POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Design and Implementations of Mobile Applications Project

Design Document

Riccardo Inghilleri (10713236) Manuela Merlo (10670533)

Professor: **Luciano Baresi**

Tutor: **Marco De Rossi**

Academic Year: 2022-2023

Contents

1 | Introduction

1.1. Purpose

The purpose of this article is to analyze our app, providing a summary of its functionality and structure. We will begin with a description of the design, which captures the functionalities of the application. We will then go on to illustrate the architectural design of the system, highlighting in particular the adoption of the Model-View-ViewModel (MVVM) pattern. Then, we will show the main capabilities and functionalities that users will interact with. To give an idea of how data is organized and managed within the system, we will present our Data Model. Integrations play a key role in expanding the functionality of the system, so one section is devoted to the integration of external APIs. Then, we will turn to the detailed representation of the user interface; in particular, we will show the application on different devices, such as iPad and iPhone. To conclude our exposition, we will delve into testing, highlighting the strategies adopted to ascertain the robustness and reliability of the system.

1.2. Project Description

In recent years we have observed a significant increase in people seeking to improve themselves physically, mentally, and emotionally. However, one of the common barriers people face is a lack of motivation or difficulty integrating these health habits into their daily routine. Our app aims to address this problem by keeping track of these activities, but also making them a game. By introducing game elements, such as point systems, records and leaderboards, we have turned the often monotonous task of health tracking into a fun and competitive activity.

However, our goal goes beyond just having fun. Habist Tracker's goal is to instill lasting habits. We have recognized that people, when faced with a challenge in a game format, often show higher levels of engagement and commitment. With each activity recorded, point earned, or achievement unlocked, users are not just playing the game, they are reinforcing habits that contribute to a healthier lifestyle.

Moreover, the social aspect of our app plays a critical role in maintaining user motivation. By allowing friends to connect, compare scores, and challenge each other, we're fostering a community of health-conscious individuals who push and support one another.

1.3. App Features

Our app is designed to offer a comprehensive experience by integrating health tracking, social engagement, and personalized user management. The features are organized into key categories for clarity:

User Account Management:

- **Multiple Sign-In Methods**
  Description: Users can register using email and password, Google, or Facebook. Once registered with one service, others can be linked from the settings view.
- **Profile Customization**
  Description: Users can enhance their profiles by adding images and personal details, such as birthdate, gender, weight, and height.
- **Account Management**
  Description: Users have the option to update their credentials, request password resets, or delete their account.

Data & Tracking

- **Health Data Integration**
  Description: Integrates with HealthKit API for automatic data retrieval, negating manual input.
- **Point System**
  Description: Awards points based on the diversity and duration of healthy activities, motivating users through gamification.
- **Detailed Activity Breakdown**
  Description: Offers an expanded view of scores, detailing all contributing activities for transparency.
- **Comprehensive Score Views**
  Description: Allows users to view daily and weekly scores, detailed scores for each activity, and personal records for each activity type.

Social Features

- **Social Integration**
  Description: Facilitates community building by allowing users to send and accept friend requests.
- **Local and Global Leaderboards**
  Description: Engages users by allowing them to compare scores with friends and the broader app community.
- **Friend's Profile Insights**
  Description: Enables users to view a friend's daily scores, weekly scores, score trends over the past week, and activity records.

Notifications & Alerts

- **Customizable Notifications**
  Description: Gives users the autonomy to set notification frequencies.

Location and Points of Interest

- **User Location Display**
  Description: With user permission, displays the user's current location on the map within the app.
- **Nearby Points of Interest**
  Description: Shows nearby points of interest such as parks, gyms, and other sport activity areas on the map.
- **Search for Places of Interest**
  Description: Allows users to search for various places of interest based on keywords or categories.
- **Interactive Map Experience**
  Description: Provides an interactive map experience where users can pan, zoom, and tap on points of interest for more information.

1.4. Reference Documents and Websites

For the development and conceptualization of this application, several reference documents and online resources were consulted to ensure accurate implementation and adherence to best practices.

The Apple Developer Documentation was instrumental. It provided extensive insights on iOS application development standards and guidelines.

For the app's backend architecture and user authentication functionalities, the Firebase Documentation was indispensable. It offered detailed guidelines on data storage and user management.

The Healthkit Documentation played a pivotal role given the health-centric nature of our app. It provided critical information on how to handle health data appropriately.

Both the Swift and SwiftUI documentations were pillars of this project, guiding the app's construction and design within the Apple ecosystem.

For advanced practices, Swiftful Thinking and Hacking with Swift were constant companions. These platforms delve deep into the nuances of Swift programming and provide advanced techniques that were beneficial for the project.

Lastly, the developer community on Stack Overflow was of great assistance, especially when troubleshooting specific bugs and errors.

Together, these references provided a solid foundation, ensuring the application was not only robust but also followed industry best practices.

2 | Architecture

2.1. Architecture Introduction

Habit Tracker is an innovative application designed to leverage the capabilities of Apple's latest frameworks and languages.

2.2. Swift: Apple's Modern Programming Language

Swift is Apple's modern programming language tailored for its diverse ecosystem, which encompasses iOS, macOS, watchOS, and beyond. Key characteristics of Swift include:

- **Performance:** Swift is optimized for performance, ensuring rapid execution of code.
- **Security:** Safety mechanisms are in place, minimizing errors and enhancing app stability.
- **Concise Syntax:** Swift's syntax is expressive yet succinct, facilitating clearer and more maintainable code.

2.3. SwiftUI: The Evolution of User Interface Design

Built atop Swift, SwiftUI introduces a paradigm shift in how user interfaces are designed and built. It offers a range of features and benefits:

- **Declarative Approach:** With SwiftUI, developers declare the UI's appearance and behavior, fostering a direct relationship between code and visual components.
- **Integration:** SwiftUI can be effortlessly combined with existing frameworks, amplifying its versatility.
- **Real-time Previews:** Adhering to the "What You See Is What You Get" (WYSIWYG) principle, SwiftUI's previews allow developers to instantly see the impact of their code changes.

- **Adaptability:** One of SwiftUI's hallmarks is its ability to gracefully adapt UI components to various screen sizes and device types without extensive manual adjustments.
- **Consistency:** Applications built with SwiftUI possess a uniform look and feel across the entire Apple ecosystem.
- **Responsiveness:** SwiftUI ensures that the UI efficiently caters to varying device orientations and resolutions.
- **Dynamics:** Built-in animations, transitions, and gestures in SwiftUI make for a fluid and captivating user experience.

An application built with SwiftUI is meticulously structured, ensuring seamless integration between design and functionality. At the heart of this structure is the "App" protocol, which serves as the main entry point of the application. This is followed by the "Scene" protocol, which represents a user interface space, such as a window or document.

Scenes comprise "views," the fundamental elements of the application interface. Each view defines a portion of the user interface and, in SwiftUI, is a reflection of the state of the application. When the state changes, the UI adapts dynamically, promoting a responsive and fluid user experience. SwiftUI's declarative syntax makes the view design process simpler and more intuitive, as developers simply define what the UI looks like in a given state and SwiftUI takes care of the rest.

To navigate the interface, SwiftUI provides "navigation views" that help users move between different screens or views. In addition, "stacks" are used to manage and organize the layout of views, ensuring a logical and appealing visual hierarchy.

2.4. Design Patterns: The MVVM Paradigm

Design patterns are essential paradigms in software engineering since they provide templates for solving recurring design problems. We decided to adopt the MVVM pattern to solve the complexities associated with event-driven programming for user interfaces.

2.4.1. A Structural Framework

MVVM represents a guiding principle for organizing and structuring software projects to create applications that are:

- **Maintainable:** Facilitates easy codebase management and modifications.

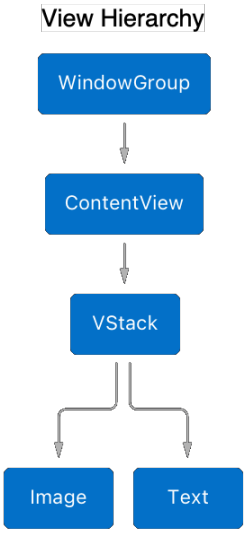


Figure 2.1: View Hierarchy

- **Testable:** Encourages development practices that support thorough unit testing.
- **Extensible:** Lays a foundation that allows seamless addition of features.

A core advantage of MVVM is its promotion of clear separation between the user interface logic and the business or back-end logic. This distinction mitigates the challenges associated with monolithic view controllers and offers a cleaner and more modular codebase.

2.4.2. MVVM Components

Model: the cornerstone of data representation, typically implemented as simple classes or structures. It remains detached from the complexities of application logic,limiting itself to data encapsulation.

View: serving as the user interface layer, the View is primarily responsible for rendering UI elements and controls.

ViewModel: a crucial component that bridges the Model and the View. It shoulders the responsibility of managing the display logic, acting as an intermediary that formats and prepares data for display.

2.5. Repository pattern

The Repository design pattern is a data access pattern used in software development. Repository design pattern isolates the domain from caring about how storage is imple-

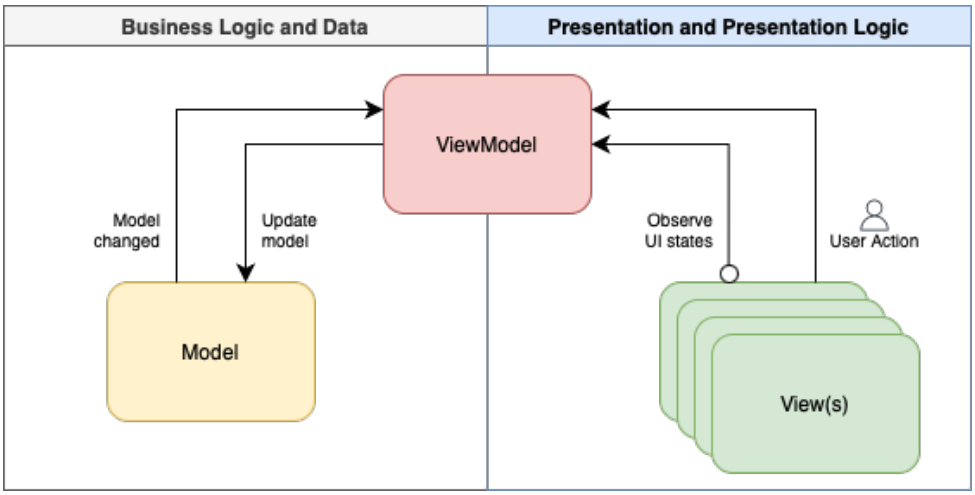


Figure 2.2: MVVM Paradigm

mented so all objects retrieved can be treated like an in-memory collection. This makes it very useful for testing.

What are Repositories?

Repositories are classes or components that encapsulate the logic required to access persistence store. Repositories, in practice, are used to perform database operations for domain objects (Entity and Value types)

Purpose of Repositories

- Centralize common data access functionality.
- Provide better maintainability.
- Decouple the infrastructure or technology used to access persistence store from the domain model layer.
- Make your code testable, reusable, and maintainable.

The Repository Pattern

After implementing the repository pattern, the Repository pattern puts a façade over the data access layer so that the rest of the application code can be shield from having to know how persistence works. The Repository pattern makes the code programmatically testable. If makes unit testing much easier since it is possible to implement mock repositories that return fake data instead of data from the database.

Objectives of the Repository Pattern

1. Maximize the amount of code that can be tested with automation.
2. Isolate the data layer to support unit testing.
3. Apply centrally managed, consistent access rules and logic.
4. To implement and centralize a caching strategy for the data source.
5. Improve the code's maintainability and readability by separating business logic from data or service access logic.
6. Use business entities that are strongly typed so that you can identify problems at compile time instead of at run time.
7. To associate a behavior with the related data.
8. To apply a domain model to simplify complex business logic.

We have used this pattern to implement the main functionalities of the app like the authentication part through APIs and the retrieval of data through Firebase.

In the sequence diagrams that will follow in the documentation, we will follow all the chain view model → repository → data source.

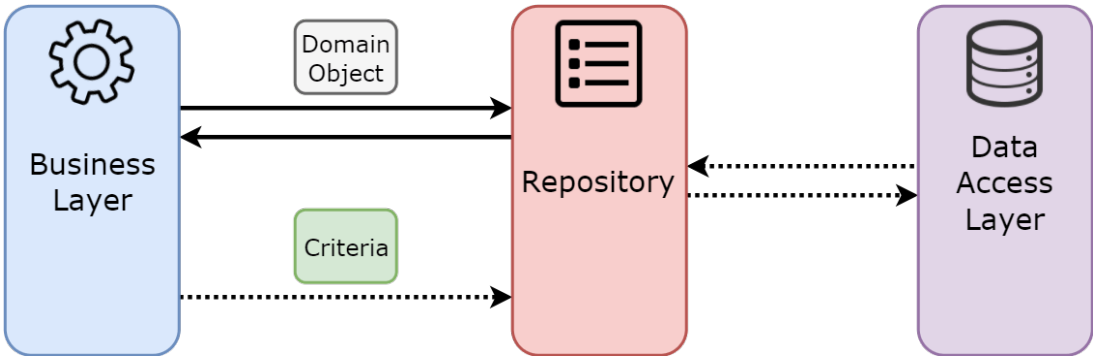


Figure 2.3: Repository Pattern

2.6. Data Model

In developing our application, we carefully structured the data using specific classes. These classes represent the Model component of the MVVM (Model-View-ViewModel) architectural pattern, as we mentioned earlier. Central to our application's architecture and functionality are two integral Firebase services: Cloud Firestore for our database needs and Firebase Storage for handling user-generated content.

Through careful attention to the requirements of the application and a thoughtful design approach to the visualization of each page, we outlined the following data objects:

User Model Description:

id: document's id.

email: user's email address.

username: chosen username of the user.

birthdate: date of birth.

sex: gender.

weight: user's weight.

height: user's height.

image: url to the user's profile image saved in Firebase Storage.

scores: An array of 8 integers representing the user's daily scores.

- **first integer:** Score for Monday.
- **last integer:** Cumulative score for the week.

actualScores: An array of objects representing the user's score for each daily activity.

records: An array of objects representing the user's activity records.

- **id:** identifier of the activity.
- **quantity:** value of the record.
- **timestamp:** when the record was logged.

friends sub-collection: Collection containing user's friends.

id: identifier of the other user.

status:

wanting: user has sent a friend request.

request: user has received a friend request.

confirmed: Both users have confirmed the friendship.

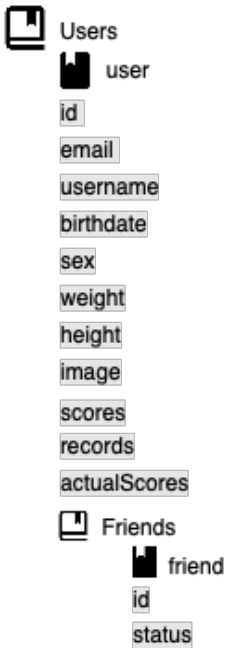


Figure 2.4: Data Model

2.7. Data Retrieval in Firebase

In our application, Firebase's Firestore database serves as a robust backend for data storage and retrieval. Firestore offers two types of data retrieval methods that fit different needs: on-demand single queries and real-time listeners. We've utilized both methods to optimize the user experience and application efficiency.

2.7.1. On-Demand Queries

For situations where we require specific, one-off pieces of information, we use Firestore's on-demand queries. A prime example is when we need to check if another user with the

same username already exists within the community. This is crucial for ensuring the uniqueness of usernames in our platform.

2.7.2. Snapshot and Snapshot Listener

Firestore provides two key features for real-time data retrieval: snapshots and snapshot listeners.

Snapshots: These represent the state of a Firestore document at a particular moment in time. Whenever there's a change in the data, a new snapshot gets generated. These snapshots are not just useful for reading data, but they also provide metadata about the document or query results, such as whether the document exists or the snapshot has been retrieved from the cache.

Snapshot Listeners: These are what bring the real-time capabilities into play. By attaching a snapshot listener to a document or collection, our application can automatically receive updates whenever the underlying data changes. This means that if another part of the application, or even another user, modifies the data, our attached listeners will immediately receive the updated snapshot.

In our application, real-time data retrieval is pivotal for a dynamic and responsive user experience. Therefore, we utilize snapshot listeners predominantly for monitoring data pertinent to the current user and for keeping track of a user's list of friends. By doing so, any changes made to a user's profile or their friends' list get reflected in real-time, without requiring manual refreshes or interventions.

2.8. Device's Services

Mobile devices offer a multitude of built-in services and features that developers can leverage to create applications. These native features, often tightly integrated with the device's hardware and operating system, provide a simplified user experience and improved performance. Listed below are some of the essential services provided by the device that Habits Tracker uses:

Notifications:

Introduction: local notifications are alerts delivered directly by the app without requiring an external server.

Details: they can be scheduled to inform users based on specific triggers. The primary advantage is that they function without the need for an internet connection and can be programmed to appear based on certain conditions met within the app.

Location Services

Introduction: most modern smartphones are equipped with GPS (Global Positioning System) capabilities, allowing them to pinpoint the device's location.

Details: location services are crucial for features that require geographic data. They can be used in a health context, for instance, to track outdoor runs or walks. Users typically have control over the granularity of location data shared, ranging from precise locations to broader geographical areas.

Internet Connectivity

Introduction: an app often needs the device's internet connectivity to fetch or send data.

Details: this built-in service allows the app to adapt its behavior based on the connectivity status.

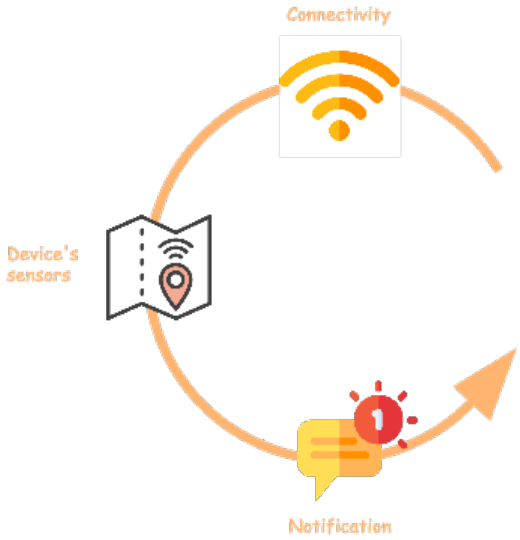


Figure 2.5: Device's Services

2.9. External Services

Modern applications often leverage various external services to enhance their capabilities, streamline user experiences, and provide scalable infrastructure. These services range from

authentication mechanisms to data integration frameworks, and their seamless integration is pivotal to the app's success. The following sections delve into specific services that are quintessential for the development of the proposed app.

2.9.1. Authentication Services

Authentication is paramount for securing user data and personalizing the user experience. Here are some of the most widely used authentication methods:

Firebase Authentication

Firebase Authentication offers an end-to-end identity solution for applications, allowing developers to authenticate and manage users seamlessly. It supports multiple authentication methods, from traditional email-password pairs to third-party APIs like Google and Facebook. Besides user sign-up and sign-in, Firebase Authentication provides functionalities for email and password updates and password reset. The seamless integration of these services allows a user to sign up with one method and subsequently link their account with other sign-in methods.

Google Authentication

Google Sign-In is a secure authentication system that reduces the burden of login for users by allowing them to sign in with their Google Account. It integrates seamlessly with Firebase Authentication, providing the backend services necessary for authenticating users with Google credentials.

Facebook Authentication

Facebook Login is an authentication solution that lets users sign in with their Facebook credentials. Once integrated with Firebase Authentication, developers can authenticate users with Facebook Login across multiple platforms.

Integration Note: one of the advantages of Firebase Authentication is its ability to integrate multiple authentication providers. A user can start by signing up with, for example, Google Authentication, and later decide to link their account with Facebook Authentication, thus providing them with multiple methods to access the app.

Note: although integrating Apple Sign In would have been a natural choice for our iOS native app, we've decided against its inclusion due to the requirement of an Apple Developer account.

Figure 2.6: Integration between authentication services

2.9.2. Data Storage

Cloud Firestore: Cloud Firestore is a flexible and scalable database for mobile, web, and server development from Firebase and Google Cloud. Unlike traditional databases, Firestore is a NoSQL database, which means it allows developers to store and synchronize data between users in real time. This makes it particularly suitable for applications that require real-time data updates. Cloud Firestore organizes its data as collections of documents, with each document containing data in key-value pairs. Its scalability, real-time capabilities, and intuitive structure make it a preferred choice for modern application development.

Firebase Storage:
Firebase Storage provides a powerful, simple, and cost-effective object storage service. It's built for app developers who need to store and serve user-generated content, such as photos and videos. One of its primary advantages is the seamless integration with Firebase's authentication tooling, ensuring that user content remains secure and accessible only to authorized users. Additionally, Firebase Storage is backed by Google Cloud Storage, which offers the reliability and speed that developers need for their applications.

2.9.3. HealthKit Framework

HealthKit is a robust framework developed by Apple, tailored primarily for its iOS and watchOS platforms. This comprehensive suite enables applications to interact with and manage the health and fitness data of the users. The intention behind HealthKit is to allow health and fitness applications to share data, offering users a holistic view of their health metrics.

Key Functionalities:

- **Centralized Data Storage:** HealthKit offers a secure and centralized storage,

known as the HealthKit Store, for all health data.It ensures data consistency across various apps, preventing unauthorized access. Allows apps to read and write health-related information, given the appropriate permissions.

- **Vast Array of Health Metrics:** HealthKit provides a plethora of predefined data types. These include, but are not limited to, body measurements, vital signs, nutrition, reproductive health, sleep, and workouts. Each data type is meticulously defined to ensure standardization across all apps.
- **Workout Management:** Beyond just health metrics, HealthKit has specialized provisions for recording and retrieving workout sessions. Allows fitness apps to read detailed summaries of user workouts, including type, duration, and active calories burned.
- **User-Controlled Permissions:** Data privacy and security are paramount in HealthKit. Before an application can access or modify a user's health data, explicit permission must be obtained. Users have the autonomy to grant or deny access to specific health metrics on a per-app basis.
- **HealthKit UI Integration:** The associated HealthKitUI framework facilitates the intuitive display of health data. It provides various view controllers to seamlessly integrate health data visualization within applications.

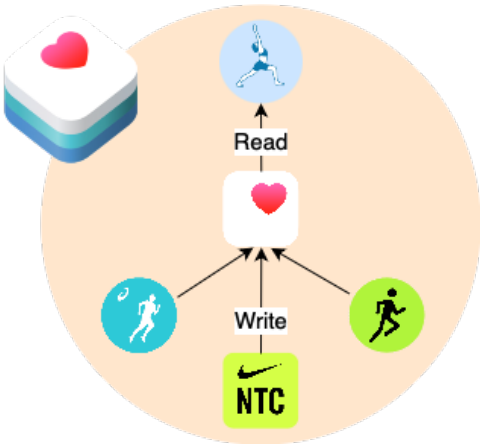


Figure 2.7: Healthkit External Service

2.9.4. MapKit Framework

MapKit is a robust framework developed by Apple, designed primarily for its iOS platforms. This comprehensive suite empowers applications to interact with and manage geographic map data and location-based services for users. MapKit aims to enable map-

related applications to provide users with dynamic and interactive mapping experiences.

Key Functionalities:

- **Interactive Map Display:** MapKit offers a versatile map rendering engine that allowed us to display interactive and dynamic maps within our application.
- **Location Services Integration:** MapKit seamlessly integrates with device location services, enabling applications to access the user's current location. It provides tools to display the user's location on the map, track movement, and calculate distances between points.
- **Annotations and Overlays:** we could add custom annotations, pins, and overlays to the map to provide additional context and information. We use this feature to mark points of interest.
- **Custom Map Styling:** MapKit allowed us to customize the map style to match the application's design aesthetics.
- **User-Interactive Features:** MapKit supports user interactions such as pinch-to-zoom, panning, and tapping on annotations for additional details. These gestures enhance the user experience and make the map intuitive to navigate.
- **User Privacy Considerations:** Data privacy and location access permissions are crucial aspects of MapKit. Apps must obtain explicit user permission to access location services and handle user location data responsibly.
- **MapKit UI Integration:** The associated MapKitUI framework provides view controllers and UI components for seamlessly integrating map views within applications. This allowed us to create cohesive and visually appealing map-based interfaces.

2.9.5. Lottie

Lottie is a library that parses animations exported as JSON with Bodymovin from Adobe After Effects and renders them natively on mobile and web platforms. Developed by Airbnb, Lottie offers app developers a new way to add crisp animations without the overhead of manual animation creation or the inclusion of large video files. It allows for dynamic, scalable, and easily alterable animations that can enhance user experience and engagement within the app.

2.9.6. Sequence Diagrams

Below, we present key sequence diagrams illustrating the interactions between users and various external services.

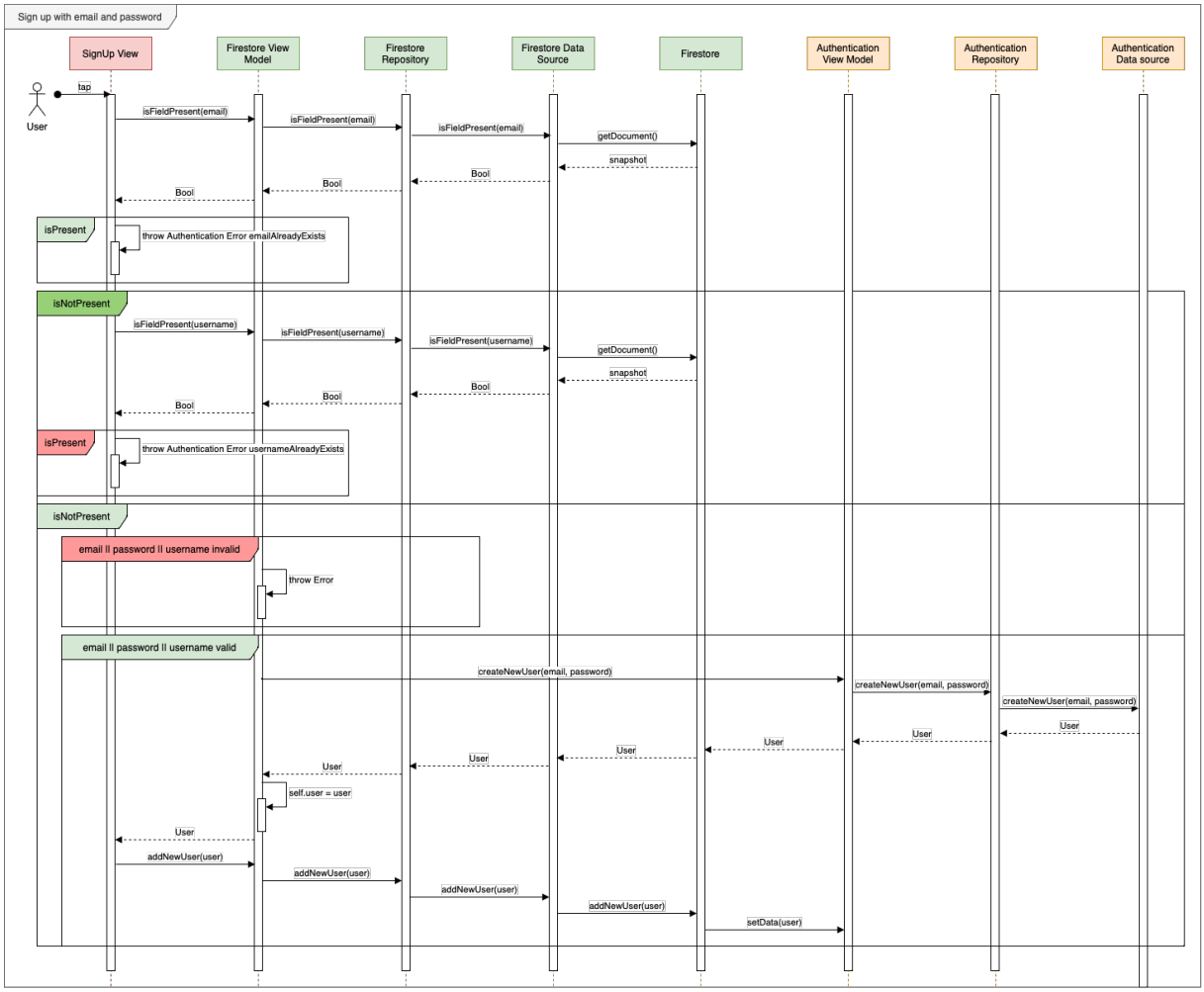


Figure 2.8: Sign Up

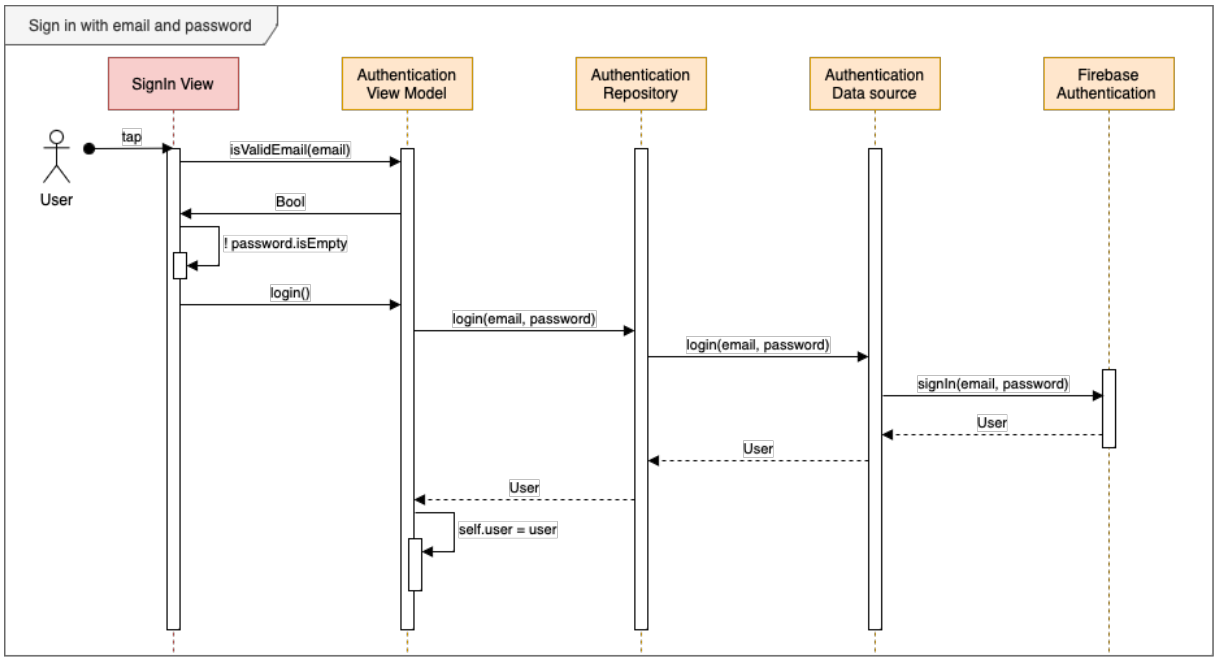


Figure 2.9: Login email

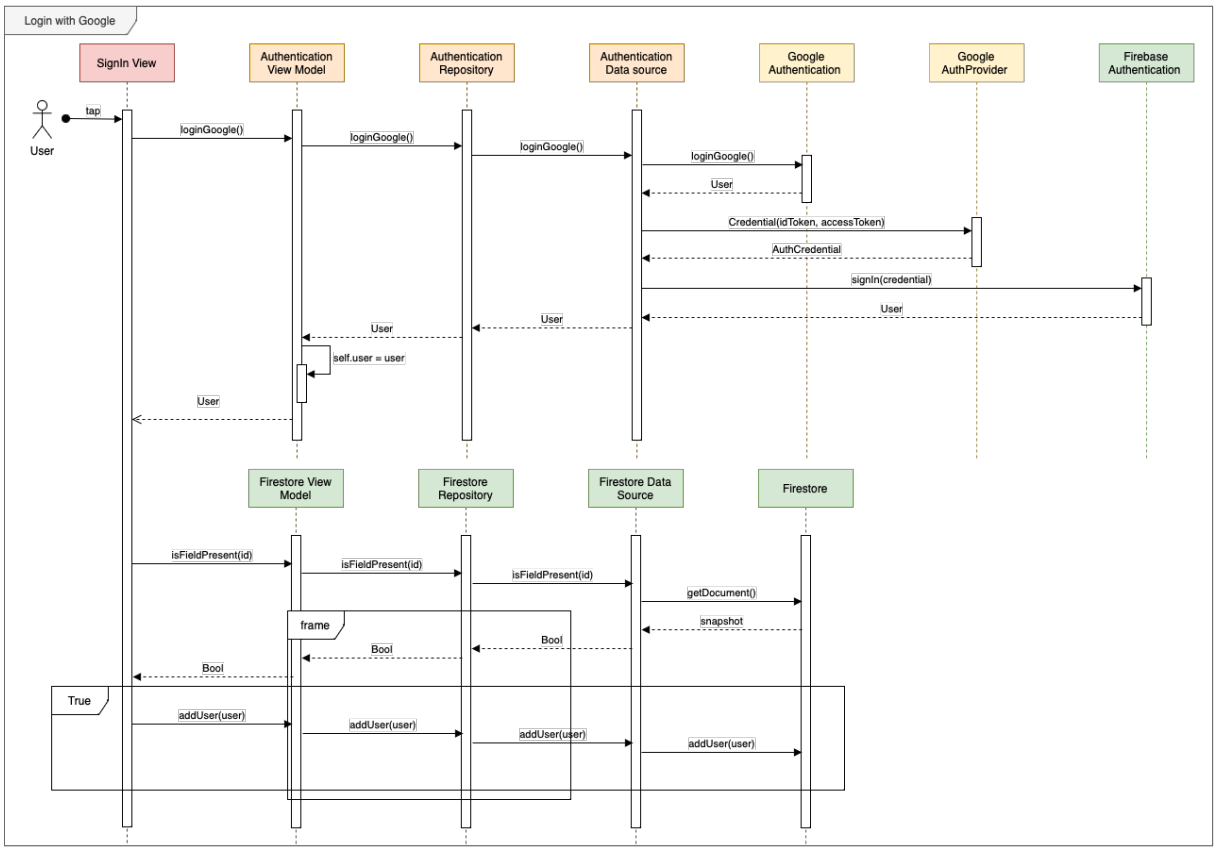


Figure 2.10: Login Google

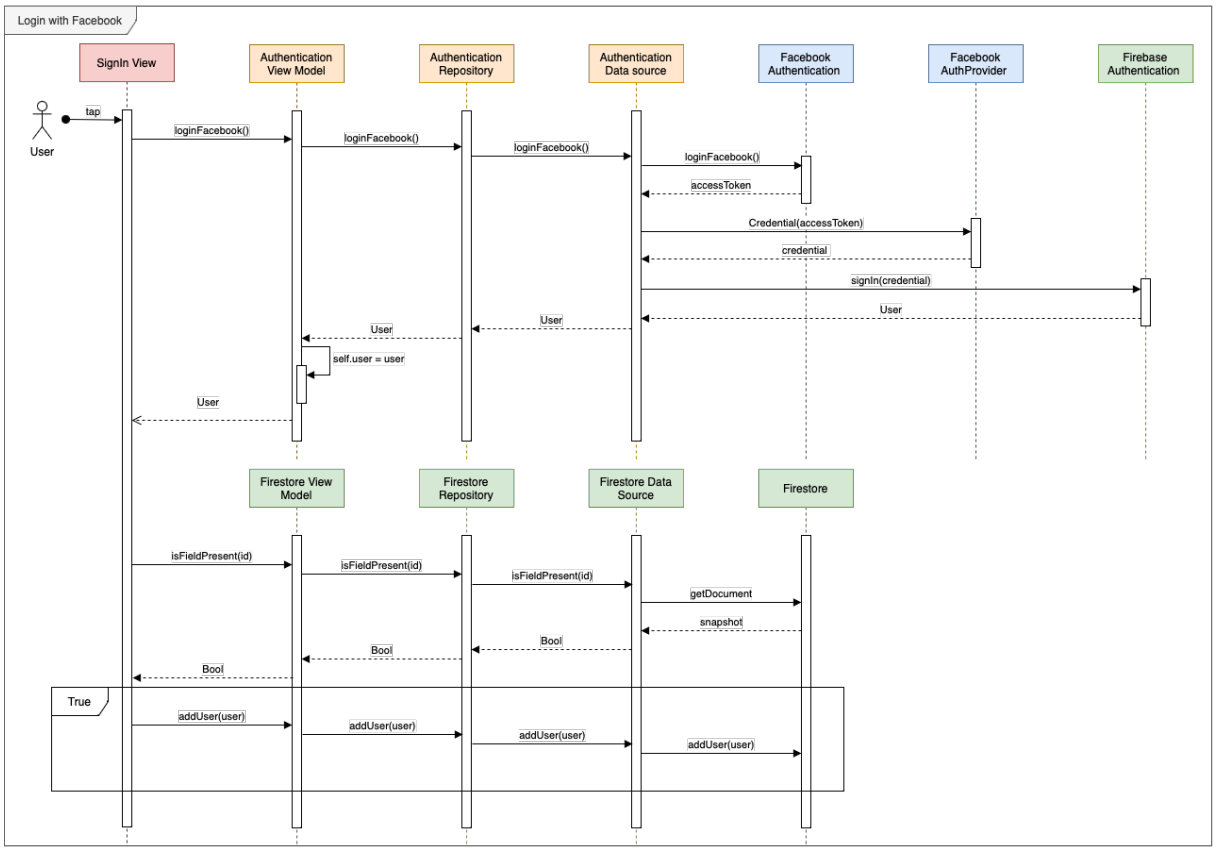


Figure 2.11: Login Facebook

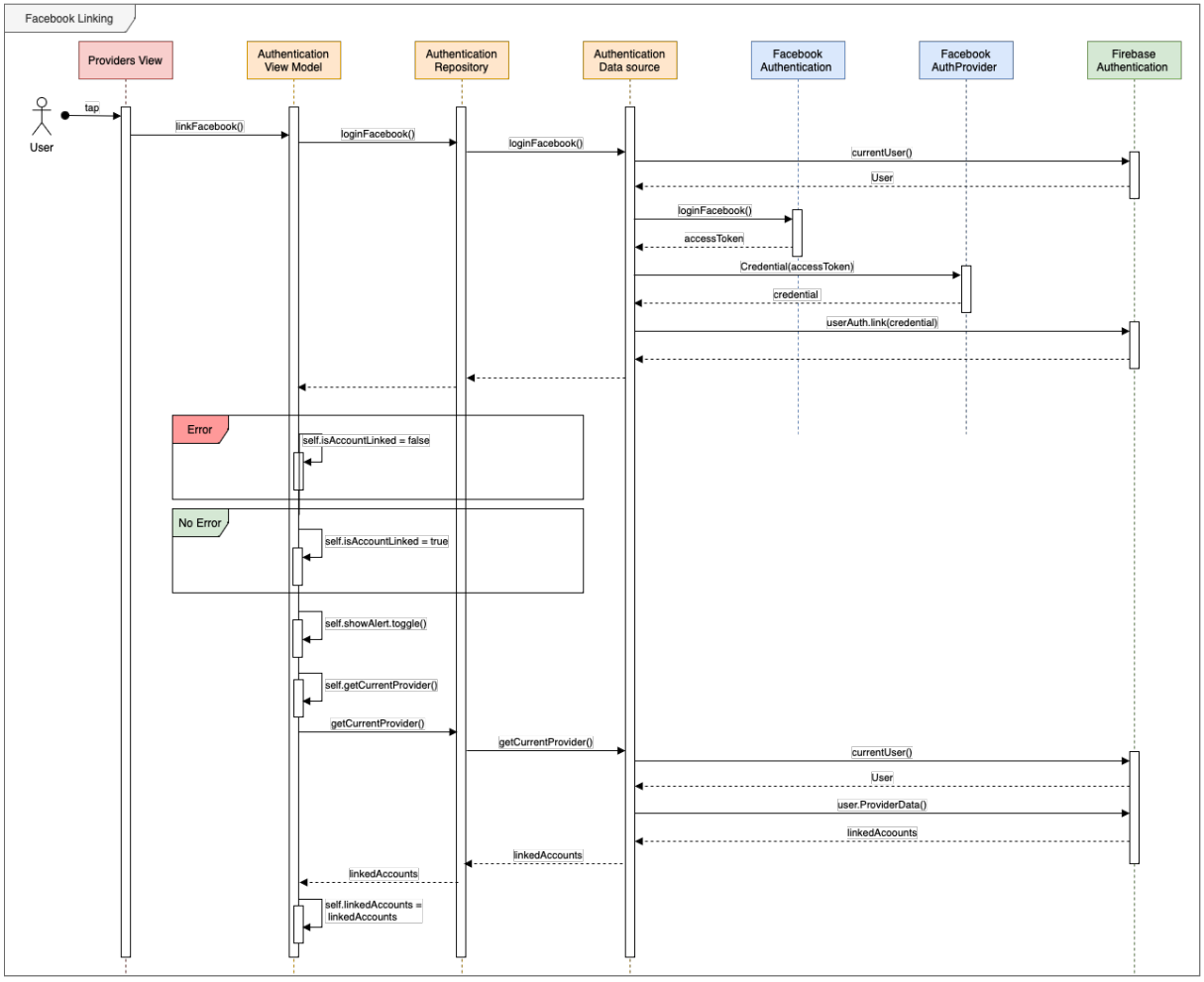


Figure 2.12: Facebook Linking

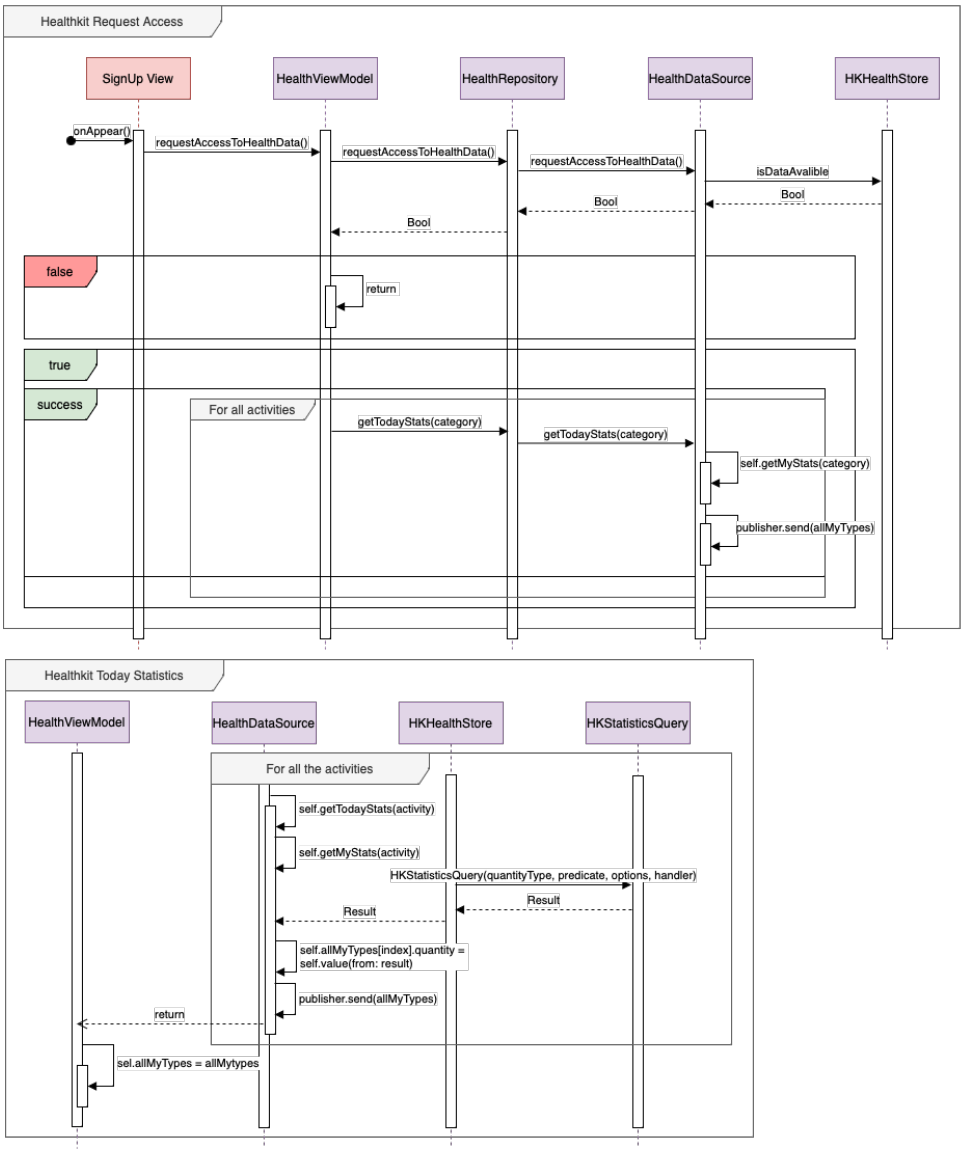


Figure 2.13: Healthkit interaction

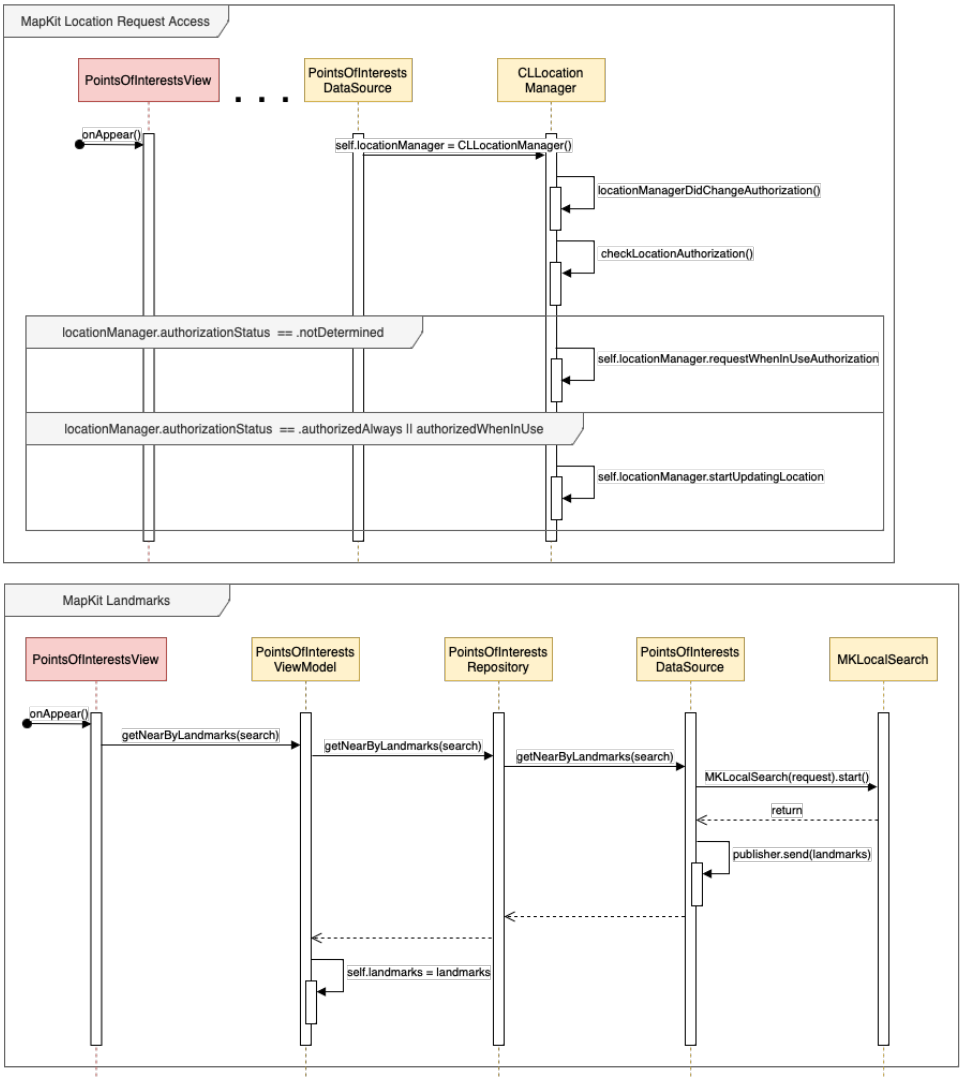


Figure 2.14: Mapkit interaction

3 | UI Design

The essence of a successful application often lies in the appeal of the user interface design, where graphics meets functionality and where user intent aligns with the application's response. Below we are going to detail the user interface of our application designed to visually fascinate and functionally facilitate.

In this section on user interface design, we present a comprehensive showcase of the application's views. These images provide a tangible representation of the aesthetic and functional elements of the design, highlighting the meticulous attention to detail employed in every corner of the interface.

But beyond the static images, understanding the dynamism of user interactions is critical. Therefore, we will delve into the flow between different views, explaining how one screen transitions to another and how user-initiated events trigger specific functionality. The flowchart representation provides clarity in understanding the user journey, making design choices transparent and justified.

3.1. iPhone Layout

3.1.1. Introduction View

The introduction view serves as the initial screen presented to users when no authentication is detected on the authentication server. This view is structured into three subviews, each highlighting a distinct feature of the app. These subviews change automatically based on a predetermined timer. To proceed to the sign-in process, users can simply tap the "Skip" button.

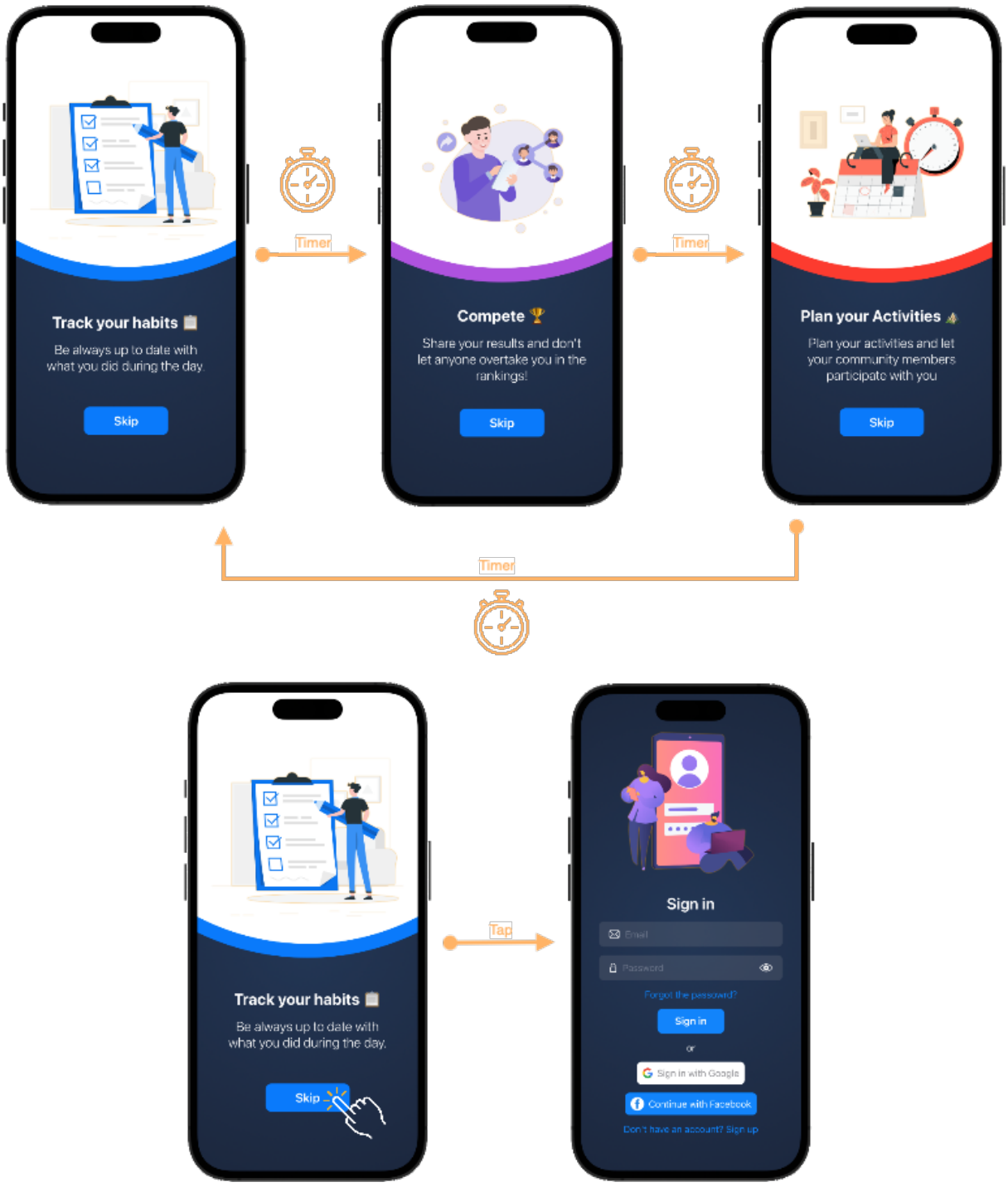


Figure 3.1: Introduction View

3.1.2. Sign In View

The Sign In view provides a comprehensive suite of functionalities to cater to the varied login preferences of the users:

- **Email and Password Login:** Users can securely log in using their registered email and password through Firebase authentication.
- **Google Authentication:** Facilitates a seamless login experience for users preferring to authenticate via their Google accounts.
- **Facebook Authentication:** For those who prefer using their Facebook credentials, this functionality provides a direct login through Facebook authentication.
- **Navigate to Sign Up:** For users who haven't registered yet, there's a direct link to navigate to the Sign Up page. The Sign Up page provides:
  - **Four text fields** for entering:
    * Username
    * Email
    * Password
    * Repeat Password
  - New users can sign up securely through Firebase authentication, ensuring their data is safely stored and protected.

  After successfully signing in through Google or Facebook for the first time, users are greeted with a pop-up prompting them to set a unique username.
- **Password Reset:** In case a user forgets the password, an option to reset it through Firebase authentication is available, ensuring they regain access to their account.

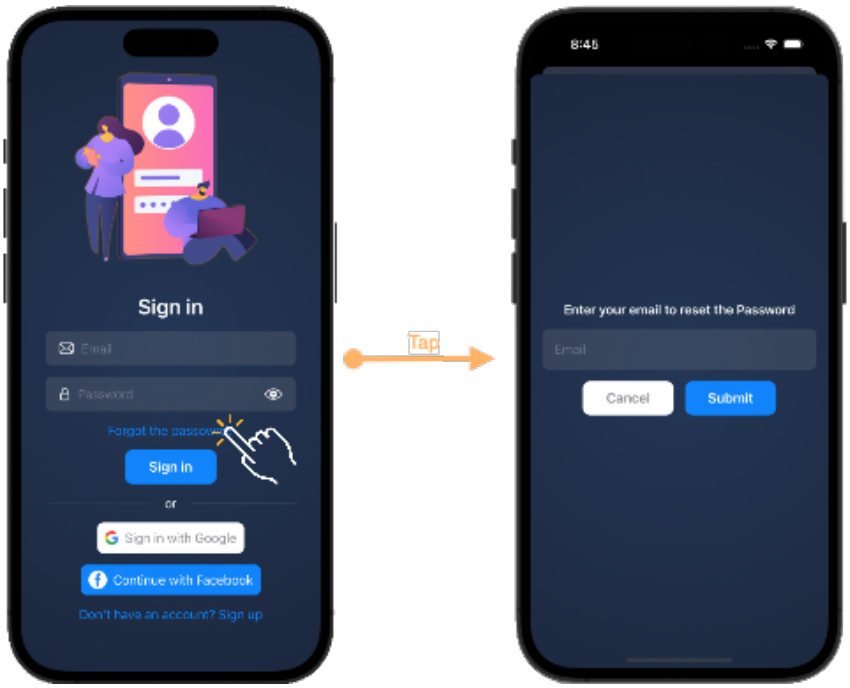


Figure 3.2: Reset Password

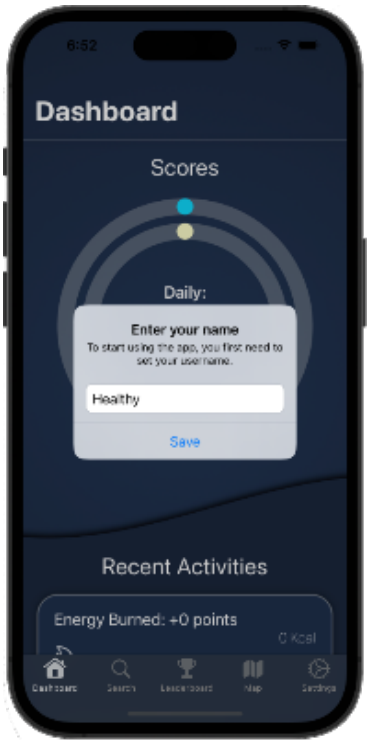


Figure 3.3: Username

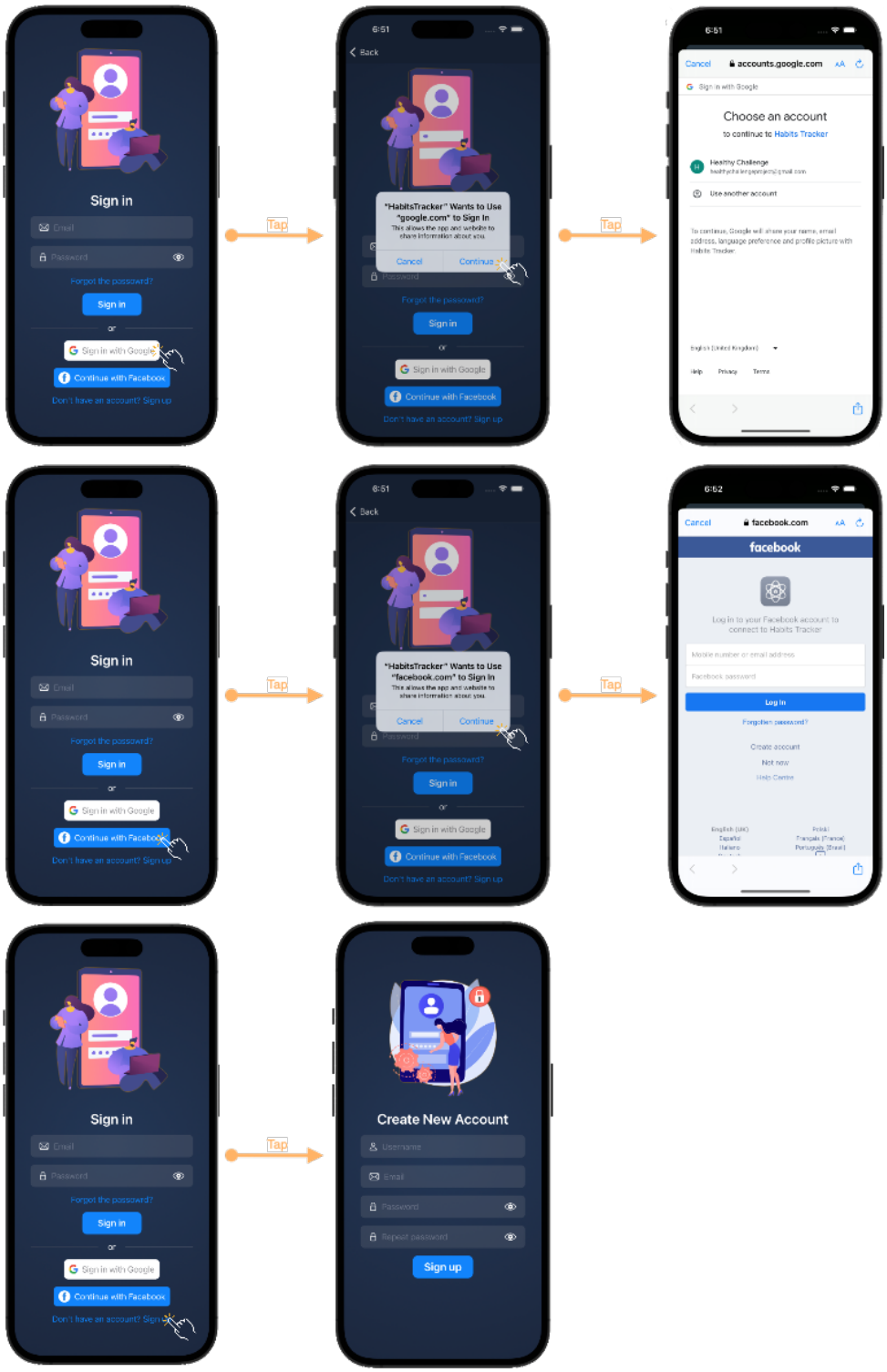


Figure 3.4: Sign In View

3.1.3. Dashboard View Description

The dashboard is the primary interface presented to users upon successful login. It is meticulously designed to offer a complete overview of the user's performance, segmented

into three main parts:

1. **Score Overview**:
    - Daily Score: Represents the accumulated score for the current day.
    - Weekly Score: Summarizes the score accrued over the week.
2. **Activity Breakdown**:
    - Detailed insights into each activity contributing to the daily score.
    - Allows users to understand which activities had the most impact.
3. **Activity Records**:
    - Showcases the record value for each activity.
    - Users can gauge their best performance in individual activities.

Figure 3.5: Dashboard View

In the top right corner, when a user receives a friend request, a small heart icon appears accompanied by the number of pending requests. Tapping on this heart will direct the user to the 'Requests' view, displaying a list of all friend requests. Here, users can choose to either accept or decline requests from other users.

Figure 3.6: Record

Figure 3.7: Request View

3.1.4. Search list View

The Search view displays a comprehensive list of all community members. By selecting a row from this list, users can access and view the corresponding member's profile.

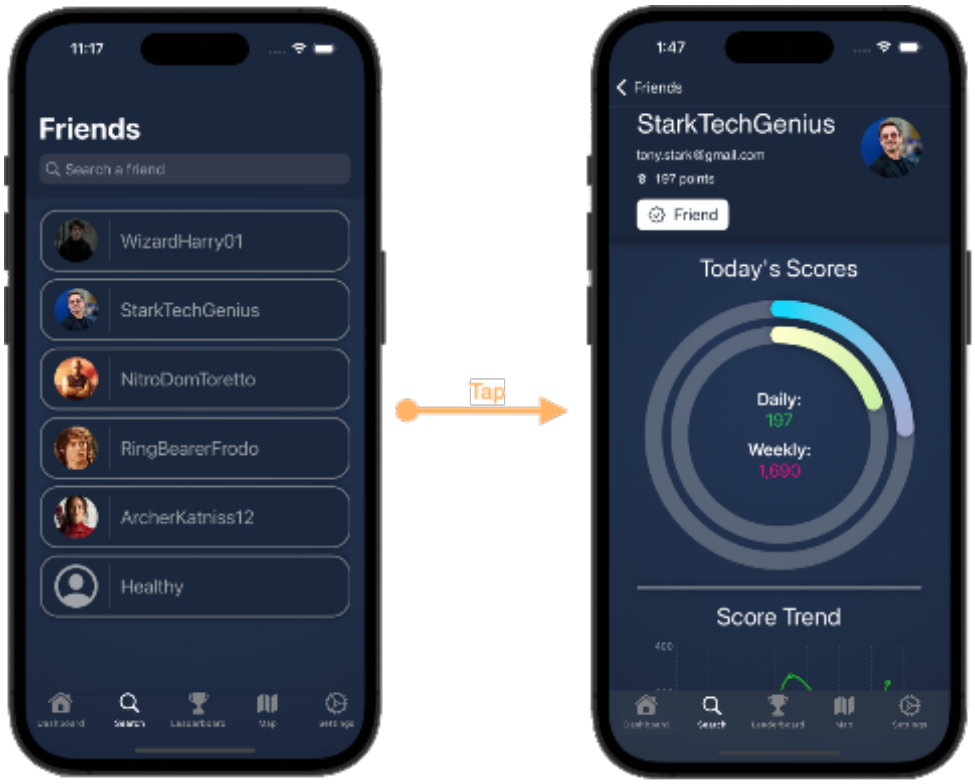


Figure 3.8: Search View

3.1.5. Leaderboard View

The "Leaderboard" view serves as a competitive tableau within the application, presenting users based on their performance metrics. Rendered as a methodically organized list, users are ranked according to their accumulated scores, which can be distinguished on either a daily or weekly timeframe. One of the salient features of this view is the capability for users to toggle between a global leaderboard and a more personalized leaderboard that exclusively displays scores of their friends. This dual-perspective enhances user engagement, offering both a broad overview and a more intimate comparison. As users navigate the leaderboard, each entry clearly delineates an individual's rank and score, promoting a transparent evaluation of personal achievements in relation to the larger community or a closer circle of friends.

Figure 3.9: Leaderboard

3.1.6. Point of Interests View

The "Point of Interests" view is a comprehensive geospatial interface integrated into the application. Upon initialization, and with the necessary user location permissions granted, this interface displays a map centered on the user's current location. Pre-determined points of interest, such as parks or gyms relevant to the user's preferences, are prominently marked with pins. Users can interact with these pins to access detailed address information. Additionally, an accompanying list presents all the marked landmarks, allowing users to select a specific landmark and view its precise location on the map. To enhance user engagement and utility, a search function is incorporated. This feature enables users to query and locate other points of interest, such as cafes or museums, expanding the range of discoverable locations in their vicinity.

3.1.7. User Profile

The 'User Profile View' functions similarly to the user's dashboard. It features sections for both daily and weekly scores, showcases the user's records, and includes a chart illustrating the daily score trends over the past week.

Figure 3.10: Points of Interests

Figure 3.11: Profile View

3.1.8. Settings View

This view provides users with several functionalities:

- Manage account credentials.
- Manage the user profile.

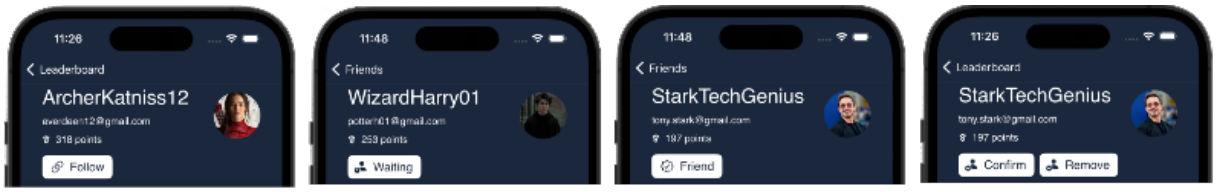


Figure 3.12: Button friend

- Link other service providers to the account. For instance, if a user is logged in with Google, they can also log in using Facebook.
- Enable and customize local notifications.
- Log out from the application.
- Delete the account.

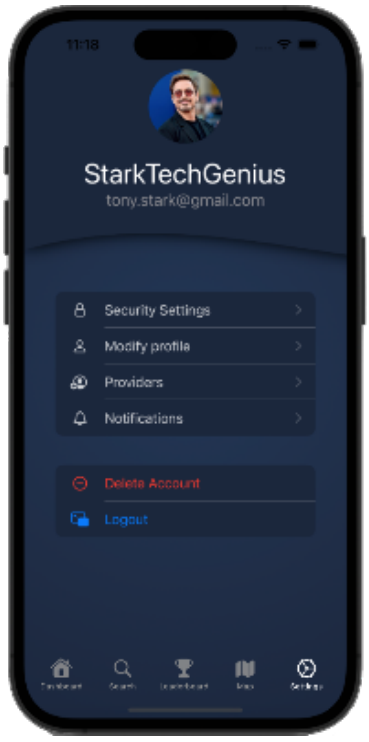


Figure 3.13: Settings View

Security Settings View

This view offers users the following security-related functionalities:

- Change the email and password, provided they are logged in with Firebase authentication.

- Sensitive operations prompt: if a user has a long login session, a pop-up will appear requesting the user to re-authenticate before proceeding with changes.

Figure 3.14: Security Settings View

Modify Profile View

This view grants users the ability to:

- Change their profile image.
- Update personal details such as weight, height, sex, and more.

3.1.9. Providers View

This view allows users to link additional providers to their current account.

3.1.10. Portrait and Landscape orientation

We've developed both portrait and landscape versions for all views in the app. However, post-implementation, we found that the landscape view was too restrictive, hindering optimal user experience. Our coordinator at Bending Spoons advised us to enforce a

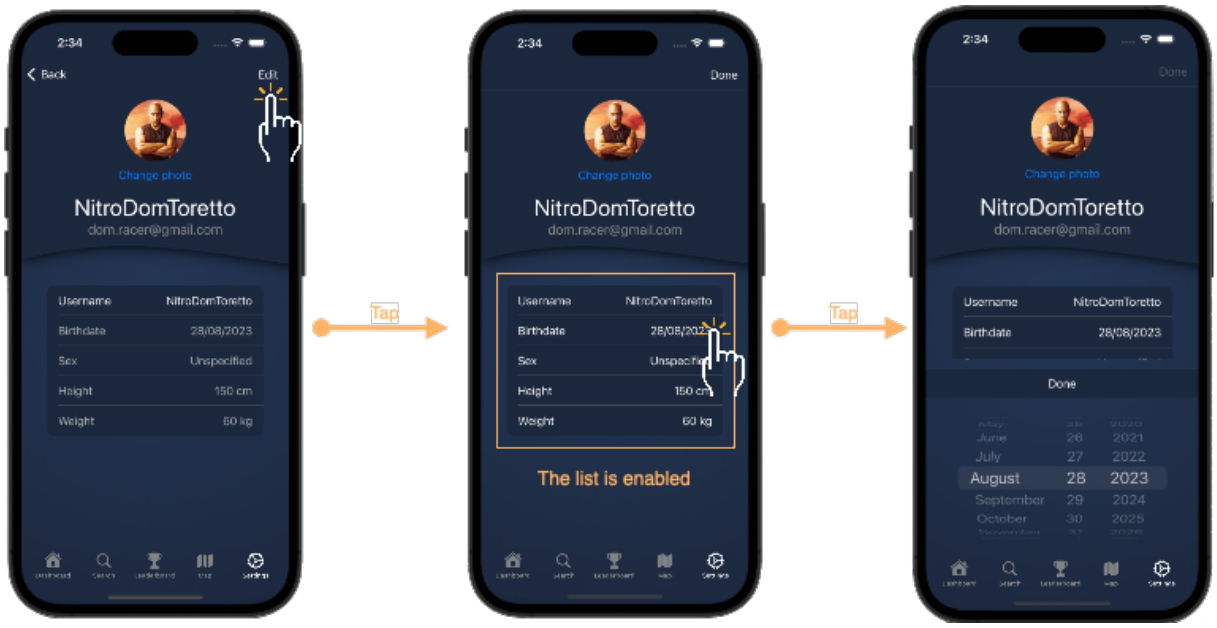


Figure 3.15: Modify Profile View

Figure 3.16: Providers View

portrait-only layout for enhanced app usability. For clarity, we've provided examples of some views to illustrate the rationale behind this decision.

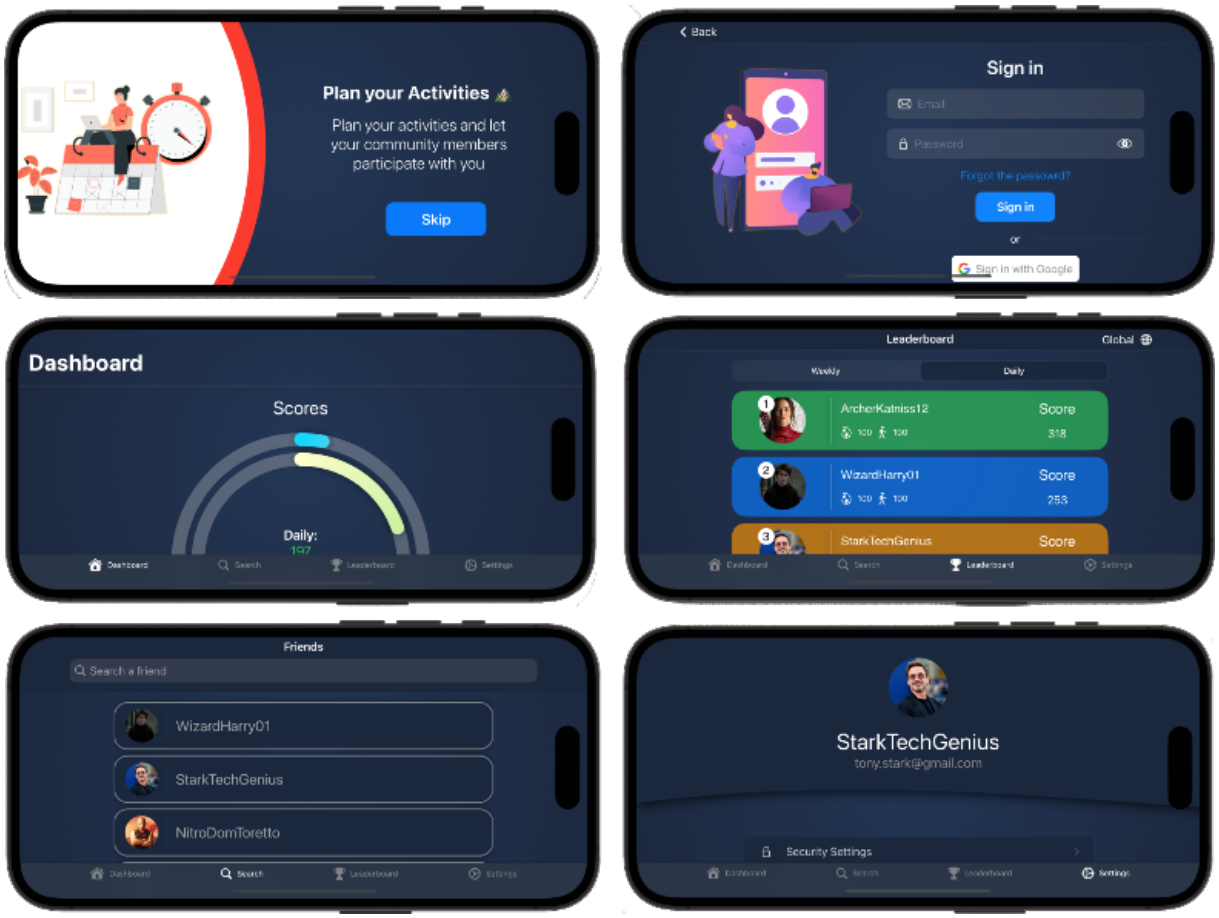
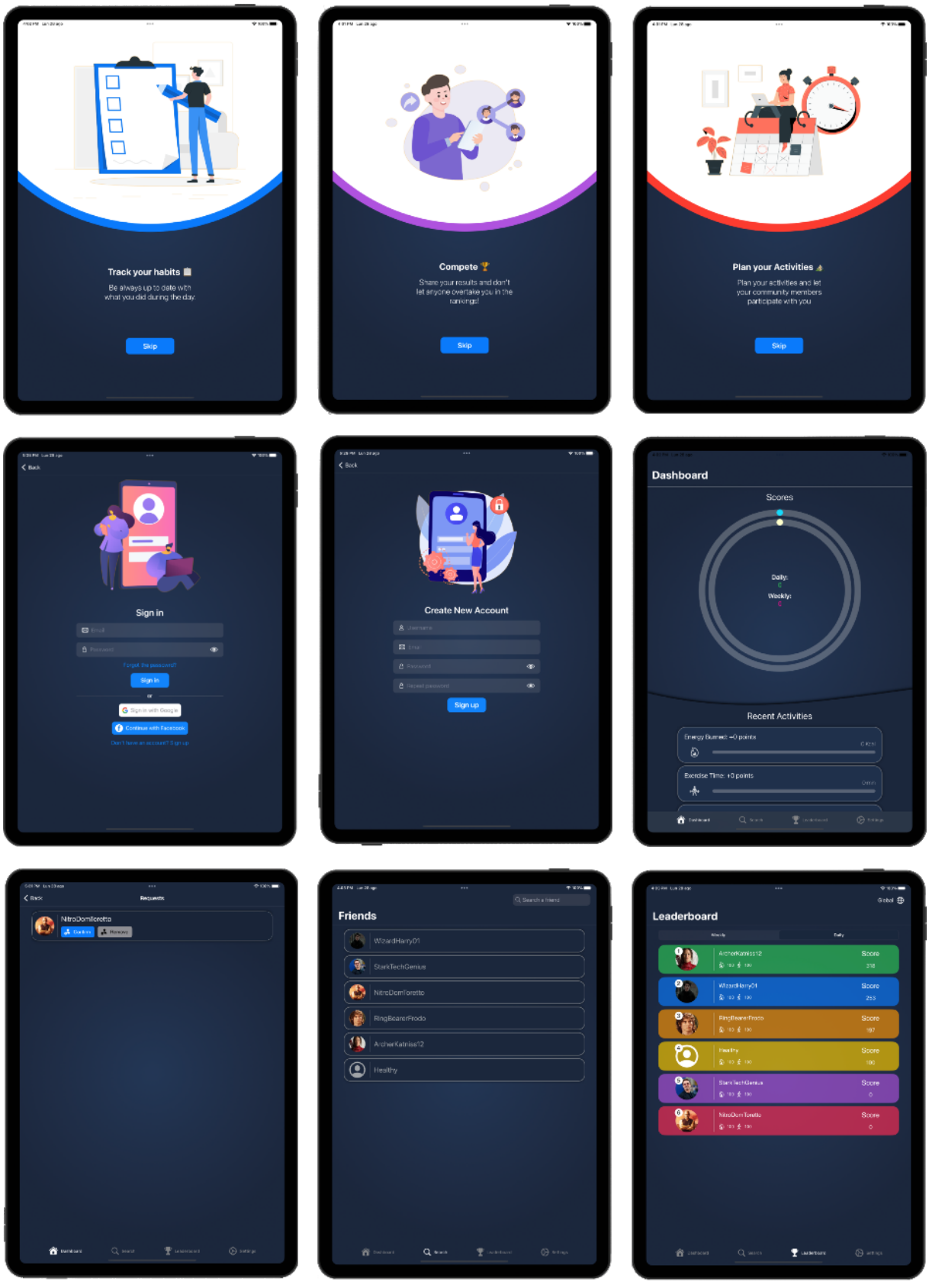


Figure 3.17: Landscape Views

3.2. iPad Layout

Below, we present the layout designs for the iPad version, showcasing both landscape and portrait orientations. The functionalities across all views remain consistent with those of the iPhone version.


Track your habits
Be always up to date with what you did during the day.
Skip
Compete
Share your results and don't let anyone overtake you in the rankings!
Skip
Plan your Activities
Plan your activities and let your community members participate with you
Skip
Sign in
Sign In
Create New Account
Sign up
Dashboard
Scores
Daily:
Weekly:
Recent Activities
Friends
Leaderboard

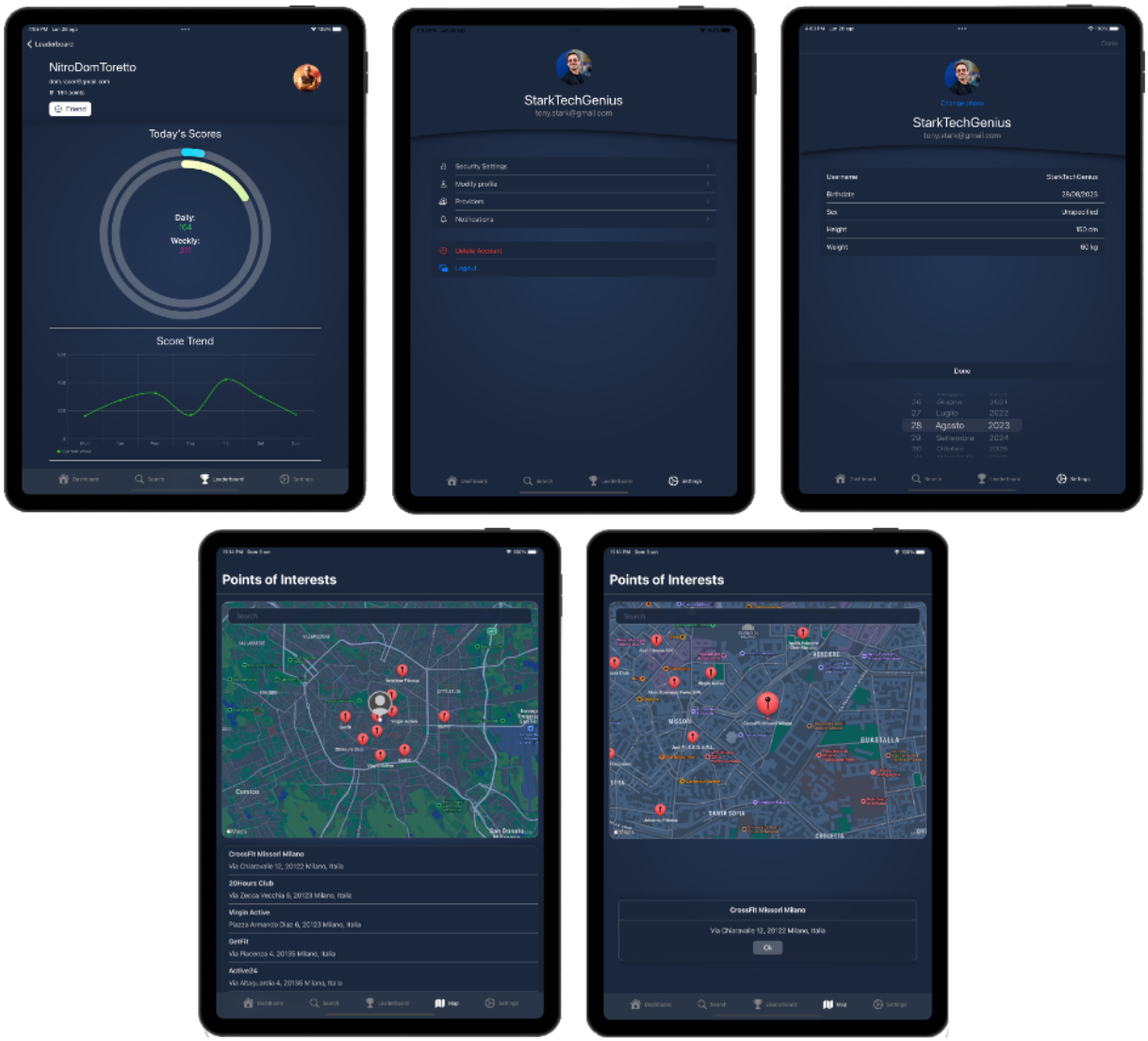
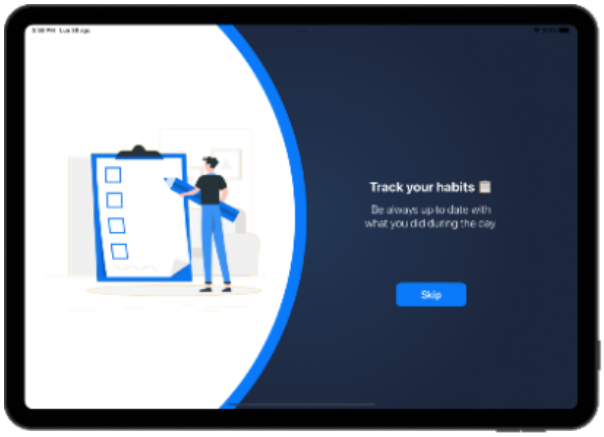
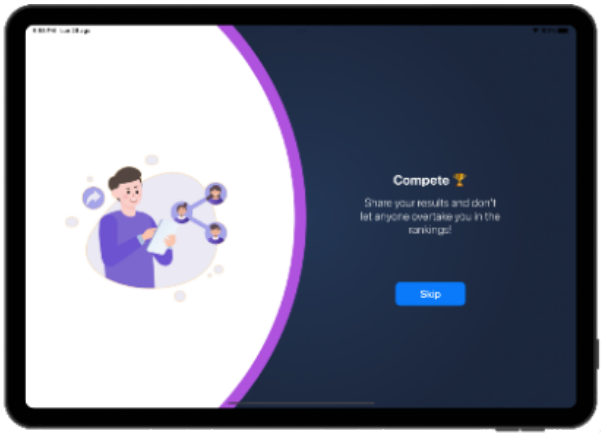
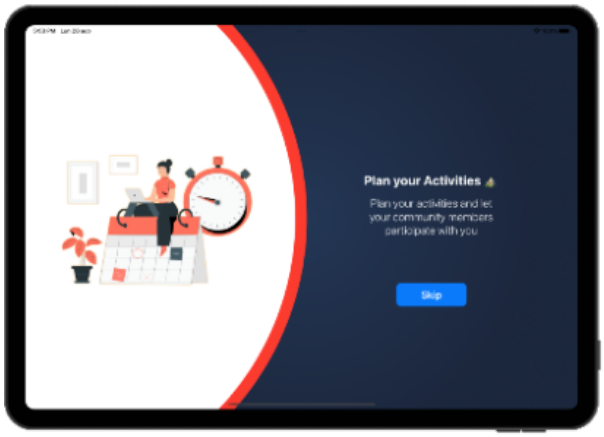


Figure 3.18: iPad Layouts


Track your habits
Be always up to date with what you did during the day
Skip


Compete
Share your results and don't let anyone overtake you in the rankings!
Skip


Plan your Activities
Plan your activities and let your community members participate with you
Skip


Sign in


Create New Account
Sign up


Dashboard
Scores
Daily:
Weekly:


Recent Activities

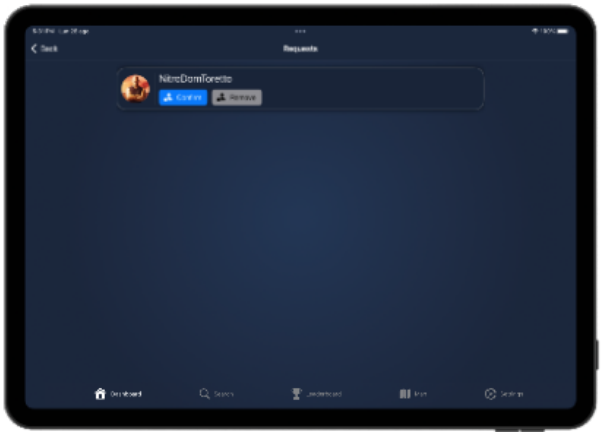
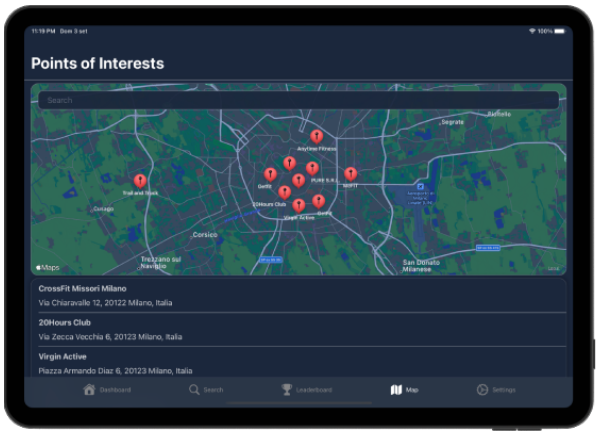
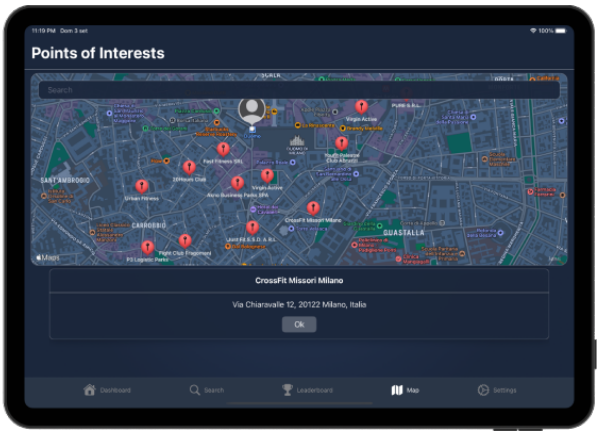

Requests

Figure 3.19: iPad Layouts


Points of Interests
Search
CrossFit Missori Milano
Via Chiaravalle 12, 20122 Milano, Italia
20Hours Club
Via Zecca Vecchia 6, 20123 Milano, Italia
Virgin Active
Piazza Armando Diaz 6, 20123 Milano, Italia
Dashboard
Search
Leaderboard
Map
Settings


Points of Interests
Search
CrossFit Missori Milano
Via Chiaravalle 12, 20122 Milano, Italia
Ok
Dashboard
Search
Leaderboard
Map
Settings

4 | Testing

4.1. Testing

Software testing is a crucial aspect of the development process, aiming to ensure the quality, reliability, and correctness of the software. In this section, we'll explore various aspects of testing, including unit testing, dependency injection, mocking, automatic testing, and UI testing.

4.1.1. Unit Testing

Unit testing is a testing technique where individual components or units of code are tested in isolation to ensure that they function as expected. These components can be functions, methods, or classes. Unit tests focus on verifying the behavior of a specific unit of code and help catch bugs and errors early in the development process.

Class Name	**Test Coverage Percentage**
HealthViewModel	95,6%
FirestoreViewModel	94,7%
AuthenticationViewModel	96,1%
PointsOfInterestsViewModel	84,7%
LeaderboardViewModel	100%
Settings ViewModel	91%

Dependency Injection

Dependency injection is a design pattern used to promote modularity and testability in software applications. It involves providing the dependencies of a component from the outside, rather than having the component create or manage its dependencies internally. This allows for easier testing by substituting real dependencies with mock objects or test doubles.

Mocking a Class

Mocking a class involves creating a mock object that mimics the behavior of a real class In scenarios where components interact with external services, such as databases or APIs, mocked classes are used to simulate the behavior of these services during testing. By doing so, the tests remain independent of the external service's availability enabling focused unit testing.

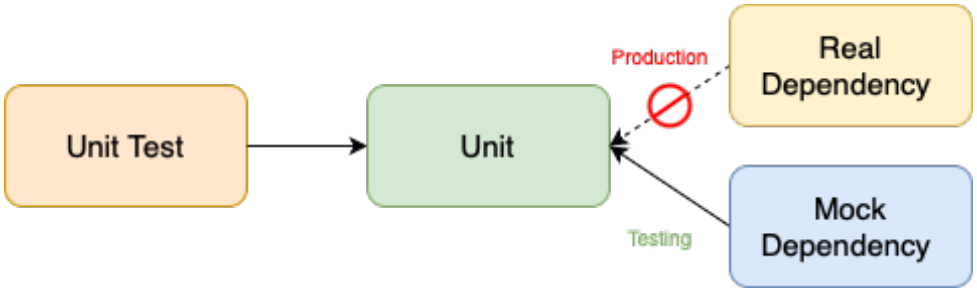


Figure 4.1: Testing with Mock and Dependency Injection

UI Testing

UI (User Interface) testing evaluates the interface through which users interact with an application. Its main goal is to ensure that the application not only looks aesthetically appealing but is also functional, intuitive, and user-friendly. Here are some detailed aspects of UI testing:

1. **Elements Interaction:** UI testing confirms that interface elements like buttons, text fields and others, respond as expected to user interactions. It ensures that elements are not only visible but also accessible and usable.
2. **Validation Feedback:** UI tests check that feedback mechanisms, such as error messages or confirmation pop-ups, are triggered correctly.
3. **Navigation Flow:** UI testing verifies that the navigational elements such as back buttons, function correctly, and guide the user through the intended paths without confusion.
4. **Automated vs. Manual:** While automated UI tests use tools and scripts to simulate user interactions and validate outcomes, manual UI testing involves human testers interacting with the application. Both methods have their advantages: automated tests are faster and can be easily repeated, while manual tests can capture nuances and subjective experiences a machine might miss.

To perform comprehensive UI testing, we have employed a combination of automated tools.

Name	Coverage	Executable Lines
CardIntro.swift	100,0%	1
HabitsTrackerApp.swift	100,0%	24
User.swift	100,0%	20
LottieView.swift	100,0%	21
LeaderboardViewModel.swift	100,0%	19
HealthRepository.swift	100,0%	16
FirestoreRepository.swift	100,0%	51
ActivityStatusView.swift	98,3%	296
AuthenticationViewModel.swift	96,1%	233
HealthViewModel.swift	95,5%	88
RecordDetailView.swift	95,3%	279
SettingsView.swift	95,1%	754
FirestoreViewModel.swift	94,7%	225
LoadingView.swift	94,0%	133
AuthenticationRepository.swift	93,9%	49
HomeView.swift	93,6%	419
SearchFriendView.swift	92,9%	184
ScoreRingView.swift	92,6%	229
LeaderboardView.swift	92,3%	531
CustomTextField.swift	92,2%	154
HealthDataSource.swift	91,9%	149
WaveView.swift	91,7%	145
SettingsViewModel.swift	91,0%	122
RequestListView.swift	90,2%	327
SignupView.swift	87,5%	361
ProfileImageView.swift	86,8%	76
NotificationDetailView.swift	86,3%	335
RecordView.swift	85,9%	99
ModifyProfileView.swift	84,9%	875
PointsOfInterestViewModel.swift	84,7%	59

Figure 4.2: Tests Coverage

5 | Future Developments

5.1. Proposals

The continuous evolution of the app is vital for meeting the demands of the users and leveraging advancements in health technology. The following list encapsulates some of our projected enhancements intended to further refine the user experience and cater to a broader spectrum of health needs:

- **Home Widgets**
  Description: Allows users to view crucial data, like their current score and leader board positions, directly from their home screens, ensuring that their health stats are always just a glance away.
- **Holistic Activity Tracking**
  Description: Beyond basic metrics like steps or distances, the app also lets users track activities like meditation and hydration, promoting a holistic approach to health.
- **Achievements and Badges**
  Description: To further gamify the experience, users can earn badges or achievements for reaching certain milestones or completing specific challenges. This provides added incentives and boosts motivation.
- **Community Challenges**
  Description: Users can participate in global or friend-based challenges, such as "10,000 steps a day for a week" or "Meditate for 10 minutes daily." Such challenges promote group activity and adherence.
- **Personalized Insights and Tips**
  Description: The app analyzes user data and offers tailored insights and advice. For instance, if a user consistently sleeps less than recommended, they might receive tips on improving sleep hygiene.

List of Figures