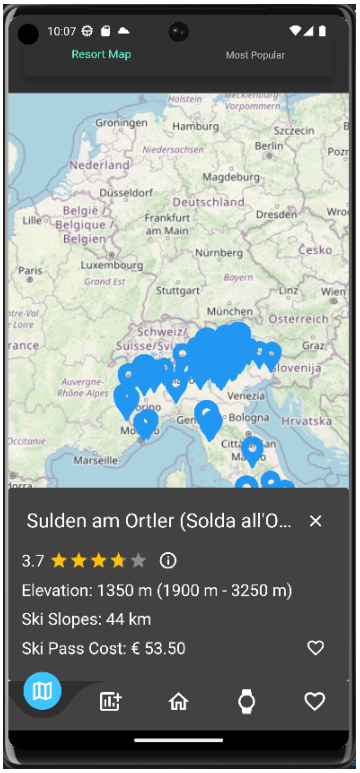
The main landing screen post-login, highlighting a summary of the app's features, user's statistics, and possibly their best performances. It acts as a hub to navigate to other parts of the app.
From the home page it is possible to reach any other screen of the app; moreover, using a special button it is possible to start a training session.
Finally the main information of a user is shown and there is a shortcut that lets the user go quickly to the settings page screen where he is able to modify his information.

Resort Map Page



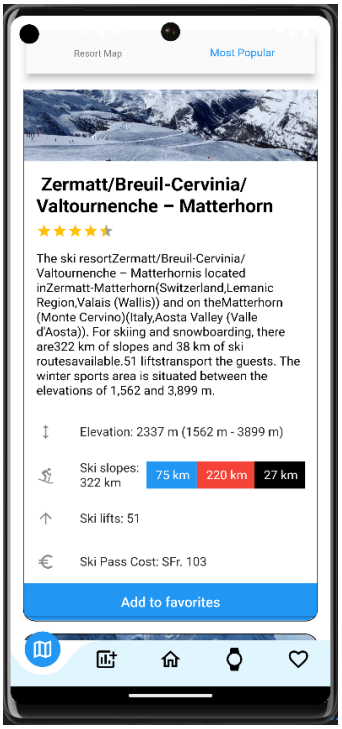


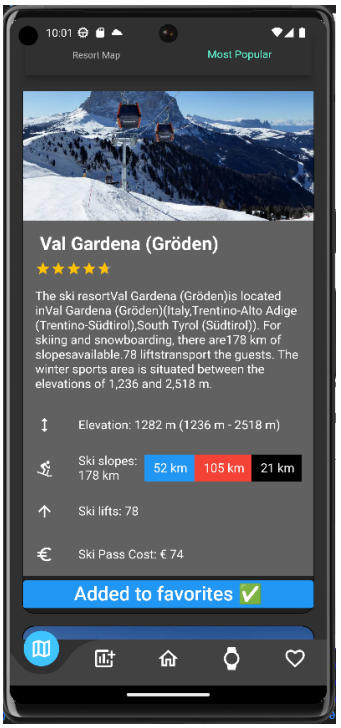




Displays a map pinpointing various ski resorts in Italy. Each resort is marked with an icon, the size of which indicates its popularity based on user reviews. Clicking on an icon appears a pop-up that provides more details about that particular; from that pop-up a user can save the resort among his favorites or can leave a review.

Most Popular Page

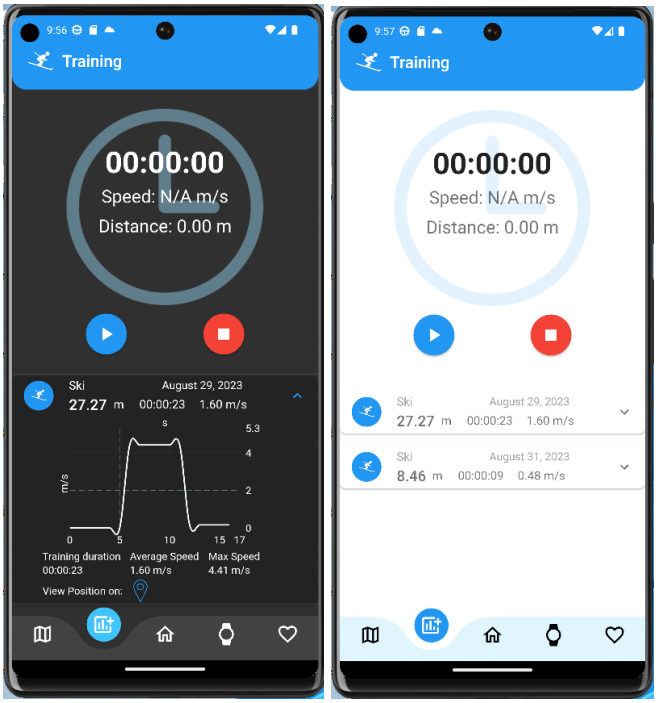


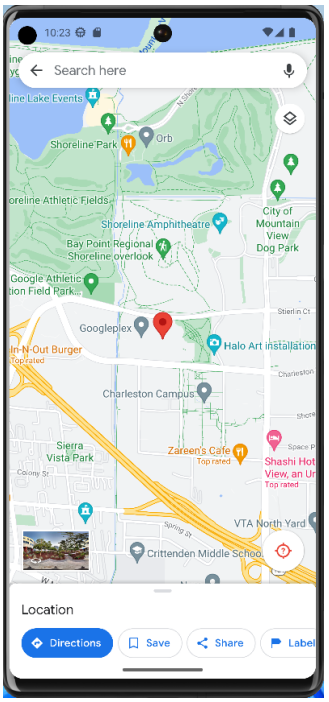


A list or grid view showcasing the top 10 ski resorts as rated by the app's community. It's a dynamic screen that changes based on user feedback and reviews. There is a button that allows you to add any resort of your choice to the favorites screen.

Furthermore there is a custom navigation bar that lets the user switch between two screens(resort map screen and most popular screen) conveniently.

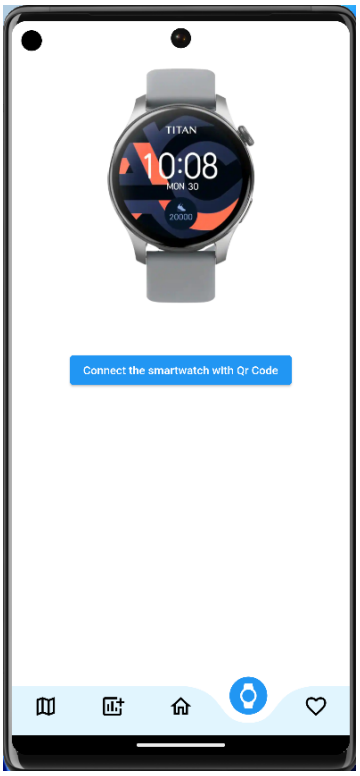
Training Page

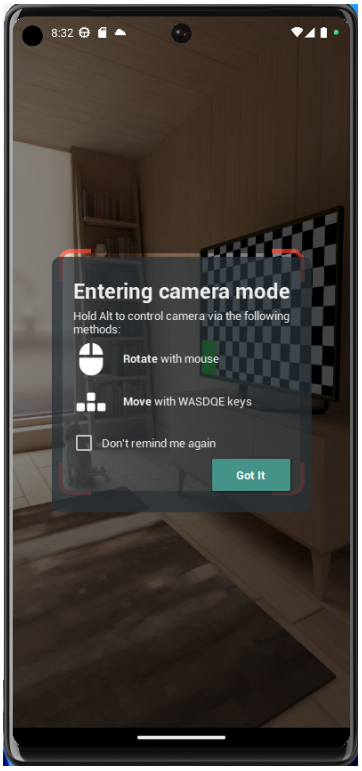




An interactive page where users can start, monitor, and stop their skiing training sessions. It provides real-time data and analytics like speed, distance, and more.
Once the workout is finished, it is entered in a workout list. All workouts are recorded and saved so the user has the history of all his workouts/performances available.
Every recorded workout contains information about altitude, speed, training time, some charts about speed and finally the position of the user. A user can see his position on the map through a dedicated button.

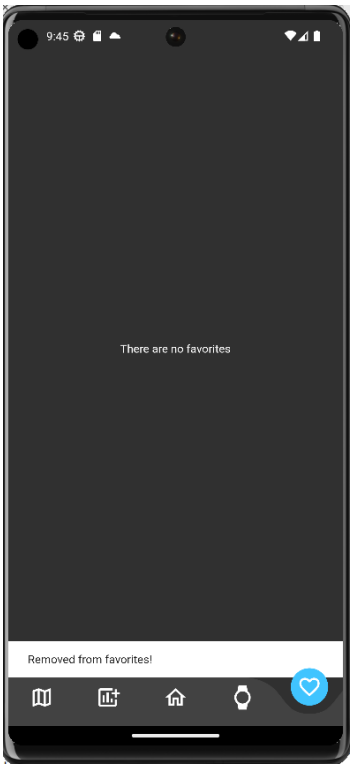
Smartwatch Connection Page

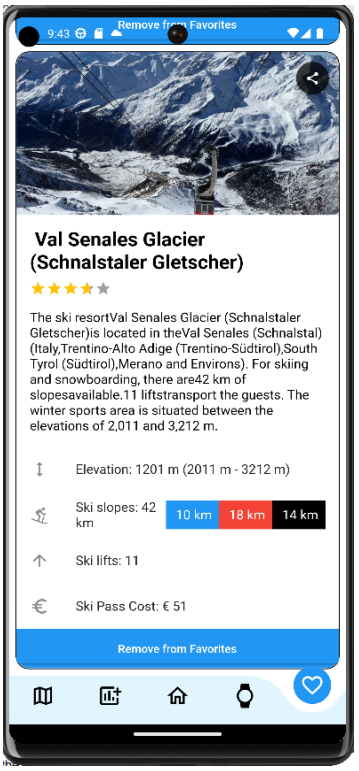


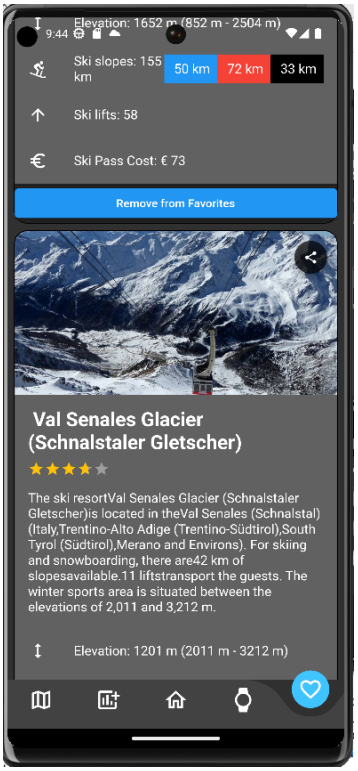


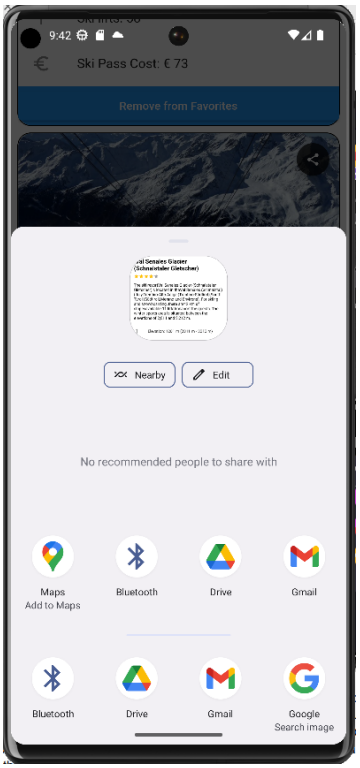
This is dedicated to pairing the user's smartphone with their smartwatch. It provides options for discovering, connecting, and managing the connection between the two devices.
By clicking on the appropriate connection button and after accepting the permissions, the camera opens with which it is possible to scan the qr code on the smartwatch and carry out the pairing.
From this screen it is also possible to log out of the smartwatch.

Favorites Page





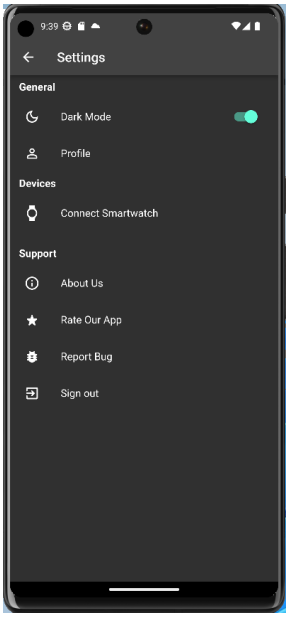


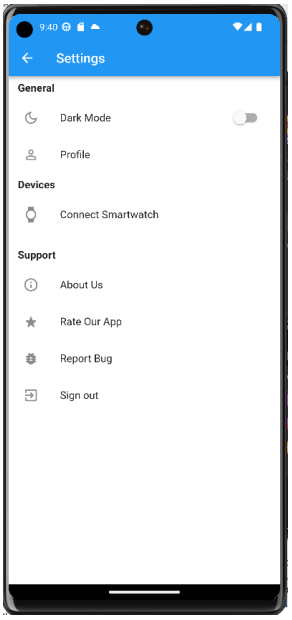


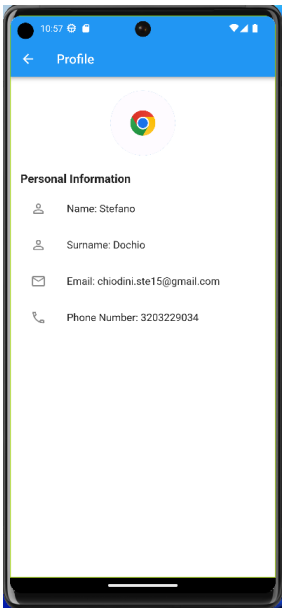
Here, users can view a list of their favorite ski resorts. This is directly linked to their profiles, enabling them to quickly access and manage the resorts they love the most.
From this page a user can share his favorites resorts; by clicking the share button the container will be transformed in an image that is shareable through the mobile phone

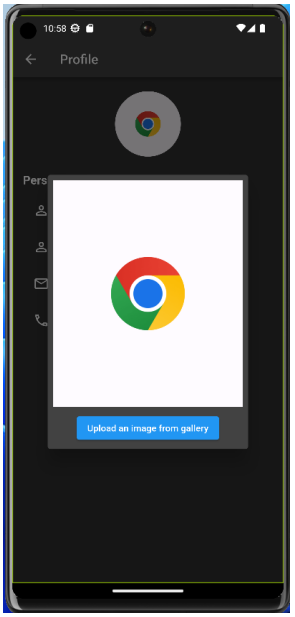
interface.In this way you can tell to anyone where you want to go and you can tell it by sending the image on whatsapp, instagram, telegram, bluetooth, email ecc.
You can also remove a resort from your favorites.

Settings Page







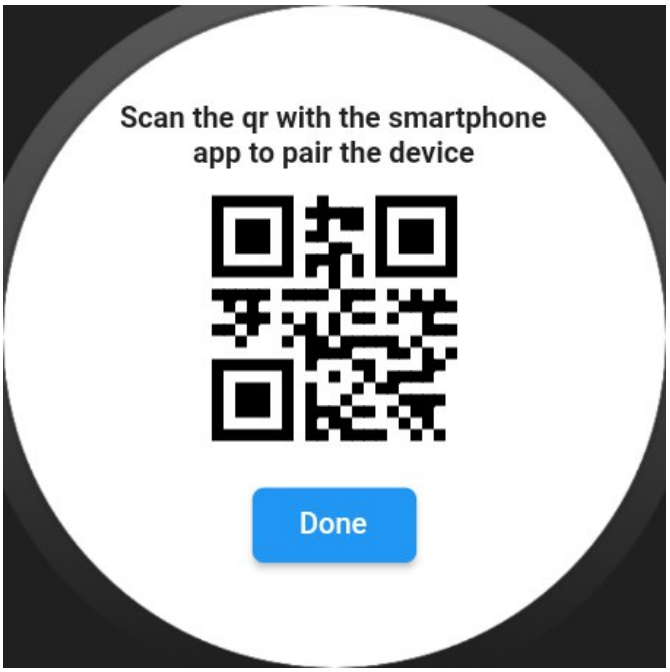


This screen allows users to personalize app settings, ranging from theme choices (light or dark) to managing profile details. A user can modify everything about his personal information; indeed, for example he is able to change his profile app picture with another image taken from a gallery.
The settings page screen also offers options to connect the mobile phone to other devices like smartwatch and to report bugs. The bug report is realized by exploiting the email channel; in particular we are using gmail to send an email with the bug description to the email team support.

Smartwatch UI

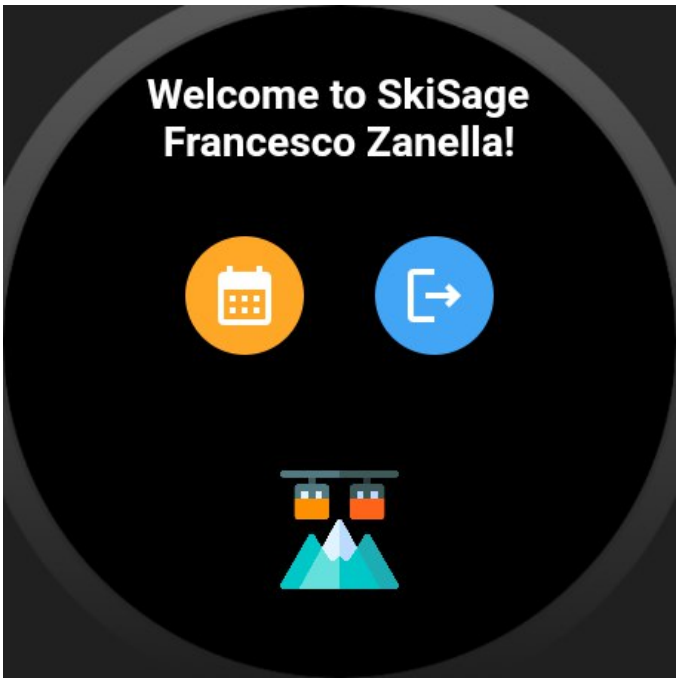
We decided to add smartwatch layout support because in our opinion this provides a more immersive user experience during training sessions, ensuring that users have instant access to their data without the need to constantly check their phones.
The whole process of connection and data exchange is managed through firebase

Connection Page



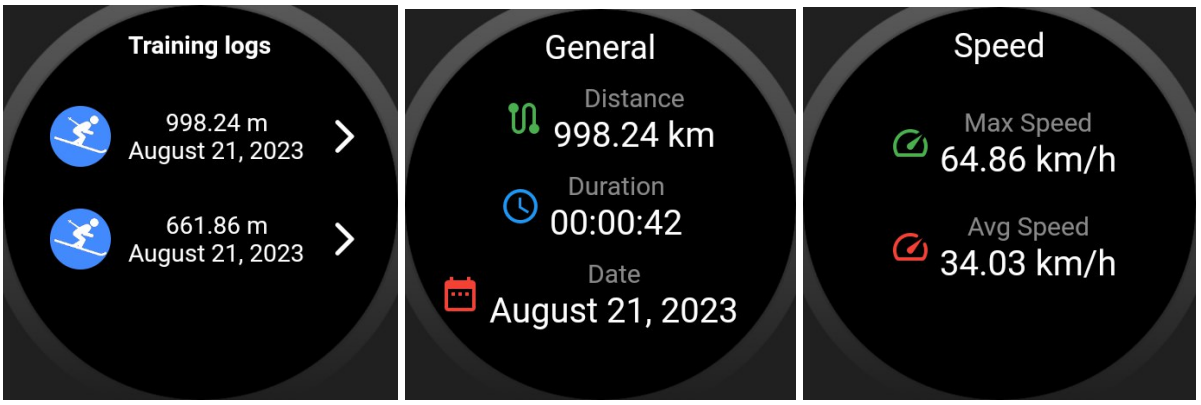
The Connection Page displays a QR code for users to scan. Once scanned using a phone, it establishes a pairing with that device. This connection is maintained until the user voluntarily disconnects, either from the phone or the watch itself.

Home Page



The Home Page offers a straightforward menu, showcasing the name and icon of the app. It features two buttons - one for disconnecting the paired device and another that navigates to the Training Page.

Training Page

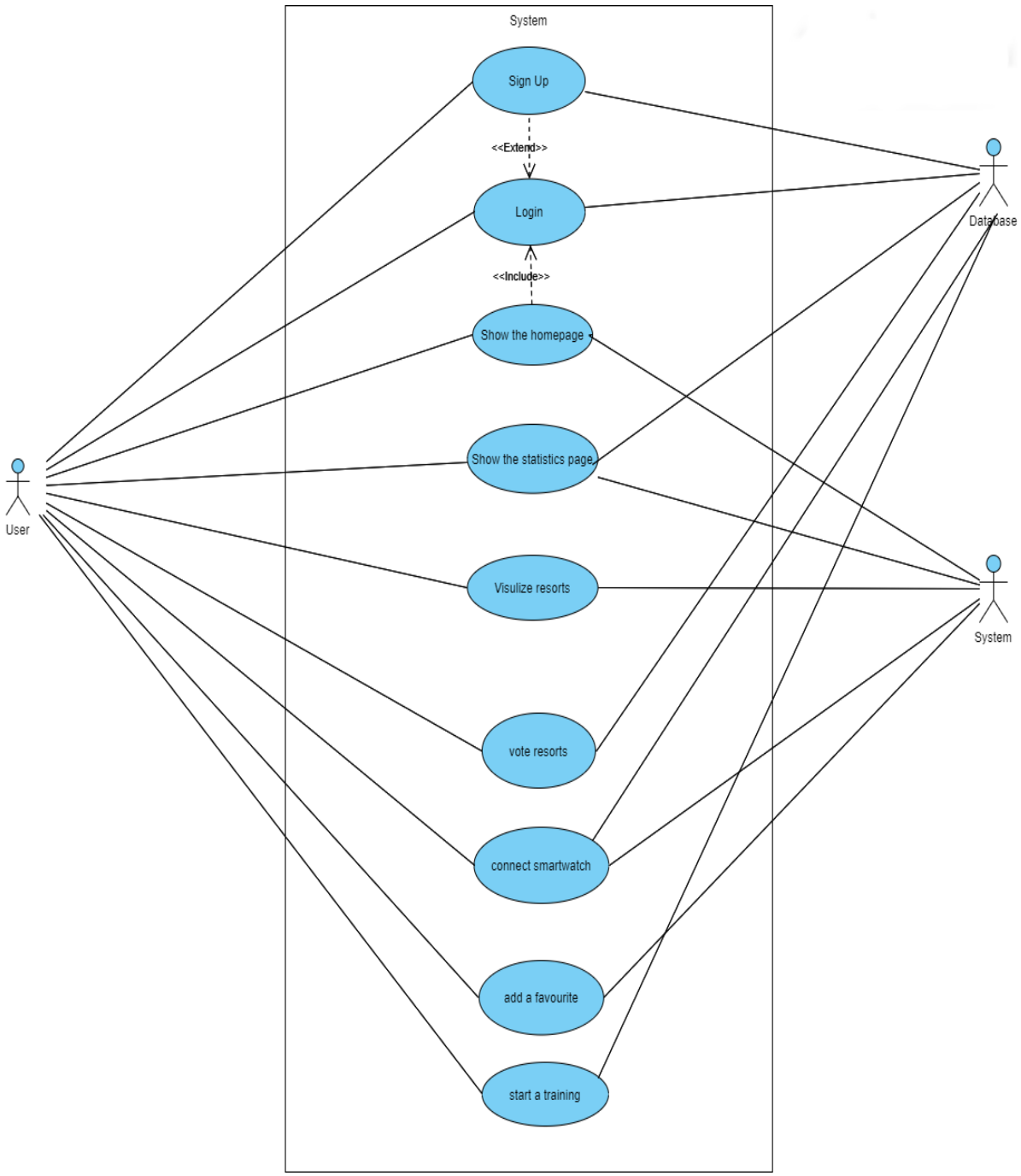


On the Training Page, users can view all their training sessions. A list presents these sessions, and for each training entry, users can access statistics such as speed, date, and duration of the training, among others.

UML Diagrams

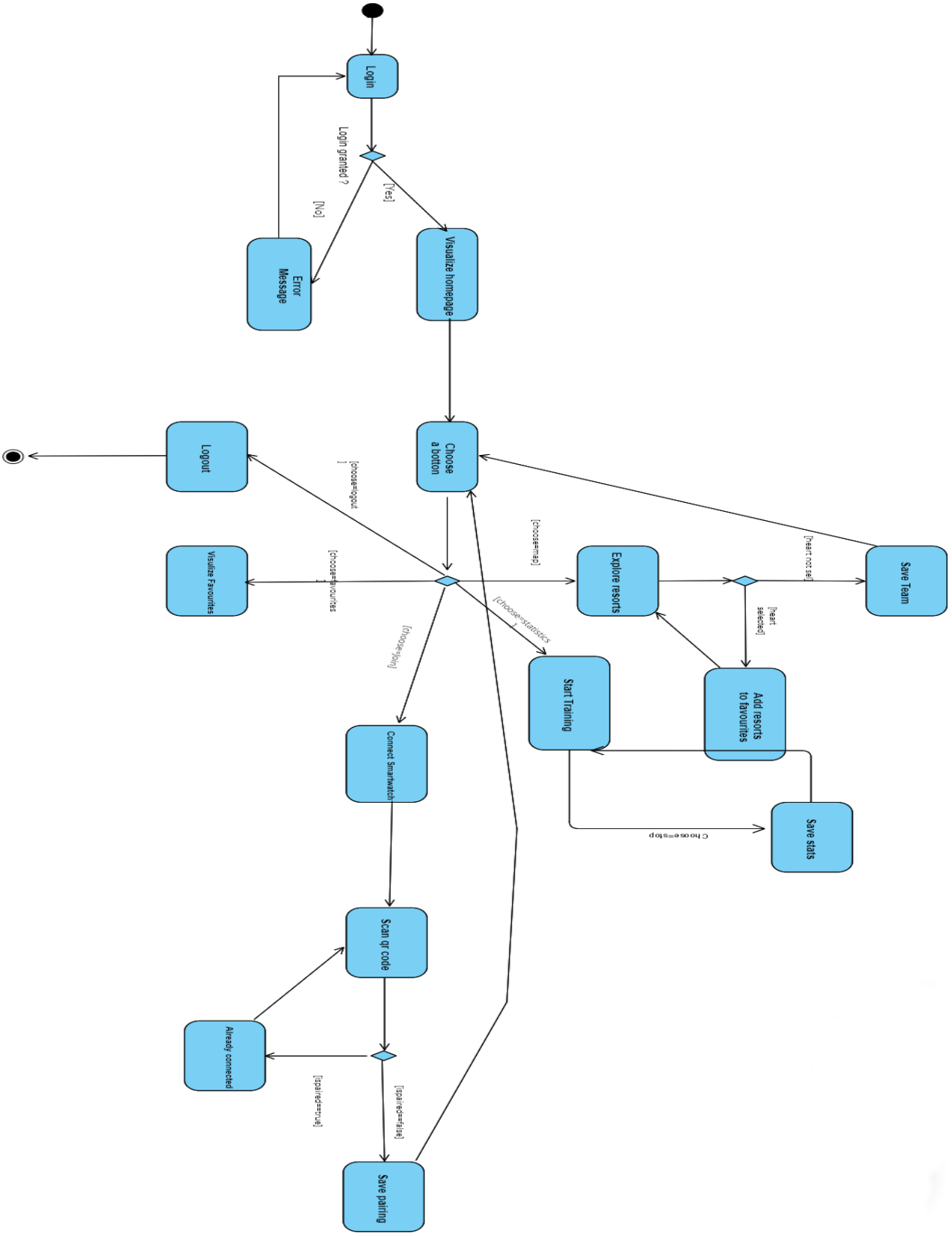
Use-Case diagram

The Use Case Diagram explains the typical interactions between the user and the system itself.



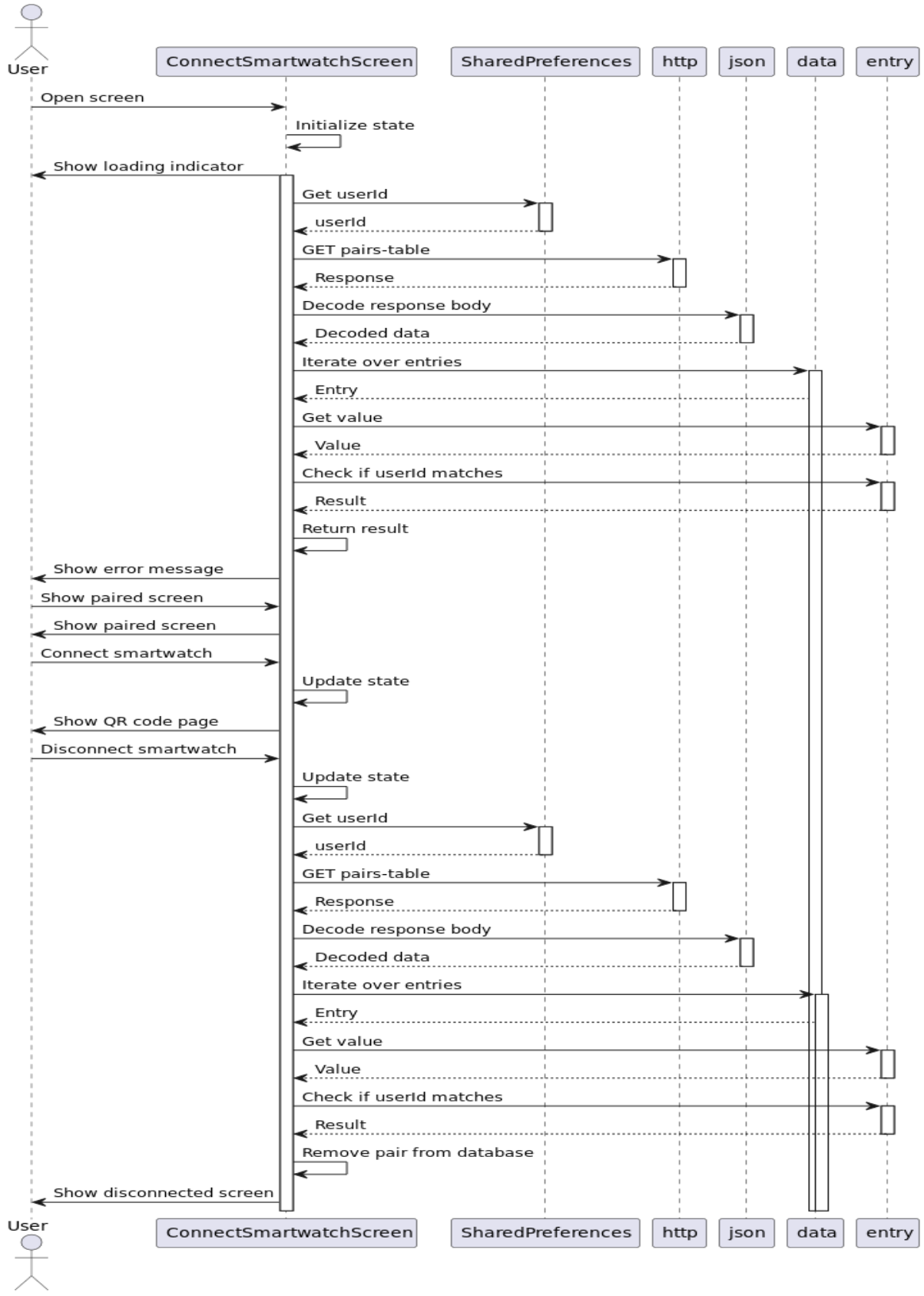
Activity Diagram

In the following Activity Diagram we want to describe in a clear and concise main processes carried out by the application.

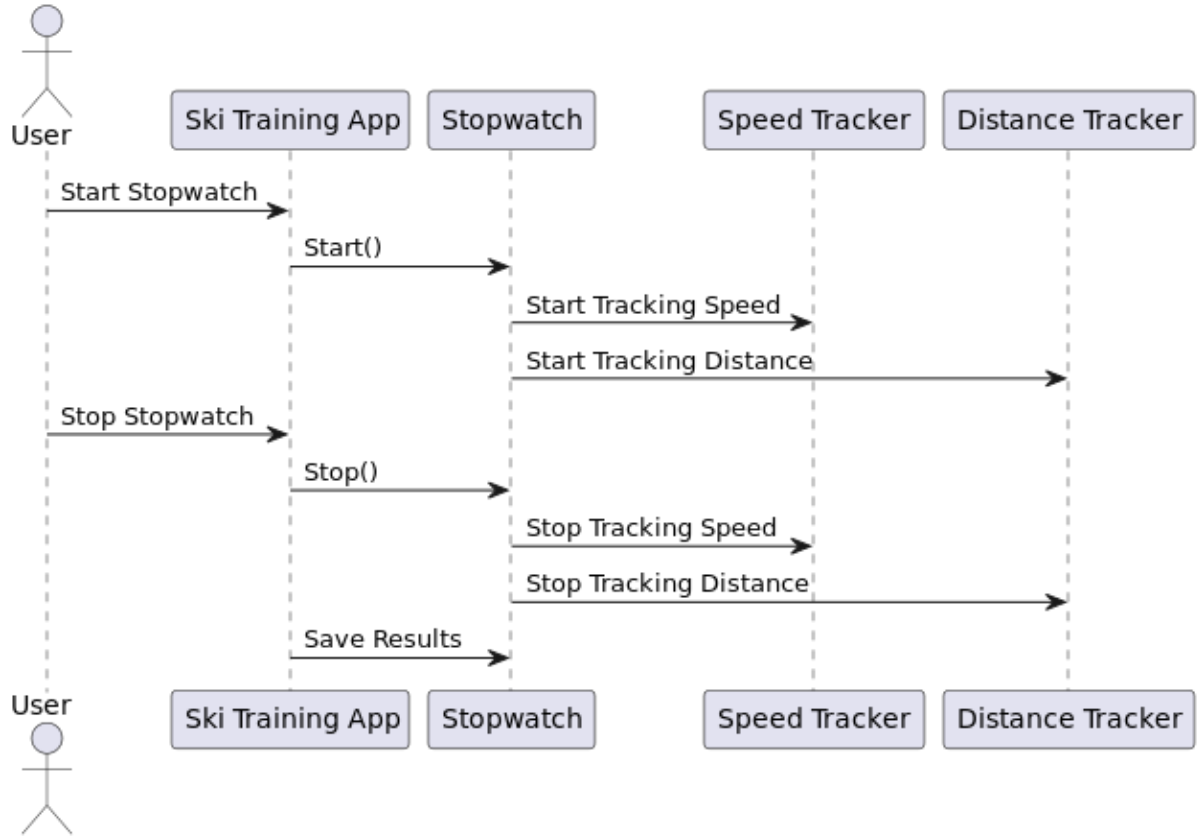


Sequence Diagrams

The Sequence Diagram allows us to visualize how a system performs a certain action. In our case it is shown how it works the process of pairing the smartphone app with the smartwatch.



This sequence diagram shows the process of starting, stopping and saving a ski training.



Testing

Four distinct test categories will be executed:

- **Unit Verification:** This focuses on confirming that the software meets the business objectives and can deliver the desired outcomes for end-users. It's a pivotal step in deciding whether the software is ready for delivery.

- **Widget Examination:** This revolves around ensuring the widget's user interface operates and appears as anticipated. This examination is multifaceted, involving various classes, and necessitates a specific test environment to mimic the widget's lifecycle accurately.

- **System Integration Tests:** The primary objective here is to confirm that a fully combined system complies with specified criteria. This is carried out within a combined software and hardware setting to ascertain the overall system's seamless operation.

- **User Tests(final phase):** This process is geared towards ensuring the mobile application operates to the predefined business standards. A selected team of testers is engaged to interact with the application to gauge its functionality**.**

Unit Analysis

The objective during this stage is to scrutinize the complete business logic segment of the software. The application contains a compact model to accurately decode data from the back-end. Hence, unit analyses will be pivotal to verify these interactions. To accomplish this, we'll simulate responses for every model class and use a series of assertions to ensure the data is accurately managed. Given the critical nature of these elements, our benchmark is to cover at least 90% of the lines, though our ultimate goal is 100%.

Widget Evaluation

The primary purpose of widget evaluations is to verify the correct instantiation and population of widgets. This is done by supplying them with data and, using precise assertions, ensuring they display as intended. Alongside the integration testing phase, we'll introduce simulated back-end data. This fictional data will be rendered and subsequently checked. Owing to certain components that won't undergo testing—since they are part of externally tested packages—we anticipate a coverage of a minimum 85%.

Integration Analysis

This stage augments the previous widget evaluation. Here, our lens shifts to the interaction between diverse widgets rather than an isolated widget. In practical terms, the process

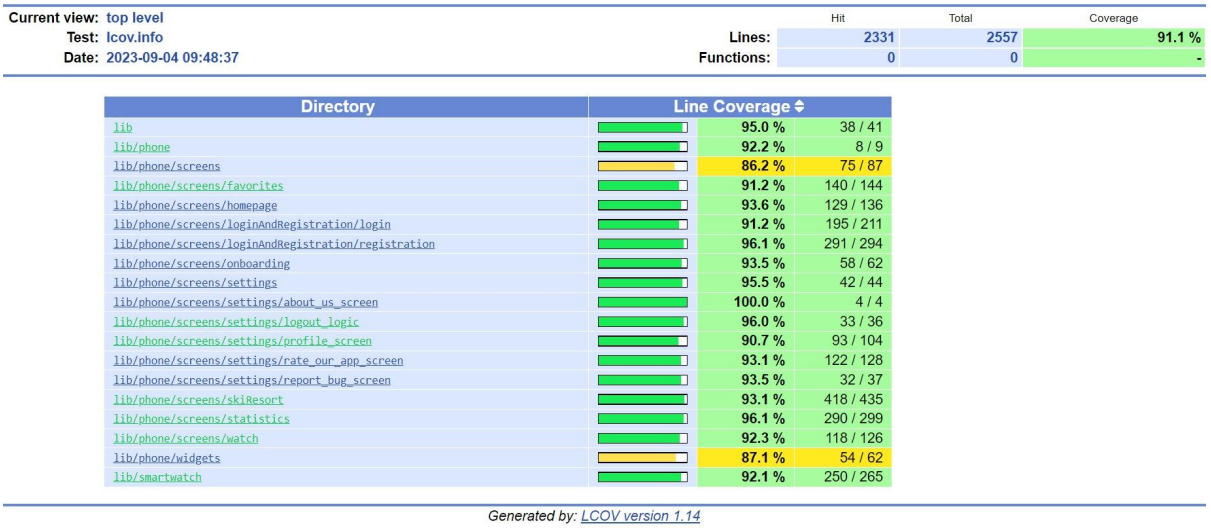
commences from a specific page. Through a combination of taps, text inputs, and gestures, we'll ascertain that operations proceed as expected.

User Testing

In the User testing phase,which is the final phase, the application has been extended to a select group comprising family members and close acquaintances. This strategy is adopted to ensure the app's functionality from the perspective of potential end-users, offering invaluable insights into its real-world performance, usability, and user experience. Given that these individuals are closely linked to us, their feedback tends to be candid and constructive. This not only ensures a smoother user experience but also adds a layer of personal assurance to the quality and reliability of our application.

Coverage Analysis

After implementing all the test designs, we conducted a code coverage analysis. The results revealed that our code coverage stands at 91%(considering the code both for smartwatch and mobile app). Suddenly we manually analyzed the uncovered code to ensure that the lines excluded from the tests were not pertinent for this type of testing: these lines are stuff like routes, color, banal statements ecc. We utilized Flutter's testing features to generate the coverage data, followed by the creation of an HTML report for easier human interpretation using LCOV; indeed the flutter report is pretty unreadable. This report provides the line coverage percentage and highlights the missing lines for each source file.



		Hit	Total	Coverage
Current view: top level				
Test: lcov.info	Lines:	2331	2557	91.1 %
Date: 2023-09-04 09:48:37	Functions:	0	0	-

Directory	Line Coverage	
lib	95.0 %	38 / 41
lib/phone	92.2 %	8 / 9
lib/phone/screens	86.2 %	75 / 87
lib/phone/screens/favorites	91.2 %	140 / 144
lib/phone/screens/homepage	93.6 %	129 / 136
lib/phone/screens/loginAndRegistration/login	91.2 %	195 / 211
lib/phone/screens/loginAndRegistration/registration	96.1 %	291 / 294
lib/phone/screens/onboarding	93.5 %	58 / 62
lib/phone/screens/settings	95.5 %	42 / 44
lib/phone/screens/settings/about_us_screen	100.0 %	4 / 4
lib/phone/screens/settings/logout_logic	96.0 %	33 / 36
lib/phone/screens/settings/profile_screen	90.7 %	93 / 104
lib/phone/screens/settings/rate_our_app_screen	93.1 %	122 / 128
lib/phone/screens/settings/report_bug_screen	93.5 %	32 / 37
lib/phone/screens/skiResort	93.1 %	418 / 435
lib/phone/screens/statistics	96.1 %	290 / 299
lib/phone/screens/watch	92.3 %	118 / 126
lib/phone/widgets	87.1 %	54 / 62
lib/smartwatch	92.1 %	250 / 265

Generated by: LCOV version 1.14

Possible future developments

In conclusion we can say that our application's source code captured many of the desired features. Yet, due to time restrictions, we chose to delay certain supplementary developments.
This could be a list of possible future functionalities that can be implemented:

- **Integration with Multiple Single Sign-On (SSO) Platforms:** While we currently support Google SSO, future integrations can encompass popular platforms like Facebook, Twitter, and Apple ID. This will provide users with a diverse range of login options, ensuring faster and more convenient access.

- **Weather Forecast Integration**: A real-time weather forecast feature can be invaluable for skiers. By integrating weather data specific to each resort, users can better plan their skiing trips, ensuring they choose the best times for optimum skiing conditions.

- **Multi-Language Support**: To cater to a global audience, the app can be made available in multiple languages. This would make SkiSage more accessible to non-English speakers, enhancing its usability across different regions.

- **Advanced Training Metrics**: While the app already offers comprehensive training statistics, future iterations can integrate more advanced metrics, such as altitude changes, gradient analysis, or even heart rate monitoring if synced with appropriate devices.

- **Integration with Skiing Gear Shops**: Users could have the ability to see where they can purchase or rent skiing gear directly through the app.