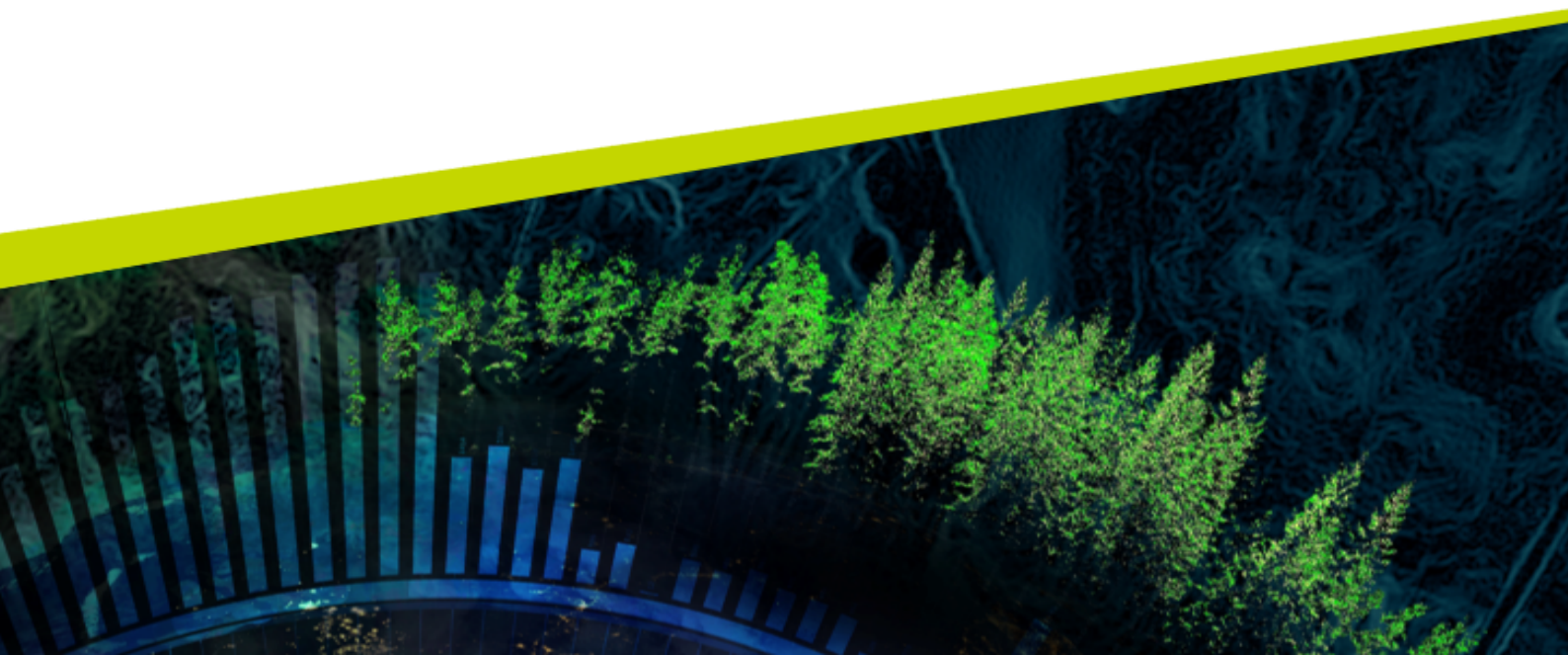
myAPTracker

Design Document

Mattia Siriani (10571322)
Matteo Visotto (10608623)

Computer Science and Engineering
Design and Implementation of Mobile Application course @ Politecnico di Miliano

v1.0 - 15/07/2022



Contents

1 Introduction

1.1 Purpose

This document is meant to provide a detailed explanation about the technical details about the implementation of myAPTracker application. In particular, the implementation will be presented alongside the interfaces that will compose the application (and the needed back-end).
Moreover, the functionalities offered by the application will be shown through the run-time view, highlighting the interaction and the view hierarchy available to the user.

1.2 Scope

The scope of the application is to create an app that let users to trace the ongoing price of Amazon products. In the app for each product the price peaks, the current price and a graph of how the price changed over time will be visualized. The user can browse the application without register or login but in order to track a product and to receive personalized notification an account is required. In the application it's also present an explore section in which the user can see the most tracked products or the products with the biggest current price drop. A logged user can choose for each tracked product different options to be notified on a price change.

1.3 Definition, acronyms, abbreviations

1.3.1 Definition

- **RESTful:** it's a software architectural style that defines a set of constraints to be used for creating Web services.
- **Tier:** In general, a tier is a row or layer in a series of similarly arranged objects. In computer programming, the parts of a program can be distributed among several tiers, each located in a different computer in a network.
- **OAuth:** It's an open standard for access delegation, commonly used as a way for internet users to grant websites or applications access to their information on other websites but without giving them the passwords.
- **Back-end:** Server side part of the system that provides API for an application.
- **Scraper:** It is a type of software used to copy content from a website.
- **Command-line tools:** They are scripts, programs, and libraries that have been created with a unique purpose, typically to solve a problem that the creator of that particular tool had himself.

- **Package:** A package consists of Swift source files and a manifest file. The manifest file, called Package.swift, defines the package's name and its contents using the PackageDescription module. A package has one or more targets. Each target specifies a product and may declare one or more dependencies.
- **Product:** A product is something that is purchasable from Amazon.
- **Paging:** It refers to dividing the request done to the API, by requesting a limited amount of data per time.

1.3.2 Acronyms

- **API:** Application Programming Interface.
- **HTTPS:** Hyper Text Transfer Protocol over SSL.
- **DD:** Design Document.
- **ER:** Entity-Relationship.
- **TLS:** Transport Layer Security.
- **SSL:** Secure Socket Layer.
- **DBMS:** DataBase Management System.
- **IdP:** Identity Provider.
- **OAuth:** Open Authorization.
- **UI:** User Interface.
- **FCM:** Firebase Cloud Messaging
- **APN:** Apple Push Notification

1.4 Reference documents

- Project specification by BendingSpoons.
- Apple developer documentation https://developer.apple.com/documentation/.
- Swift documentation https://www.swift.org/documentation/.
- Slide of Design and implementation of mobile application.
- myAPTracker API documentation https://aptracker.matmacsystem.it/docs/

1.5 Packages used - Application

- SwiftUILoadingIndicators https://github.com/SwiftfulThinking/SwiftfulLoadingIndicators.
- FacebookLogin https://github.com/facebook/facebook-ios-sdk.
- GoogleSignIn https://github.com/google/GoogleSignIn-iOS.
- FirebaseMessaging https://github.com/firebase/firebase-ios-sdk.
- FirebaseAnalytics https://github.com/firebase/firebase-ios-sdk.

1.6 Packages used - Scraper

- SwiftSoup https://github.com/scinfu/SwiftSoup.
- SwiftArgumentParser https://github.com/apple/swift-argument-parser.

1.7 Document structure

- **Section 1: Introduction**
  This section offers a brief description of the specification and required functionalities, also providing definition and acronyms that can be found in this document.
  It also provides the main structure of the document itself.
- **Section 2: Architectural Design**
  This section is addressed to the developer offering a detailed description of the architecture and its components. The first part describes the chosen paradigm and the division of the system in its layers. Then a better description of the application is given including the general flow for each main function that the app provides.
- **Section 3: User Interface Design**
  This section contains several mockups of the user interfaces and refers to the client side experience. Mockups are provided by means of diagrams in order to describe the general application flow.
- **Section 5: Implementation, Integration and Test Plan**
  The last section describes the procedures for the implementation phase followed by testing and integration. It provides a detailed description of the core functionalities with a complete report about how to implement and test them.

2 Architectural Design

2.1 Overview

In Figure 1 it's present the black-box view of our architecture, that is defined by three different layers represented:

- **Presentation Layer:** it's the layer that is used to present data from the application layer in an accurate, well-defined and standardized format; in this case it is represented by the mobile application.
- **Application Layer:** it's the layer that manages all the functions that controls the business logic of our system (back-end).
- **Data Layer:** it controls how the data are stored and accessed (DB).

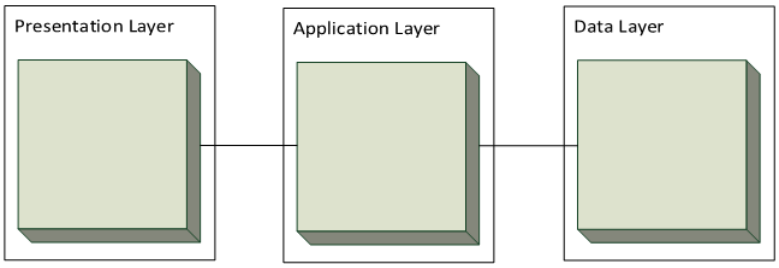


Figure 1: Three layers application

Generally, the architecture divide the application in three different layers: the user can interface with the mobile application which is connected to the application layer by means of RESTful API. The Application layer represents the back-end with the application logic and the Data layer is the the DBMS that provides (by DBMS APIs) the function to retrieve and store the data by the application server.

2.2 Deployment View

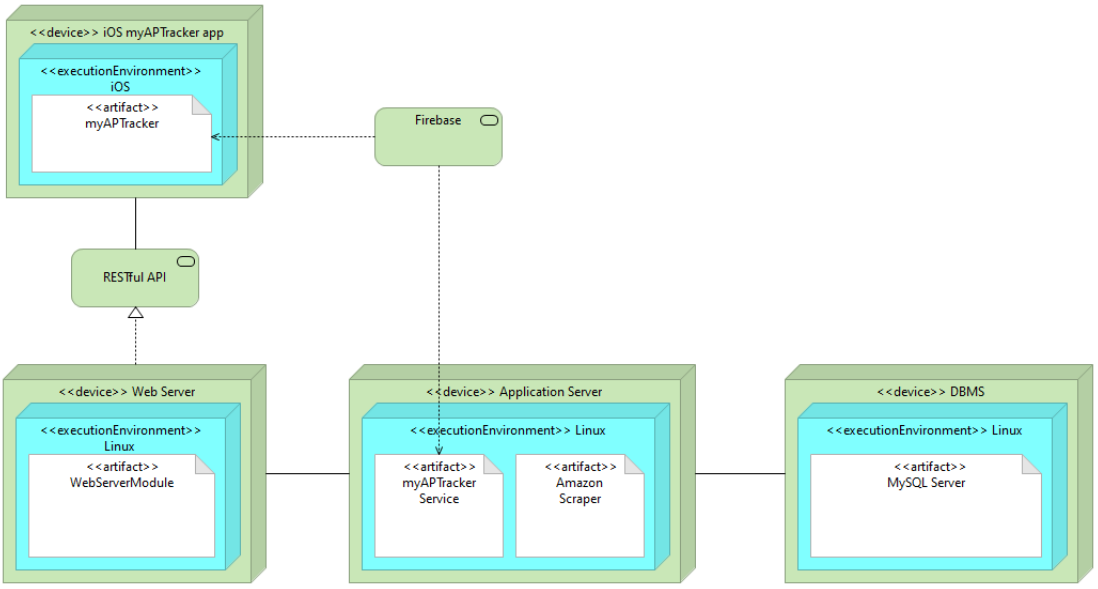


Figure 2: Deployment Diagram

The deployment diagram in figure 2 shows the topology of the system's hardware and specify the distribution of components. For each device its Operating System is indicated.

- **Tier 1**: It is the client machine. It is an iPhone or and iPad where the application is installed.
- **Tier 2**: It consists in the web server that provides the RESTful APIs managing HTTP requests and forwarding them to the application server.
- **Tier 3**: it consists of the application servers. They provides all the business logic, allowing to manage requests and it is connect to the data tier using the DBMS gateway.
- **Tier 4**: it consists of the database management system servers. The data are stored in this device and it provide to the application server tier the operations needed to manage the data.

2.3 Runtime View

2.3.1 Local registration

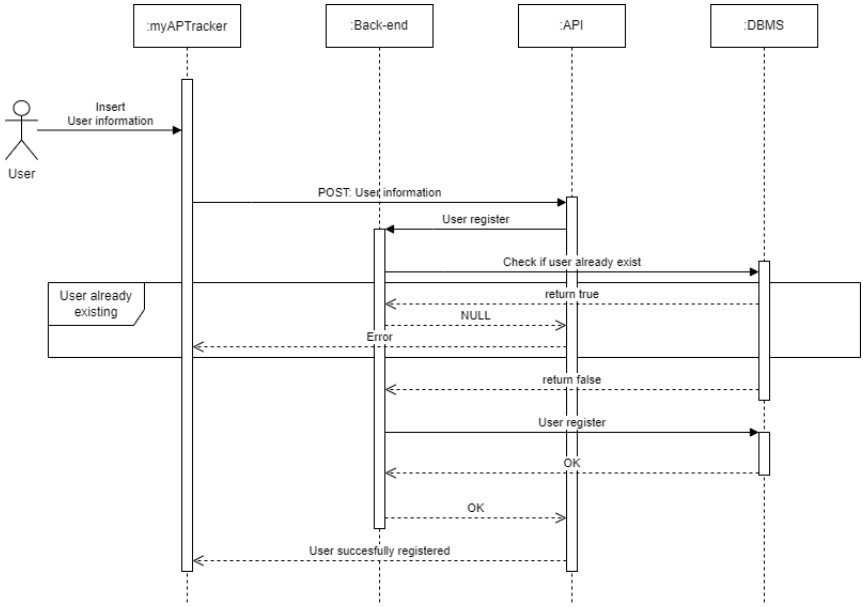


Figure 3: Local registration sequence diagram

This is the sequence diagram of a local registration done by a user. When registering, if the credentials inserted (email) are already stored in the DB an error will occur and it is returned to the application, otherwise the user data and the credential will be stored in the database and a success response is returned.

2.3.2 Local login

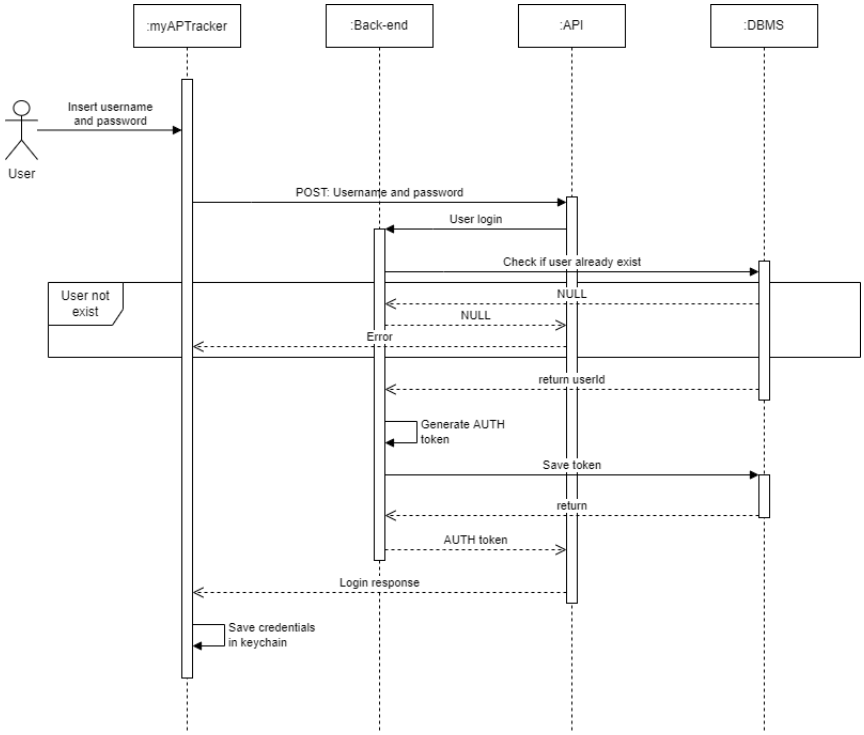


Figure 4: Local login sequence diagram

This is the sequence diagram of a local login done by a user. Before the login the user must be registered to the myAPTracker or an error will occur. As a positive response of the login flow, an auth token and a refresh token are returned by the server. The refresh token is then used in order to require a new auth token when expired (the refresh flow is omitted since it is equal to the login one but the refresh token is sent instead of credential).

2.3.3 Social login - Google

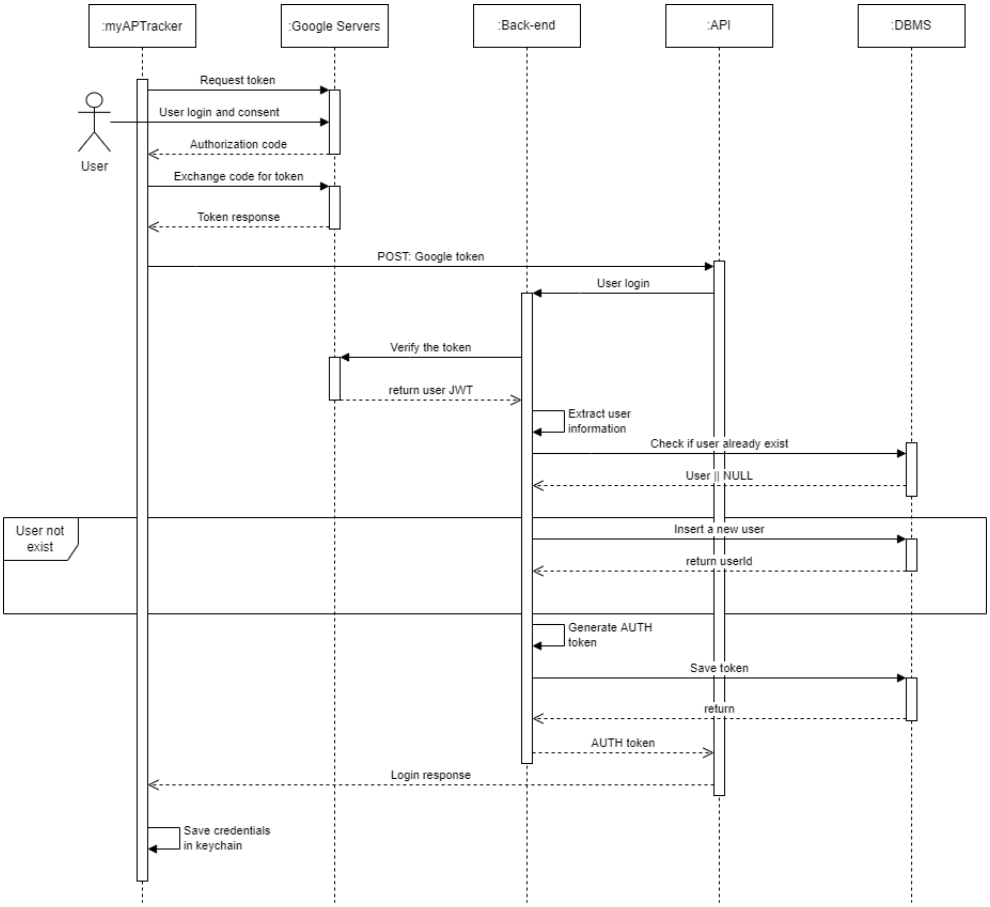


Figure 5: Google Social login sequence diagram

This is the sequence diagram of a login from a user that use Google as external service for the social authentication.
The flow can be considered the union of the registration and login ones. Once the application send the google token received by the social authentication, it is validated and then used to retrieve user data. If the user email is not present in the database the new user in inserted using the register function. Finally, in both cases, the auth token and the refresh token are returned to the application.

2.3.4 Social login - Facebook

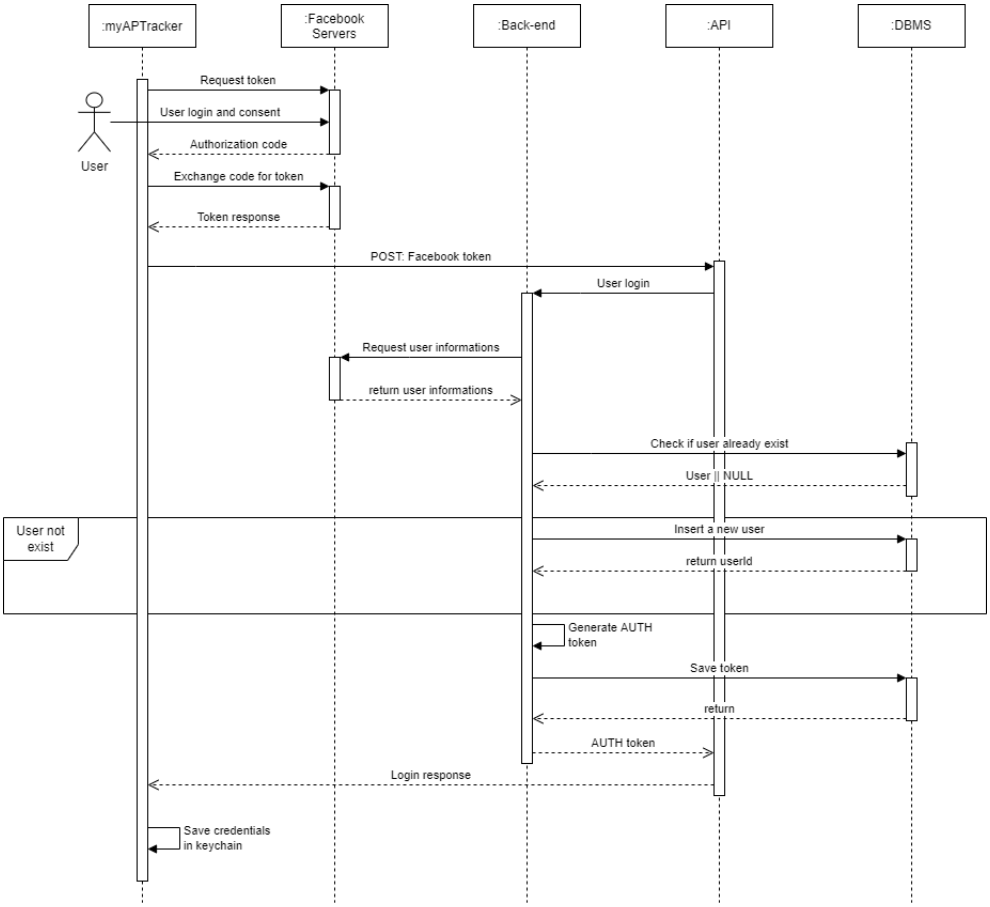


Figure 6: Facebook Social login sequence diagram

This is the sequence diagram of a login from a user that use Facebook as external service for the social authentication.
The flow is equal to Google's one.

2.3.5 Social login - Apple

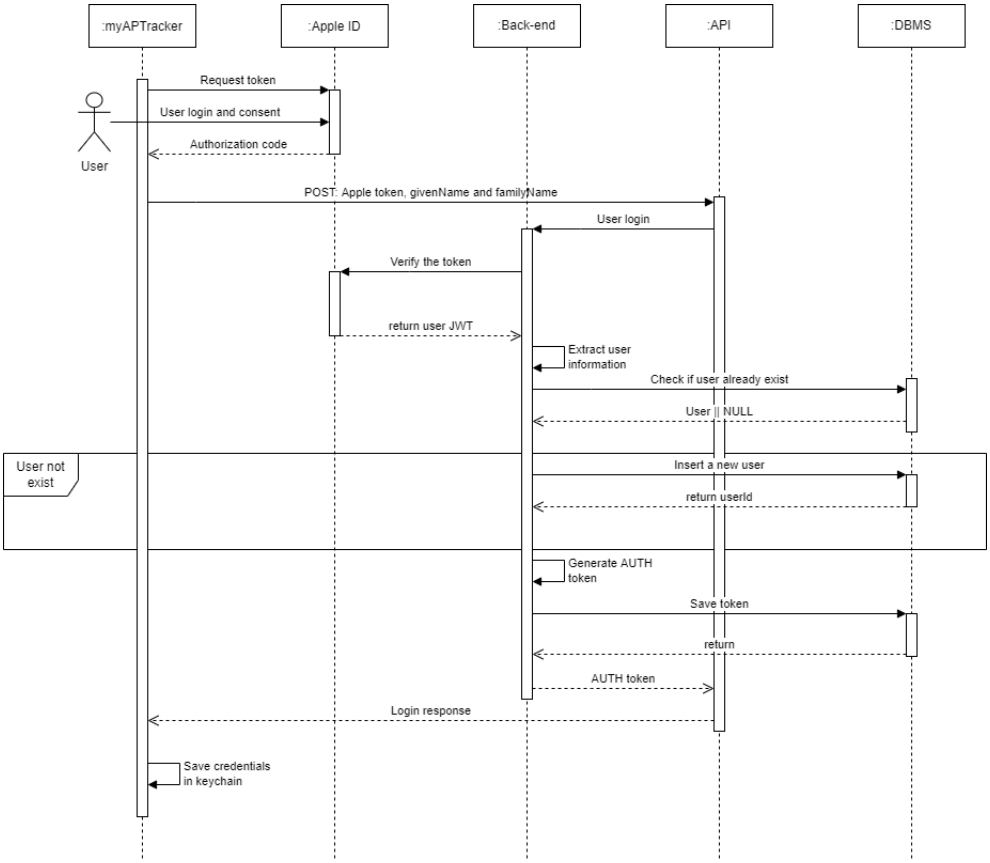


Figure 7: Apple Social login sequence diagram

This is the sequence diagram of a login from a user that use Apple as external service for the social authentication.
Very similar to the other social flows, Apple Sign-In requires also to send the user data (name and surname) since these data can be dynamically set by the user at the fist sign-in and they cannot be retrieved in a further moment.

2.3.6 Track product

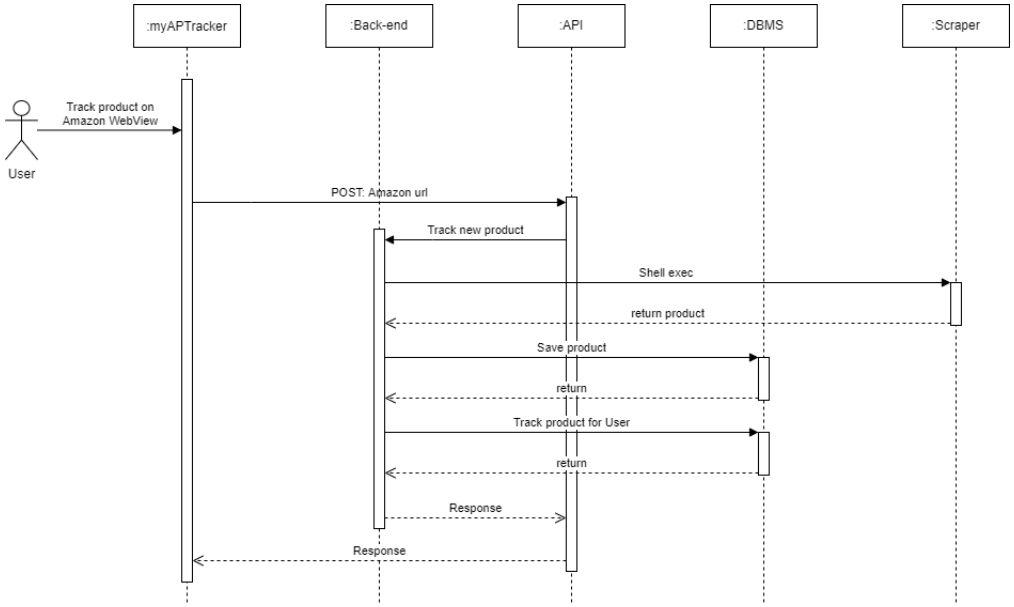


Figure 8: Track product sequence diagram

This is the sequence diagram when a user try to track a product. Once the application post the amazon URL to the endpoint, the server function calls the Scraper by shell execution it retrieves all the relevant information about the given product, storing them later on the DB. Finally, the product is marked as tracked in the database for the user who posts the URL.

2.3.7 Generic API

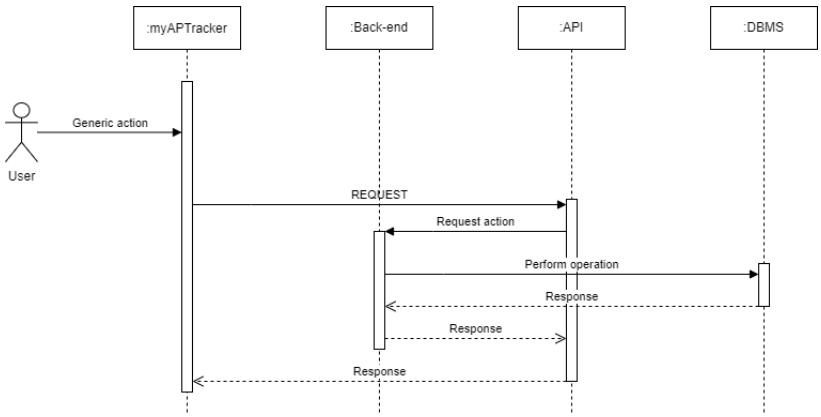


Figure 9: Generic API sequence diagram

This is the sequence diagram when a user do a generic action that involves the call of an API. All the functionalities that require to communicate with the database make a request to the specific endpoint which calls the correspondent server function. The server communicate with the database to get/save/update the data and it returns a response to the application.

2.3.8 Update price and notify users

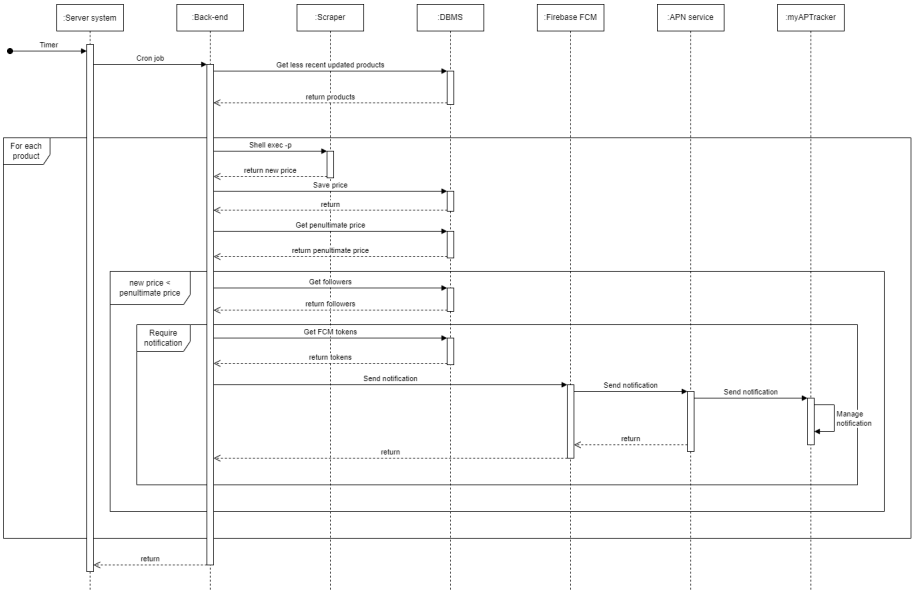


Figure 10: Update price and notify users sequence diagram

This is the sequence diagram when a user receive a notification for a product drop. This flow is not invoked by a user action but, instead, by a cron-job scheduled in the server OS.
The server load all the products that need to be updated and for each of them calls (by shell execution) the Scraper (using –price-only flag) which returns the updated price the server now store in the database. Then the server loads the penultimate price (since the new one is already saved) and if the new is lower then the previous one, all the user tracking it are taken.
Now each user could have a different notification setting so, for each one, the condition is checked and, if satisfied, the tokens of user devices are sent to Firebase Cloud Messaging (FCM) endpoint.
Finally, FCM is configured to forward notification to APN service which is responsible to send push notification to devices.

2.4 Logical Description of Data

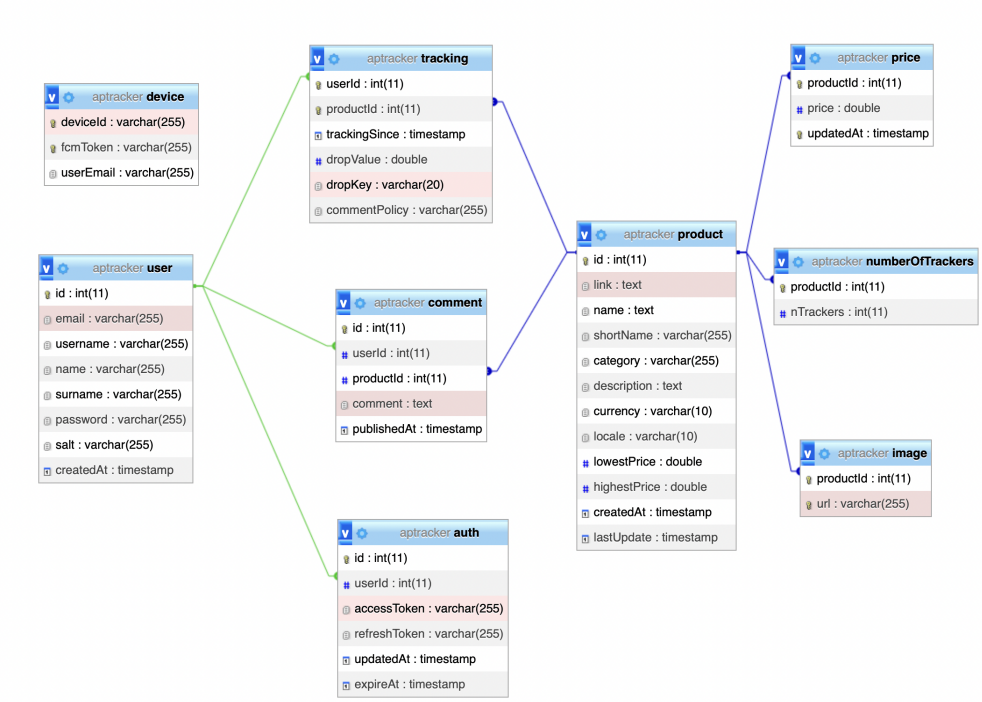


Figure 11: Back-end database ER diagram

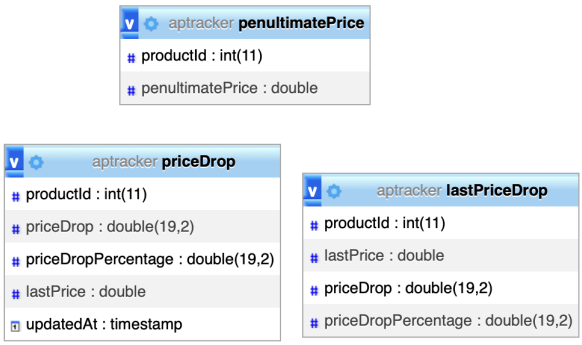


Figure 12: Useful Views in the DB

The two figures above show the database structure used for our back-end and some useful views that we have used for retrieve some set of prices.

The database (fig: 11) contains all the relevant information regarding the user and the product stored. The *user* table stores all the information created for a user when he registers while his authentication credentials (auth token, refresh token and expiration date) are stored in the *auth* table.
The *product* table contains the product information retrieved from the scraper with the *image* table used to store the multiple images related to a product and the *price* table for the product price history. The *comment* table stores the comments a user post in a product page while the *tracking* table keeps in relation a user with the tracked products. A *numberOfTrackers* table is used to keep easily track of the number of trackers for each product and it is kept updated by triggers.
Finally, the *device* table is responsible to keep the association of the device token with its FCM token and eventually, if a user is correctly logged in, the record is associated also with the user email.

The Views (fig: 12) are used to get relevant data faster, in particular to get products in different orders. The most relevant API that require this views are:

- **GetByLastPriceDrop:** get the products ordered by the drop between the last two prices ordered by absolute drop.
- **GetByLastPriceDropPercentage:** get the products by the drop between the last two prices ordered by drop percentage.
- **GetByPriceDrop:** get the products by the drop between the highest and lowest price ordered by absolute drop.
- **GetByPriceDropPercentage:** get the products by the drop between the highest and lowest price ordered by drop percentage.
- **GetMostTracked:** get the product ordered by most tracked product.

For all those API also a correspondent paging function is present to allows the user to load less products and divide them in pages.

2.5 Architectural Style and Patterns

2.5.1 Four-tiered architecture

The usage of a four-tier architecture allows having four layers with completely different goals:

- Presentation Layer (PL);
- Data Presentation Layer (DPL);
- Business Logic Layer (BLL);
- Data Access Layer (DAL).

Each tier is responsible for a specific layer: this implies that the logic is separated and changes at a certain tier, doing like that will not affect the other layers. This improves the maintainability of code and it is easier to implement new features.
This architectural style allows to improve security because users can not have direct access to database structure.
Finally, this division improve also the simplicity to add new feature and perform updates on the logic without changing the presentation layer.

2.6 Model View View-Model (MVVM)

Model-View-ViewModel (MVVM) is a software design pattern that is structured to separate program logic and user interface controls.
The separation of the code in MVVM is divided into View, ViewModel and Model:

- **Model:** provides the logic for the program, which is retrieved by the View-Model upon its own receipt of input from the user through the View.
- **View:** is the collection of visible elements, directly interactable by the user, which also receives user input. This includes user interfaces (UI), animations and text.
- **ViewModel:** is located between the View and Model layers. This is where the controls for interacting with the View are present and contains also the controls for modify the data present in the Model.

This pattern has been applied to the application.

2.6.1 Model View Controller (MVC)

Model-View-Controller is a software design pattern used for developing User interfaces that divides the related program logic into three interconnected elements. This is done to separate internal representations of information, from the ways information is presented to and accepted from the user.
These three components are:

- **Model:** the central component of the pattern. It is the application's dynamic data structure, independent of the user interface. It directly manages the data, logic and rules of the application.
- **View:** any representation of information such as the JSON output of the API interface.
- **Controller:** accepts input from the client and converts it to commands for the model that generally lead to state changes in the view.

This pattern has been applied to the back-end.

2.7 Other Design Decision

2.7.1 Easy usability

Since our target is a customer of every age, the application is designed to be very simple and intuitive, so particular attention must be put on the organization on the interfaces following official Apple guideline taking care also to provide a device-dependent user interface.

2.7.2 Security

The application has an encrypted communication with the backend going on a secure channel using SSL protocol over HTTP (obtaining HTTPS) and also the application is integrated with social authentication (Apple, Facebook and Google) relying on OAuth2.0 protocol.

3 User Interface Design

In this section is present the design of all the screen of our application (for iPhone and iPad), the view used (with also the packages used), the widget view used and the most relevant flows of the functionalities that a user can exploit in myAPTracker. All the iPhone screen are taken from an iPhone 12, while the iPad screen are taken from an iPad Air.

3.1 Application flows

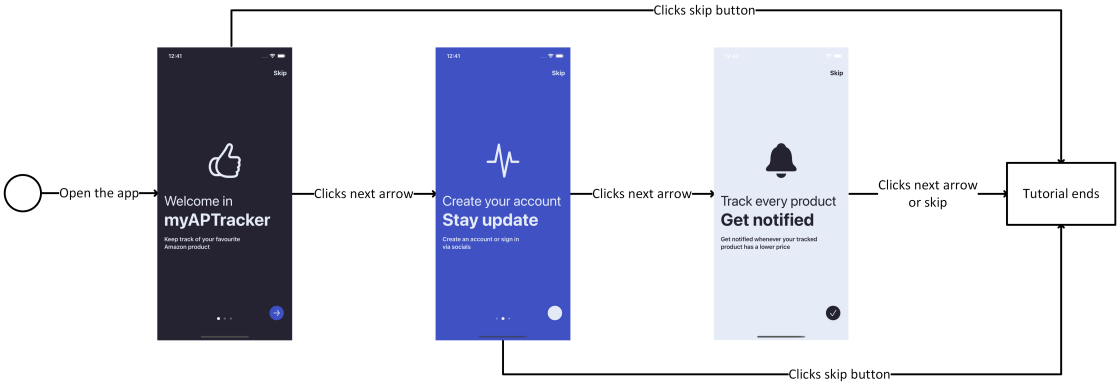


Figure 13: User do the tutorial

In this flow (fig: 13) the user has just downloaded the application, so the first thing that the app will shown at the first opening is a tutorial of how to use the application. The user can decide to view the complete tutorial or skip it.

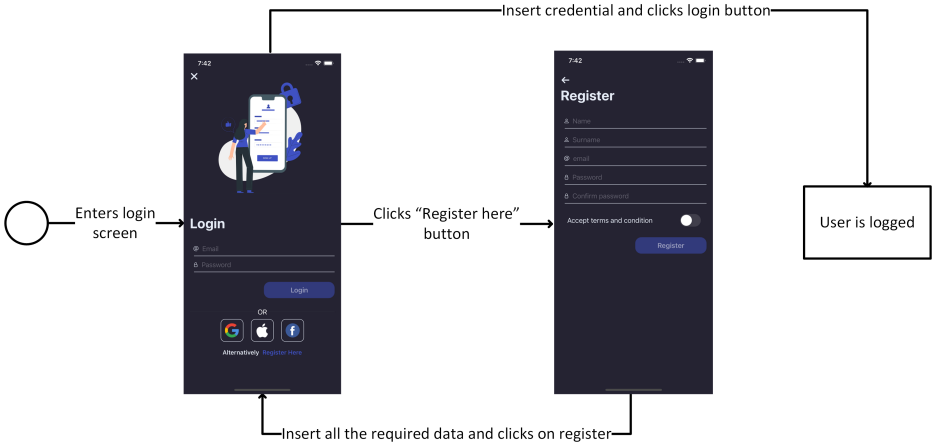


Figure 14: User register and login locally

In this flow (fig: 14) the user decide to register to the application, by clicking

the register button below the social logos and filling all the requested data. Then he can logs in by inserting the credential that has already listed in the registration phase.
If the user has already an account can just insert his credentials and logs in.

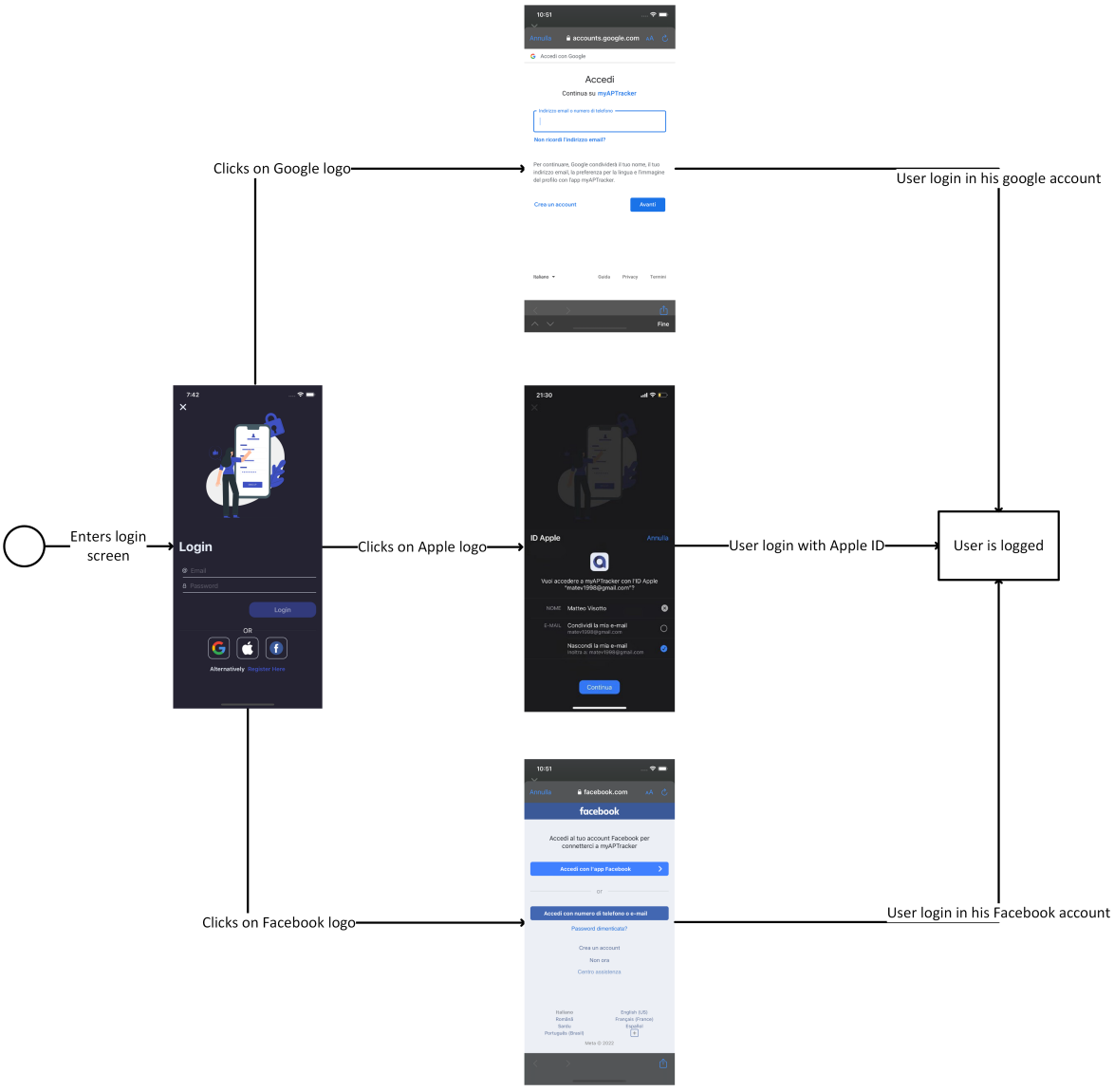


Figure 15: User register and login with social

In this flow (fig: 15) the user decide to authenticate through a social authentication mechanism. In order to do that, the user decide the social from which he wants to be authenticated and do the login process in the social. After that procedure the user is authenticated in the application.

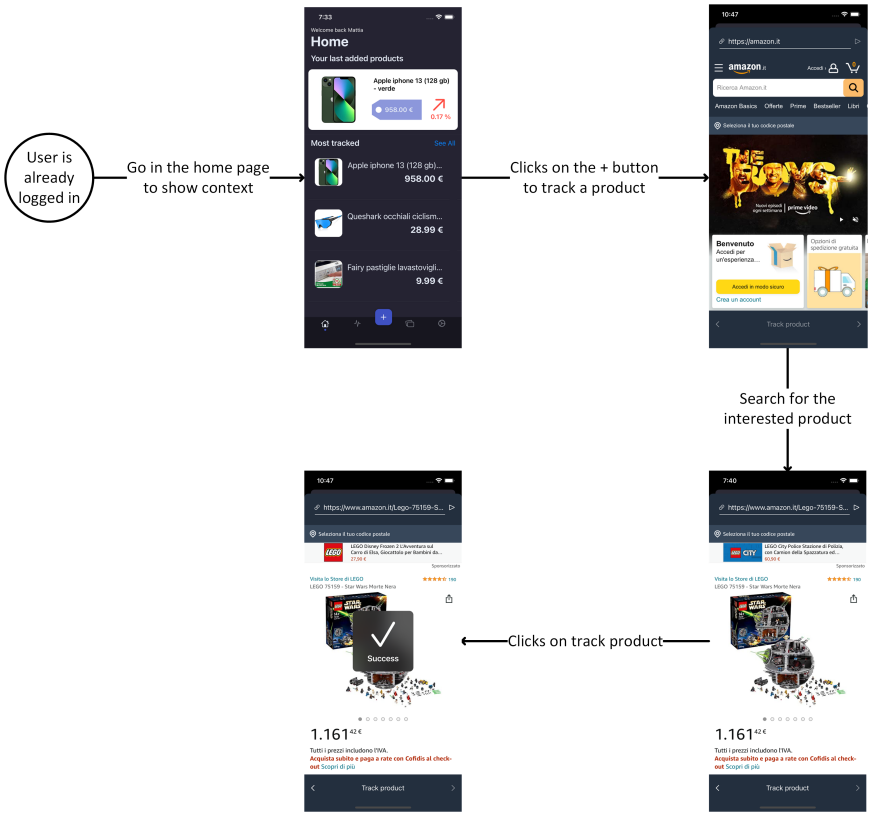


Figure 16: User add/track a product

In this flow (fig: 16) the user decide to track a new product. In order to do that, the user clicks on the plus button present in the bottom tab bar and a sheet will open containing a WebView of the Amazon Homepage (in the iPad the position of the button is different how we will see later). Then a user can search for the product he is interested in and, once reached the product page, he can track it by clicking the "track" button on the bottom.

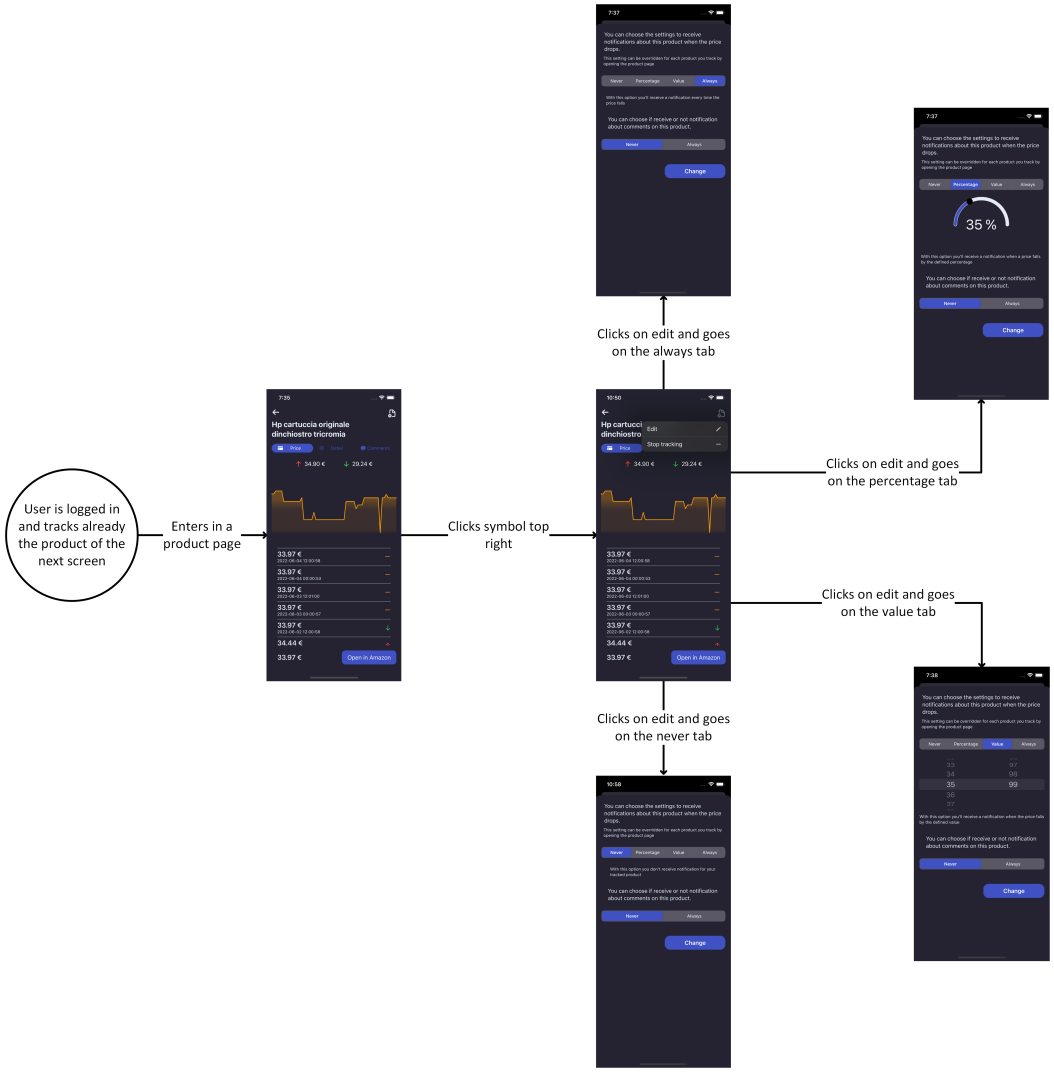


Figure 17: User set a range to a product in order to receive a notification

In this flow (fig: 17) the user decide to modify the rule under which he receives a notification for a specific product. In order to do that, the user enters in the product page, clicks the symbol in the top right and then choose to edit the product settings. At this point the can choose 4 different types of possible notification:

- **Always:** every time the price change the user will receive a notification for this product.
- **Never:** the user will never receive a notification for this product.
- **Percentage:** every time the price goes below a certain percentage (from the price present when the notification setting was saved), the user will receive a notification for this product.

- **Value:** every time the price goes below a certain price, the user will receive a notification for this product.

Finally, the user can also choose to receive or not the notification on comments about that product and save the settings by clicking on change.

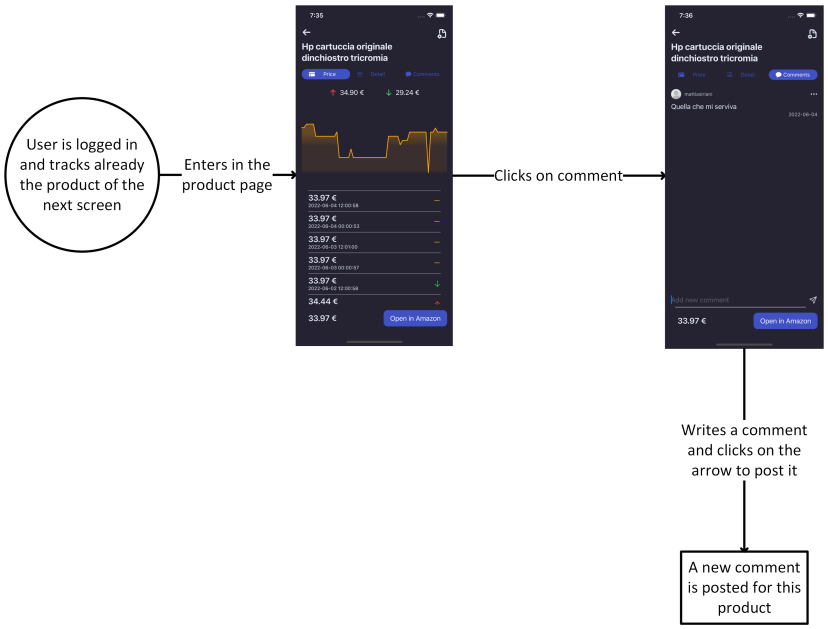


Figure 18: User post a comment for a product

In this flow (fig: 18) the user wants to post a comment about a specific product. In order to do that, the user enters in the product page and goes on the comment section. After done that, the user writes what he wants to say about this specific product and then click the arrow on the right to post it. The page will redraw and the user comment will be present in the page.

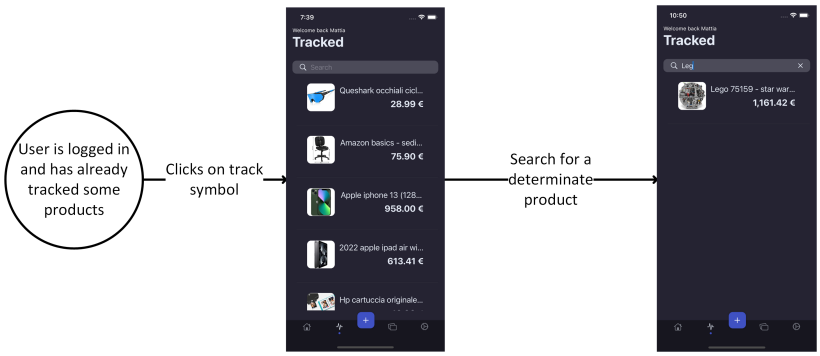


Figure 19: User search a product between all the products that he has tracked

In this flow (fig: 19) the user wants to search a product that he has already

tracked, in order to see it. To do that, the user goes in the tracking section, present in the bottom navigation bar and at this point he will type the name of the product he is interested in and all the products that correspond to the search will be shown.

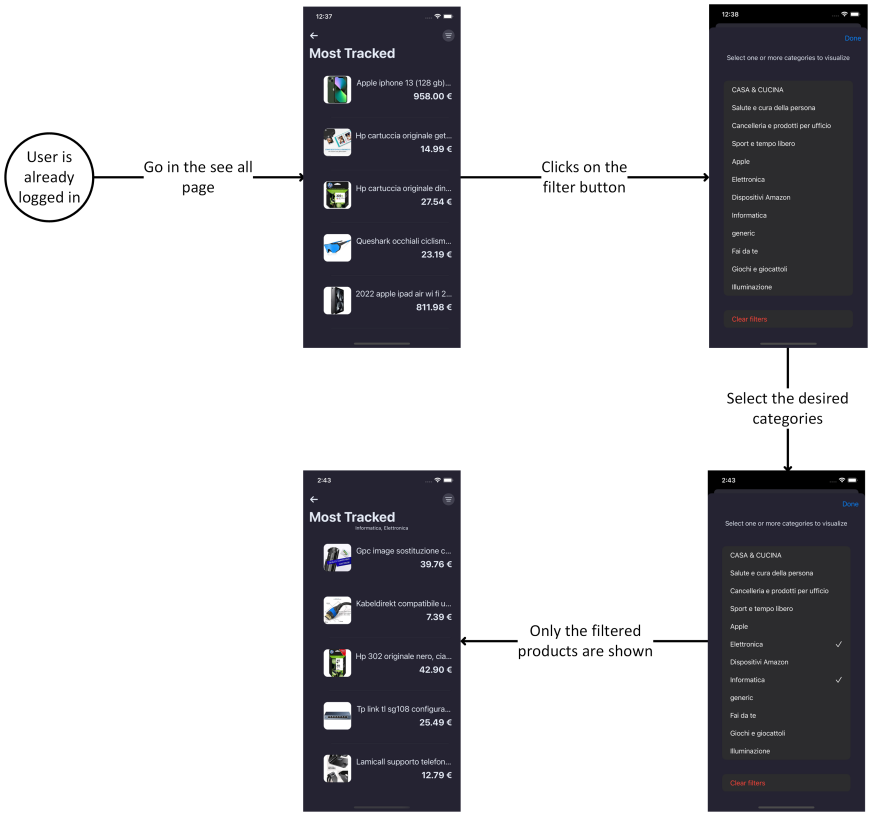


Figure 20: User filter by category

In this flow (fig: 20) the user wants to see only a certain subset of categories of products in the the see all view. In order to do that, the user decide to click on the filter button in the top right corner of the screen and at that point a sheet will be opened, showing the different categories from which the user could choose.
At this point the user selects a subset of categories he wants to visualize.
Finally, the user has to click on the confirmation button in the top right corner and the chosen subset will be shown.

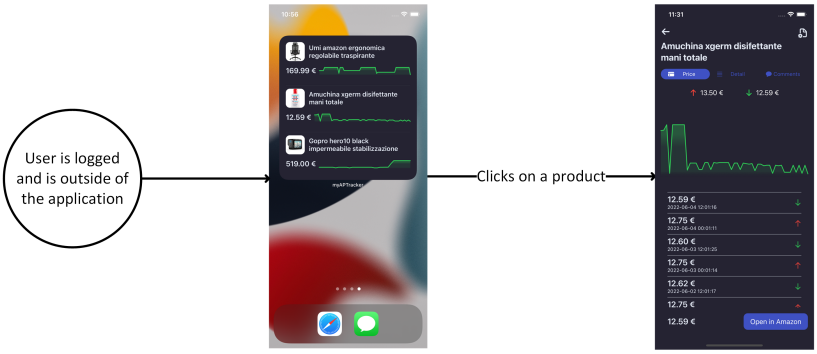


Figure 21: User use a widget

In this flow (fig: 21) the user wants to see a product because he saw from the widget that the product has had a price drop. In order to do that, the user clicks on the product he is interested in and myAPTracker will open and show the product clicked.

3.2 iPhone and iPad screen

In this section are presented all the view of our application for iPhone and iPad. In order to avoid duplicated images, the views that were equals for both devices are omitted. Finally, though the iPad could handle portrait and landscape, only the portrait views for the iPad are shown below.

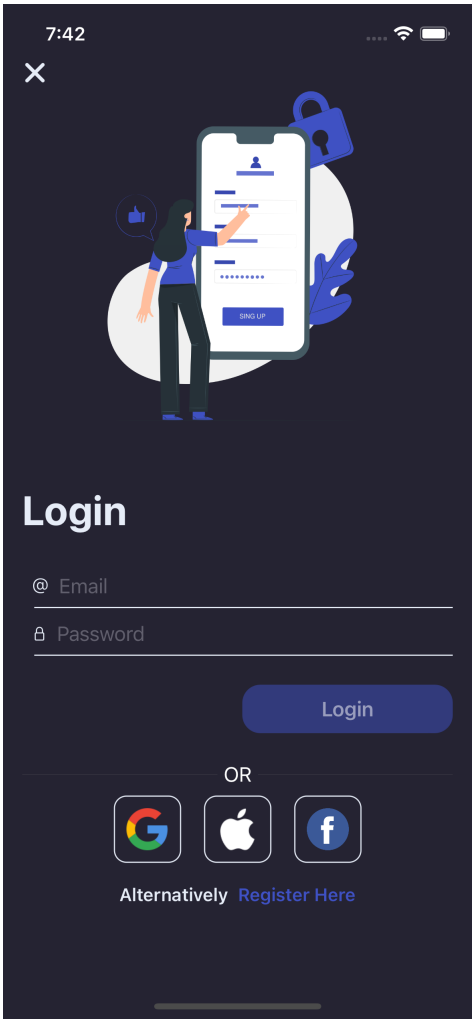


Figure 22: Log in screen

This is the login view (fig: 22). Here the user can login inserting its credential and click on login button. From this page the user can also authenticate by means of social such as Facebook, Google and Apple. Finally, if the user wants to create an account can navigate to the register page with the "Register Here" button.

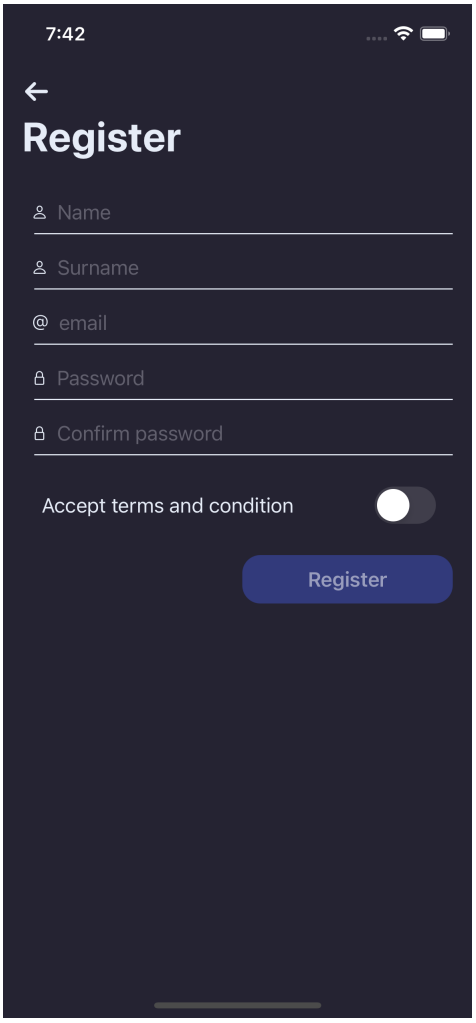


Figure 23: Registration screen

This is the register view (fig: 23). The user can register an account by filling all the requirement parameters, accepting the terms and condition and clicking on register.

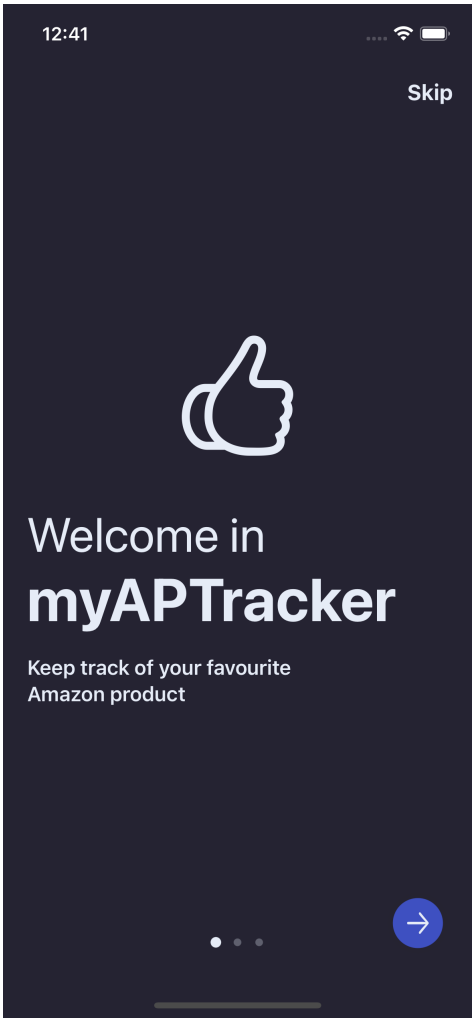


Figure 24: First screen of the tutorial

This is the first page of the tutorial (fig: 24). This view welcomes the user in the application. From here the user can continue the tutorial or skip it.

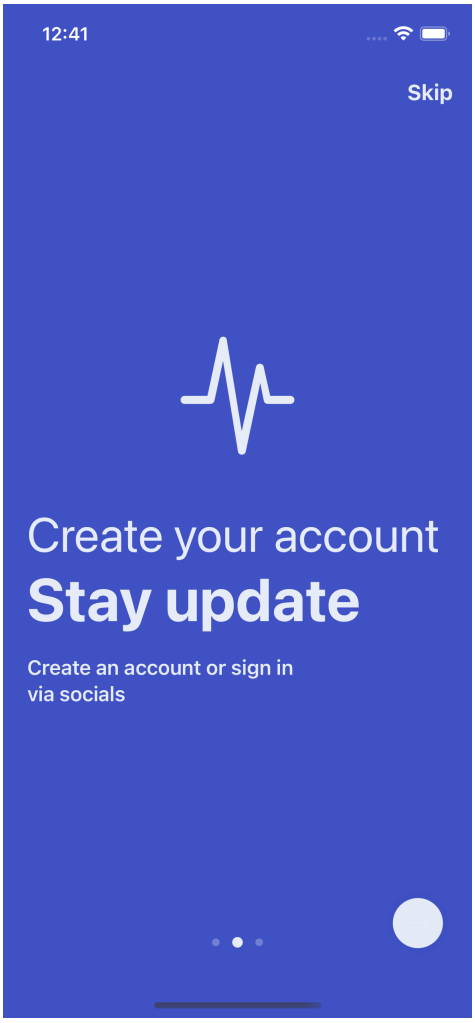


Figure 25: Second screen of the tutorial

This is the second page of the tutorial (fig: 25). This view invite the user to log in in order to have full access to the application. From here the user can continue the tutorial or skip it.

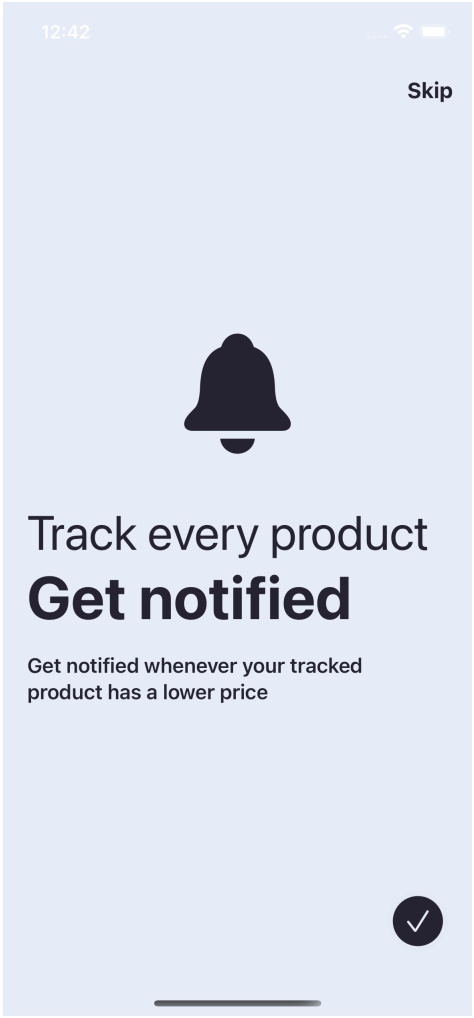
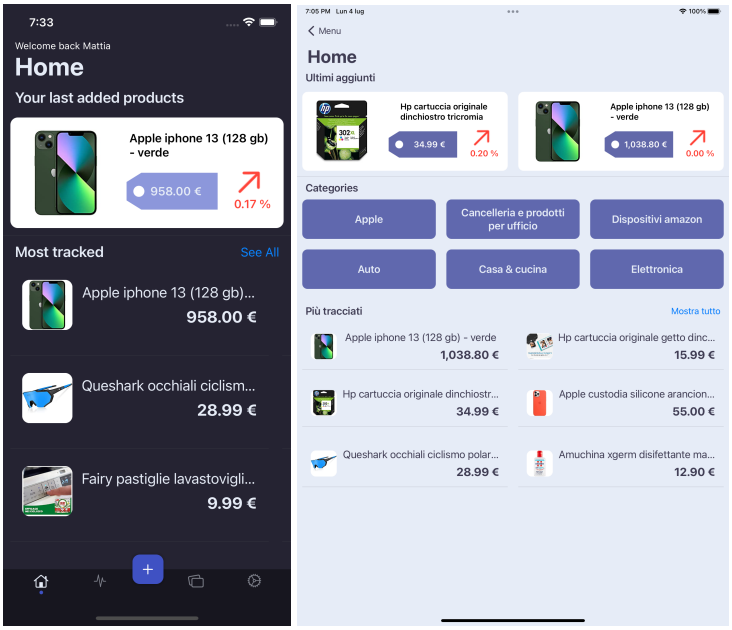


Figure 26: Third screen of the tutorial

This is the last page of the tutorial (fig: 26). This view invites the user to turn on the notification, because the application will notify the user when some products become available below the threshold set by the user. From here the user can exit from the tutorial.

Figure 27: User track a new product screen

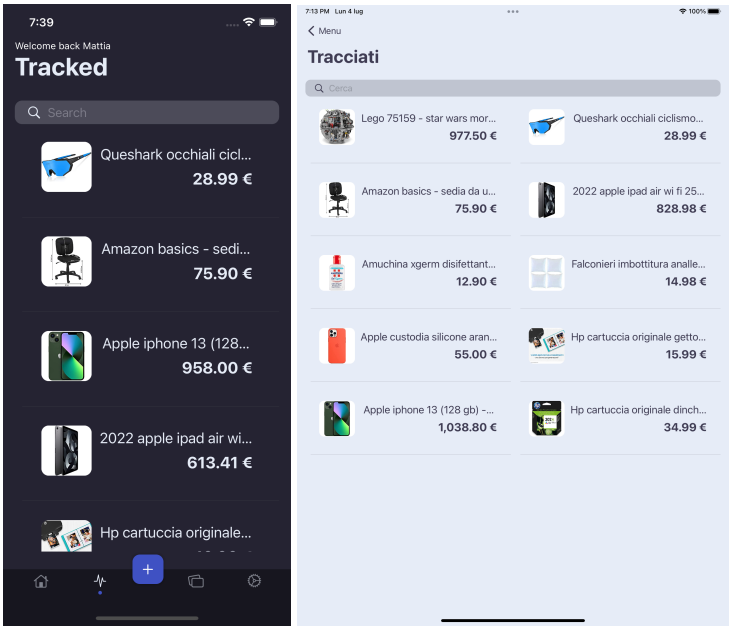
This is the view where the user can start track a new product (fig: 27). It is a web browser configured on Amazon website with an handler to detect if the page is a product or other screens. For security reasons the user is not allowed to leave amazon website.
The Amazon browser in accessible by clicking in the "plus" button from the main view which is a container with a tab bar to navigate between the 4 main screens (on iPhone) while on the iPad the same view is accessible from the application sidebar (see the end of the section).



(a) iPhone **(b)** iPad

Figure 28: Home screen

This is the home view (fig: 28) which is shown when the app finish launching. From here the user can scroll his last tracked product on the top or he can also navigate through the most tracked products. In addition, in the iPad interface, the user can also view a shortcut to access categories.



(a) iPhone **(b)** iPad

Figure 29: User tracked products screen

This is the tracked product view accessible only when a user is logged (fig: 29). From here the user can vertically scroll all his tracked products or can directly search for a specific product, using the search bar.

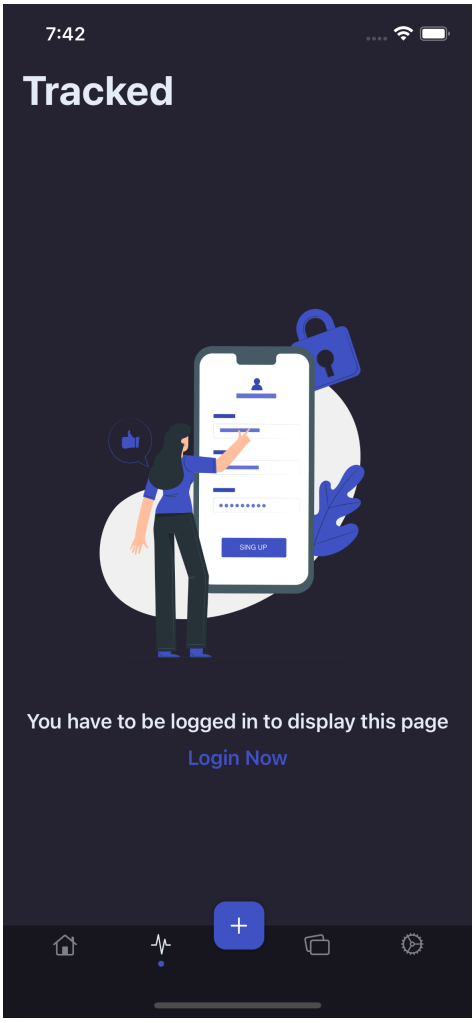
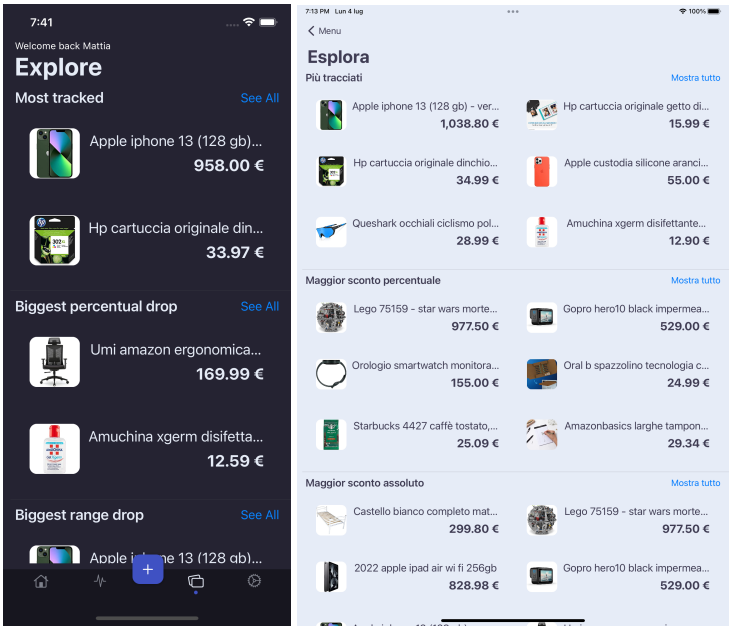


Figure 30: User tracked products screen, if user is not logged

This is the tracked product view when a user is not logged (fig: 30). Since the user can track product only if it is registered, this view is useless if the user is not logged, so it is only present a suggestion to invite the user to login.



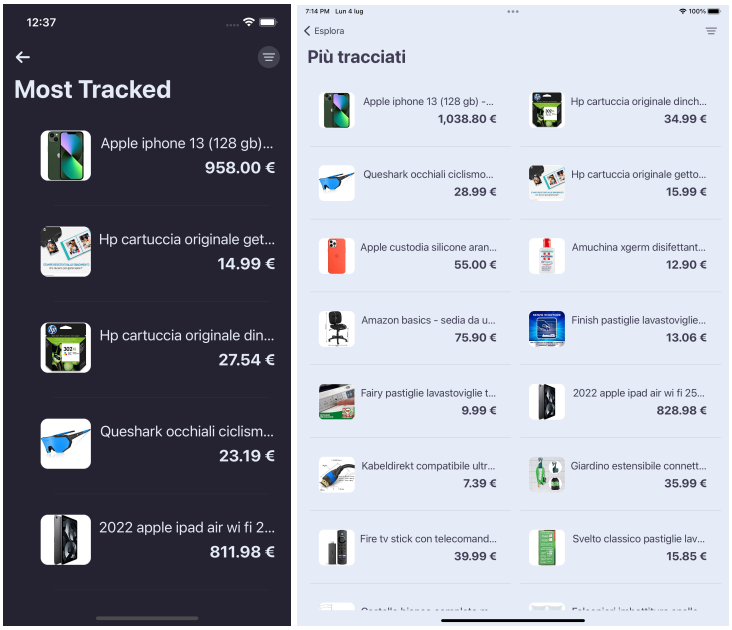
(a) iPhone **(b)** iPad

Figure 31: Explore screen

This is the explore view (fig: 31). From here the user can see three different interesting sections (on how the products are ordered):

- **Most tracked:** the most tracked product by application users are shown.
- **Biggest percentual drop:** product are ordered by percentual difference on price
- **Biggest range drop:** here are present the products that have had the highest range drop between maximum value and minimum value.

From all the described section (and also from the home) the complete list can be displays by clicking on the "See all" button near the section name:



(a) iPhone **(b)** iPad

Figure 32: See all explore screen

This is the see all view, all the products of the selected section are present (fig: 32).
This view is an infinite scrollable vertical stack view, where at first will be load only a certain amount of product and while scrolling down more product will be added to the stack dynamically. In this view it is also present a button in the top-right corner to filter them based on the category.

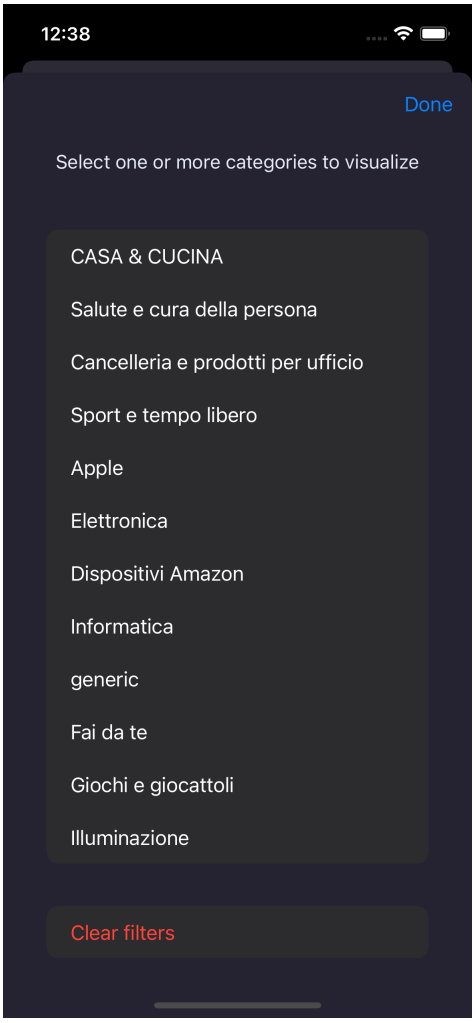


Figure 33: Category selection screen

This view (fig: 33) let the user filters between different categories to show only the relevant products to him. This view can be accessed from every see all section of the explore view (fig: 32).

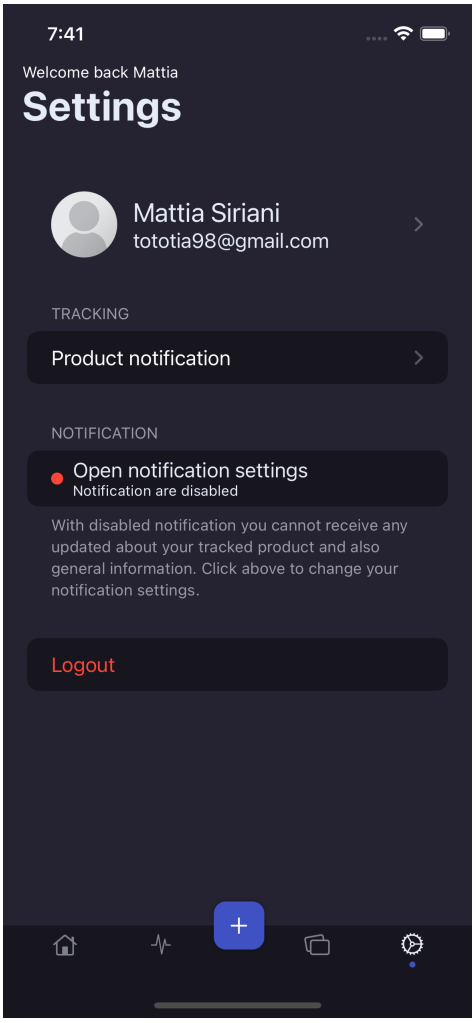


Figure 34: Settings screen

This is the settings view when the user is logged (fig: 34). From here the user can access to his personal information and modify them, change the general product notification (default value that triggers the notification, assigned to a product when is being tracked by the user), open the notification settings of the device and Logout.

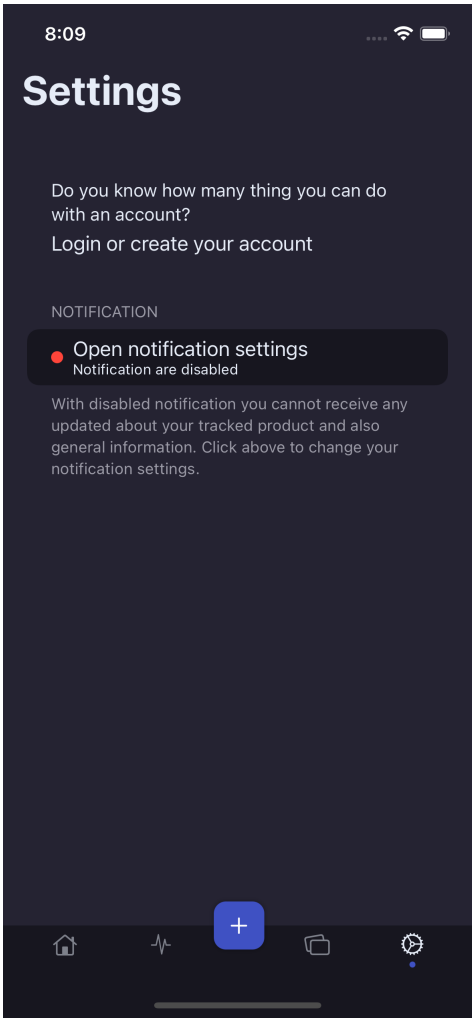
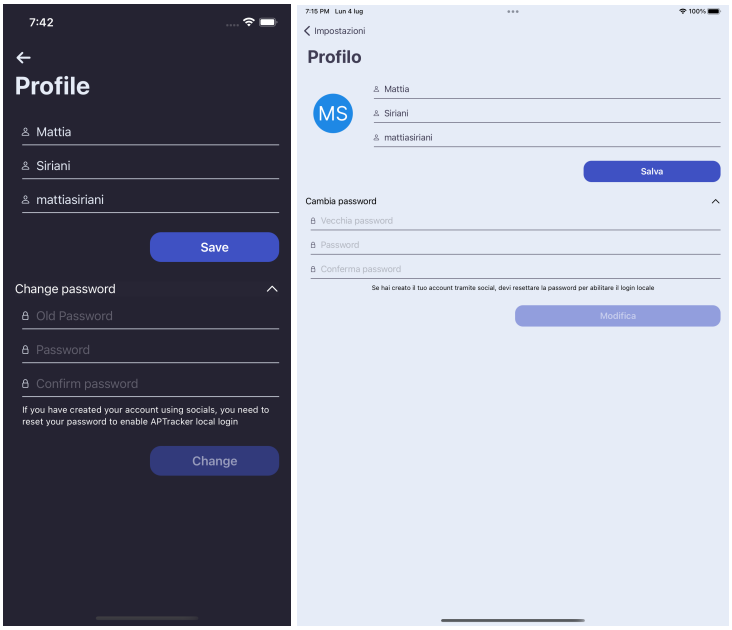


Figure 35: Settings screen, if user is not logged

This is the settings view when the user is not logged (fig: 35). From here the user can login in the application and open the notification settings of the device.



(a) iPhone **(b)** iPad

Figure 36: User profile settings screen

This is the view where the user can access his personal information(fig: 36). From here the user can change his personal information, such as name, surname, nickname, and password.

For the views regarding a product, there are always present three tab buttons that brings to:

- **Price view:** containing price information.
- **Detail view:** containing details of the product.
- **Comments view:** containing comments on the product leaved by other application users.

It is also present an horizontal stack at the bottom of the screen containing the actual price of the product and a link to open the product in the Amazon website.

Figure 37: Product screen

This is the view of a product (fig: 36). From here the user can access all the principal information regarding a specific product:

- **Graph:** a graph containing the ongoing prices of the specific products.
- **Peaks:** the minimum and maximum value since the product has been added to the system the first time.
- **Price history:** a vertical scrollable stack that containing all the prices since the product has been added to the system the first time until the last update.
- **Edit product:** a button in the top-right corner let the user to edit the product notification settings (if the user is tracking it, this option let the user to

override the default one) or stop tracking the product. If the product is not tracked yet, it can be added to the tracking list of the user.

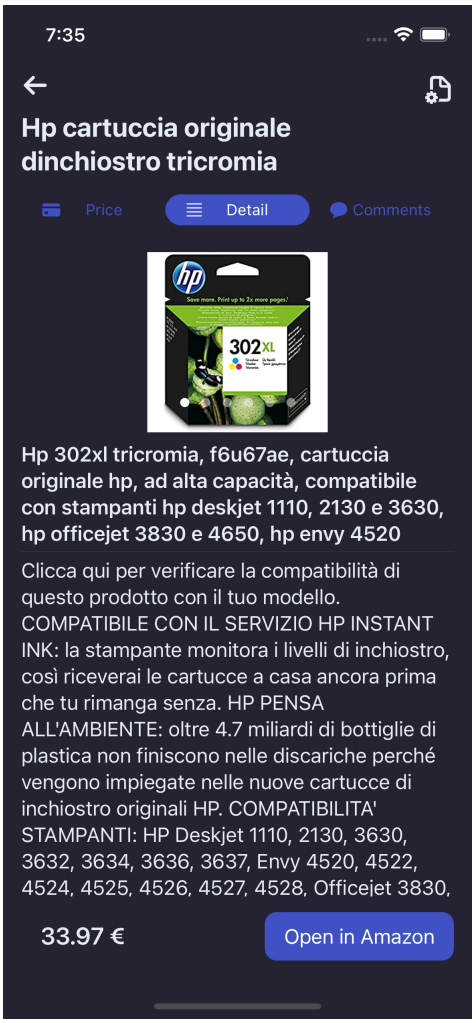


Figure 38: Detail product screen

This is the view containing the detail of a product (fig: 38). From here the user can browse all the photos of the product (full screen mode by clicking on an image), the full name, the category and the description.

The comments of the product are comment inserted by the user of myAPTracker and not comments of Amazon.

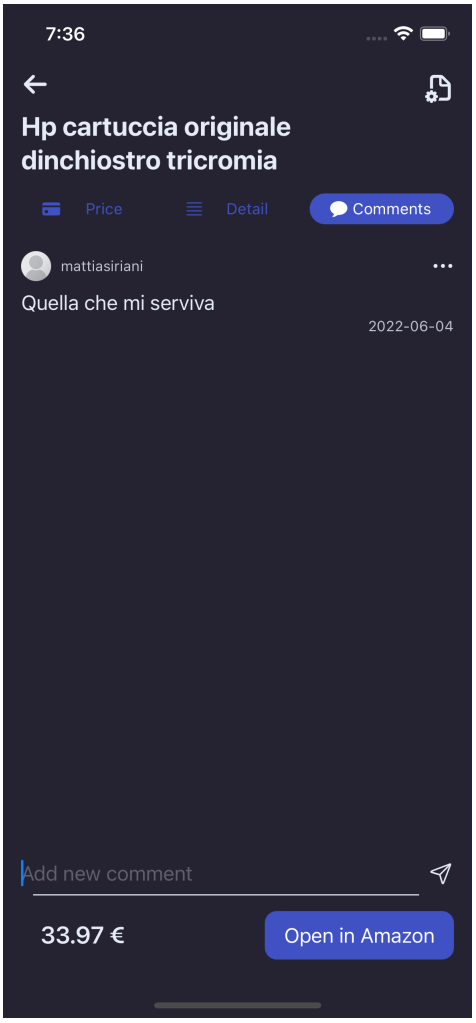


Figure 39: Comments product screen

This is the view containing all the comments that a product has ever received (fig: 39). From here the user can see the comments of the product or adding a comment itself. Finally, a user can delete its comment, after he post them.

The next screen are the iPad screen for the products and are put in a separate section, because here is not possible to switch between three sections, but only between two. In fact the price view is always visible on the left while the user can switch between detail and comments on the right.

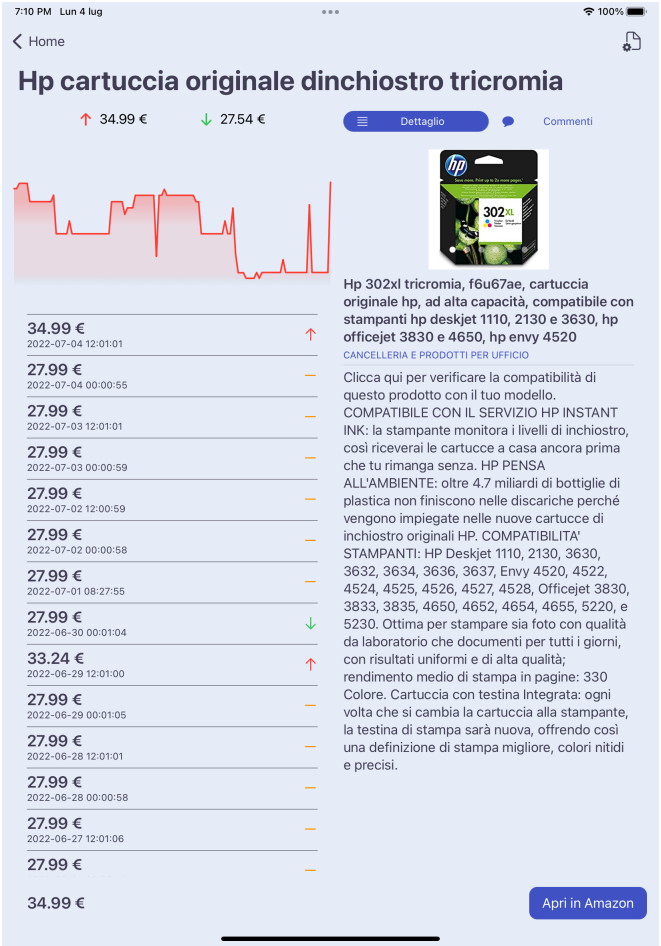


Figure 40: Product and detail screen

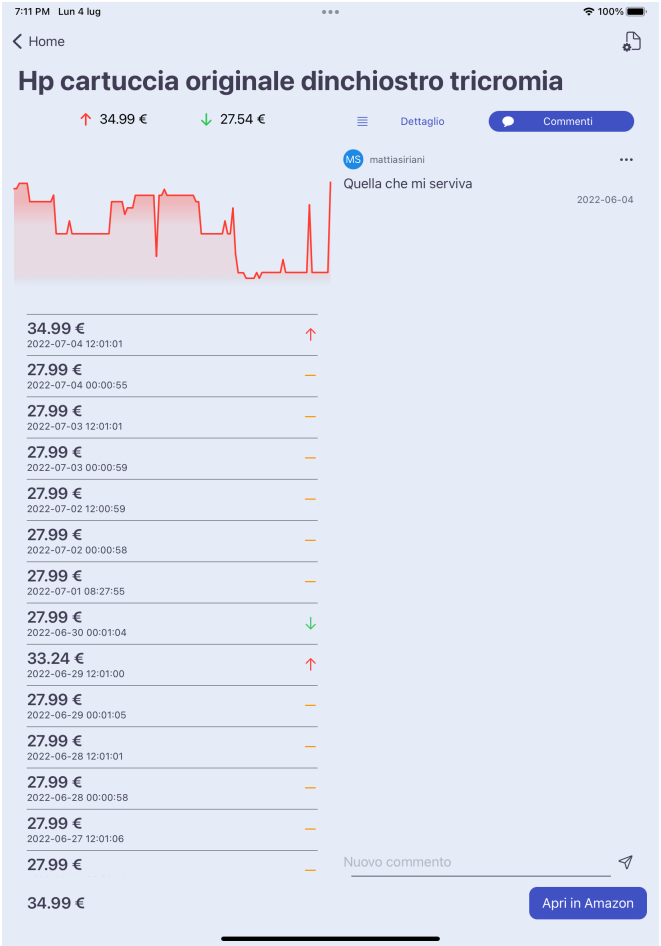
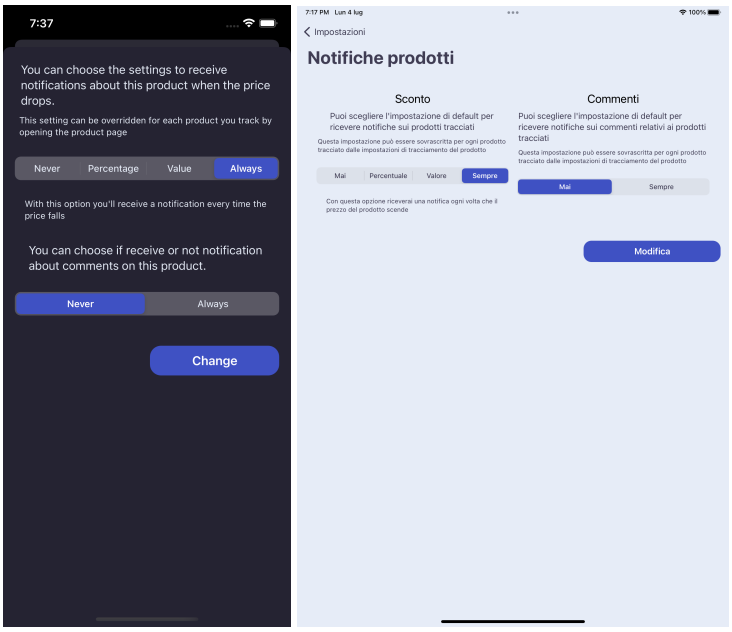


Figure 41: Product and screen

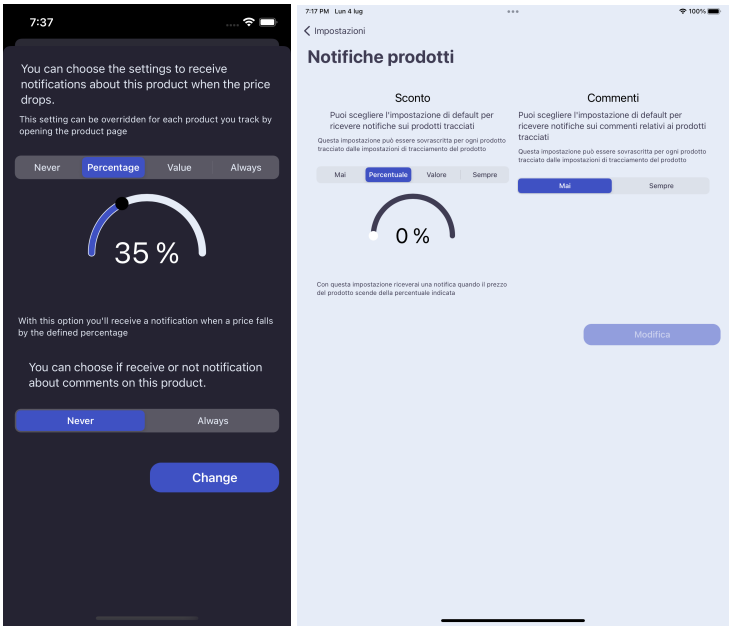
The following screen represents the tracking settings. Here, the user can choose if receive the notification about a price drop as well as a notification about a new comment posted in the product page. In the next pages we will explain better the available options about notification for products since for comments it is only possible to enable or disable them:



(a) iPhone **(b)** iPad

Figure 42: Receive notification always or never screen

In figure 42 the "always" option is selected, no further options are available (as well as "never") since a notification is sent any time the price falls down.



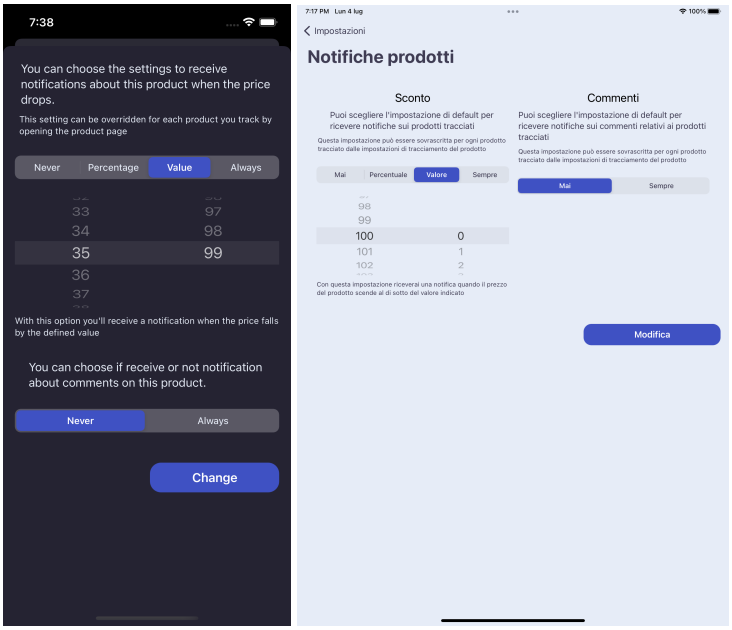
(a) iPhone **(b)** iPad

Figure 43: Receive notification from a percentage of the actual price screen

In figure 43 we can see the "percentage" setting, the half circle slider let the user to set the percentage value under which the notification is sent. For example, if the user set the value to 35% the notification is delivered by the system only if the difference between the last and the available price when the user starts tracking the product is grater or equals of the 35%.

$$(\frac{newPrice - targetPrice}{targetPrice} * 100) * (-1) \geq settingValue \quad (1)$$

Since in the interface the percentage is set as positive, we multiply the drop percentage for -1 in order the consider a discount as positive.



(a) iPhone **(b)** iPad

Figure 44: Receive notification if product is under a value screen

Finally in figure 44 we can see how the "value" option looks like. In this case the user can set a target price and when the product price in less or equal the target the user will be notified.

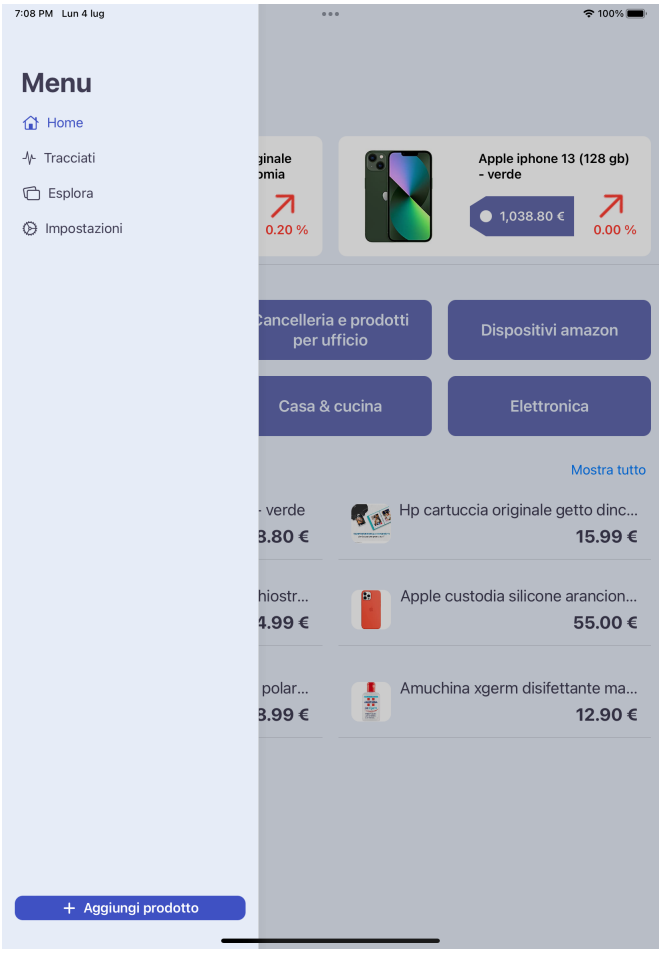


Figure 45: Menu iPad screen

An additional screen is required to be shown for iPad. This screen is regarding the menu. From here the user can navigate throughout the application.

3.3 Views used

The following views are sub-view (component) of the views described before.

- The bottom TabBar View (present only for iPhone) contains different section that the user can navigate:
  - **Home:** The main page of the application
  - **Tracking:** The view where products tracked by the logged user are visible
  - **Explore:** The view in which there are some suggestion of products divided in different section.
  - **Settings:** The view that let the user to customize settings of the application.

  This view has been used in the main views of the application. Precisely in the Home view, tracked product view, explore view and settings view (fig: 28 & 27 & 31 & 34).
- The graph view is used to show the ongoing price of each product. This view has been used in the home of each product (fig: 37).
- The prices history view (fig: 37) List that shows all the prices since the product has been added to the application the first time, each cell row has also an arrow that goes up or down, if the price of the product is increased or decreased. If the price is not changed a - symbol is shown. This view has been used in the home of each product (fig: 37).
- The HGrid view displays products in a horizontal ScrollView and it is used to show the most tracked product in the home of the application (fig: 28).
- The infinite scroll view displays N products in a vertical List (using a paging mechanism), when the user reach one of the last products, a new request for the successive page will be done and N more elements will be added to the vertical stack, until no more element will be available. This view has been used in the see all view of the explore section (fig: 32).
- The image visualization view (fig: 38) displays some images and let the user swipe horizontally between them. This view has been used in the detail section of a product (fig: 38).
- The search bar view let the user write in it, binding the searched text with the view model. This view has been used in the user tracked product view (fig: 29).
- The percentage circular slider view let the user slide on a semicircular slide to get a percentage value. This view has been used in the percentage view to decide when receive the notification (fig: 43).

- The vertical picker view let the user pick two different value, in order to form a price, with two different pickers. This view has been used in the value view to decide when receive the notification (fig: 44).
- The price tag view creates a visual price tag. This view has been used in the home of the application (fig: 28).
- The last tracked product view has only one product displayed per time, to see the other product the user needs to slide horizontally. This view have different element inside:
  - **Price tag:** with the actual price of a product.
  - **Image of the product**.
  - **Name of the product**.
  - **Ongoing of the product:** an arrow that goes up or down, if the price of the product is increased or decreased, with the percentage below it. If the price product has not been modified a - symbol is shown.

  This view has been used in the home of the application (fig: 28).
- The single comment view show a comment of a user. In this view is present: the username of the user that has published the comment with his avatar, the comment text, the date when the comment has been published and 3 dot symbol (visible only if the current user is also the comment creator) that allow the him to modify or delete the comment. This view has been used in the comment section of a product (fig: 39).
- Since SwiftUI doesn't provide a WebView, the WebView View is a UIViewRepresentable used to create a bridge between UIKit framework component (WKWebView) and SwiftUI. The UIViewRepresentable let also to connect the ViewModel used in SwiftUI view with the function implemented in the UIViewController by means of the Coordinator class that is the class in which the WKNavigationDelegate is implemented. This view has been used to show the Amazon website), through the click of the + button in the TabBar view; for that reason this view can be accessed from all the main views of the application (fig: 28 & 27 & 31 & 34).
- The loading indicator view is a view with an animation that shows the user that some loading is ongoing in the app. This view has been used in all the views the require a loading screen. For instance all the view that load content from the API.

3.4 Widget view used

Those are the available widgets for myAPTracker (figs: 46 & 47).
From here the user can keep under control the price and the graph of some products; also the user can access the product in myAPTracker by directly clicking on them.

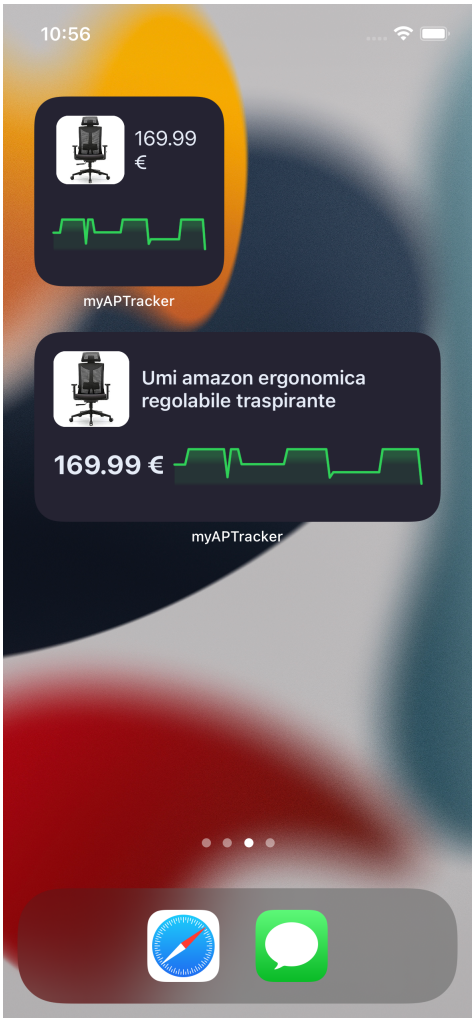


Figure 46: Small and medium widgets views

Figure 47: Large widget view

3.5 Apple Watch Application

In order to have a quick access to the most important section of the application, we have also provided a simple application for the Apple Watch.
In this application the user can see the 10 most tracked products and the top 10 products with the highest percentage price drop.
Finally, if the user is logged in his iPhone, he can see all his tracked products in his wrist.

Figure 48: Apple watch home

In figure 48 is shown the home of the application. This view is composed by the three possible sections that the user can navigate.

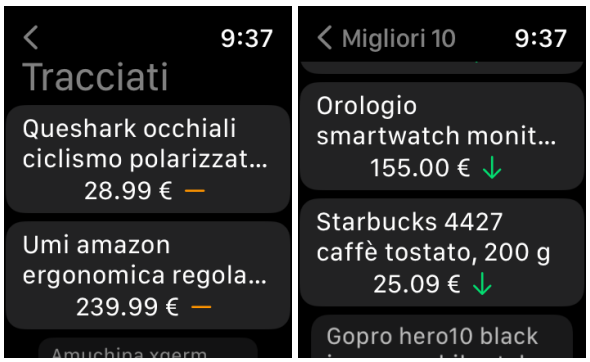


Figure 49: Apple watch list

In figure 49 is shown how each one of the three sections display the products. For every product the name and the current price is reported with a symbol which indicates if the price has fallen down (bottom green arrow), the price is growth (up red arrow) or it remained constant (orange dash).

Figure 50: Product Screen

In figure 50 is shown how the product screen looks like. Since in the Apple Watch the space is limited not all the information available in the iOS application were reported.
We decide to display the most relevant information for a user such as the name (with the category), the current price and the current discount if available, the highest and the lowest price in the product history and some images to better identify the product.

4 Implementation, Integration and Test Plan

The whole system presents the following components:

- Application (myAPTracker): the application for iOS downloadable from the app store.
- Web Server.
- Application Server.
- Internal Database.
- External services such as authentication via social and notification managing.

Those elements have been implemented following a bottom up approach, in order to have an easier testing. Our testing phase will mainly relies on UI testing, because most of the logic is on the back-end. The main focus will be on the Application and on the Application server, because in the Application relies all the visual part and is the point from which the user can interact with the rest of the system; while the Application Server part contains all the relevant functions that make the Application works properly.
The following table presents the main functionalities of the system, highlighting for each of them the importance for the costumer and the difficulty of its implementation.

Table 1 Implementation and testing precedence

Functionality	Importance for customer	Difficulty
Back-end	High	High
Application views	High	High
Local registration and Login	High	Low
Social registration and Login	High	Medium
Notification	Medium	High
Scraper	High	Medium

All the functionalities described in this document relies on a DBMS that has to be implemented firstly.
According to table 1 we decided to implement the functionalities following their importance.

- **Back-end:** the back-end contains the fulcrum of all the system, since contains all the functions that are essential for the functioning of the system. The back-end is the communication point between the DB and the application and this connection is possible thanks to the use of multiple APIs.
  The testing details are not relevant for the purpose of this project.

- **Scraper:** this functionality is of relevant importance to populate the DB with Amazon products.
  To test this functionality we have taken different products (available or not on Amazon) and verified if the JSON that output the scraper was the correct one.

- **Application views:** the application views are the most important part of myAPTracker. The views have been realized for iPad and iPhone, with different designs according to the device dimensions.
  The testing of this part is crucial and it has been done with the XCTest framework of Swift. Different test has been done to see if elements were appearing on screen, to test the functionality of some views and also visual test regarding the redraw of the view based on some logic.

- **Local registration and Login:** the two functionalities are related to the local DB where the credential of a user are stored and then retrieved.
  In this parts is important to test the ability to save and retrieve User attributes from the DBMS correctly.

- **Social registration and Login:** the two functionalities are related to different external services, such as Google, Facebook and Apple (implemented using different methods).
  In this parts is important to test the ability to authenticate the user using the different external services.

- **Notification:** this functionality is implemented through Firebase and the use of the device table in the DB.
  To test the notification mechanism a test notification could be send to all the devices or to a precise device through Firebase.

4.1 Testing

We test both the logic and the User Interface of our application. Several tests have been done for all the typologies of supported device (iPhone, iPad and Apple Watch). All the tests have a proper naming, defined by all the following items element separated by an _:

- test: all XCTest should start with the word test in order to be recognized as tests.

- Type of device: iPhone, iPad or Watch.

- Name of the view.

- Name of the UIElement or part of the view most involved in the current test.

- Action that the test is going to test.

For instance a proper name could be:
"test_iPad_SettingsView_UserProfileInformation_UserShowInfoAndGoBack". For all the UITest the override of the "setUpWithError" function has been exploited to set up all the different tests in order to reach its correspondent view. This has been done in order to reduce as much as possible the duplication of code.
Below are present the results of our tests, divided by type of tests.

4.1.1 Unit testing

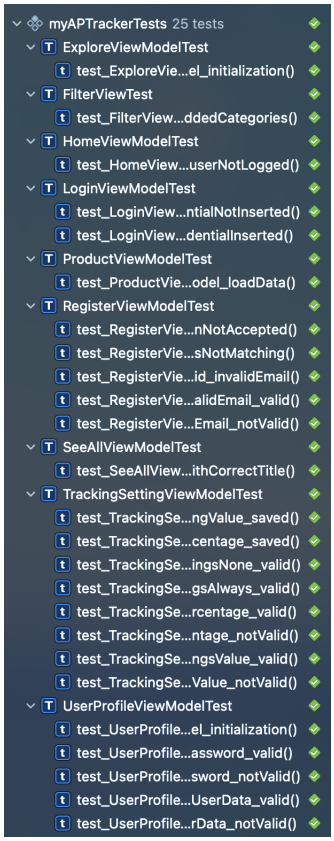
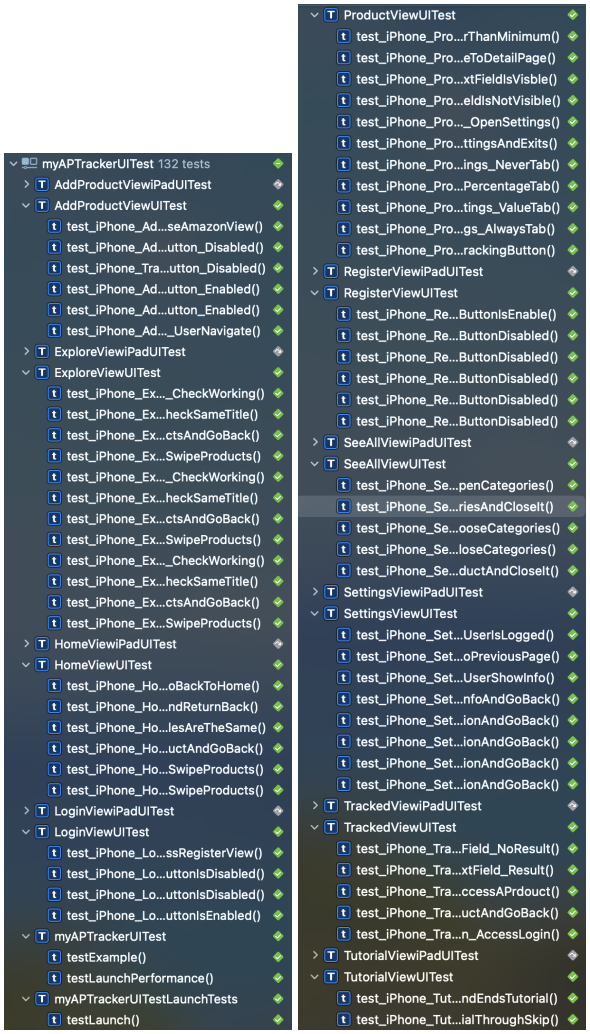


Figure 51: Unit testing

In those test (figure: 51) we have tested the logic of our application. Some tests regard internal logic of the application, for instance if the values in the Shared Preferences were corrected or if after the initialization of a ViewModel the variables are correctly initialized. While other test regard the testing of async calls to the API and the correct reception of the data, for the initialization phase and also for directly calls to the functions.

4.1.2 UI testing - iPhone



(a) iPhone 1 **(b)** iPhone 2

Figure 52: UI testing - iPhone

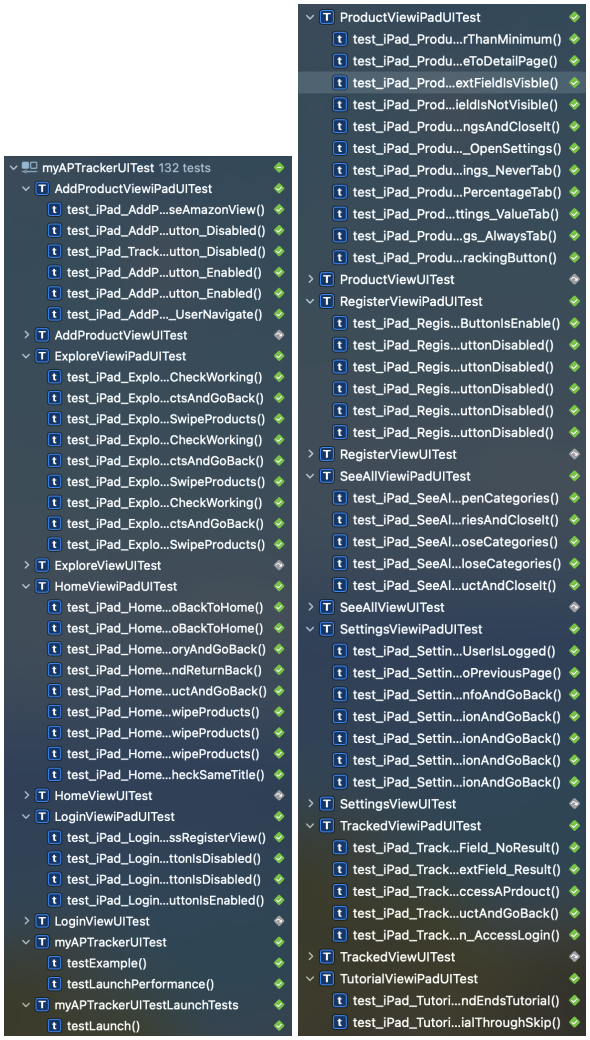
In those test (figure: 52) we have tested the UI of our application, only for the iPhone views, tests of iPad views are skipped. We have tested the general working of our application from the point of view of an external user, therefore:

- Navigation throughout the views.
- Presence of a precise view.
- Logic to enable buttons or disable it.

- Logic of different views (i.e. swipe to see more product, swipe to close a sheet, etc...).
- Navigation in the Amazon WebView.

To test the tutorialView, we have exploit the launchArguments() function of the XCTest framework, to pass as arguments a boolean to forcibly trigger (only for the test) the tutorial view to appear. All the test have been adapted to manage a user logged in, but also if it is not logged in; furthermore, some tests have been done specifically for logged user and for non logged one (for instance we have tested the navigation in the WebView, where a logged user should see the "Track product" button, instead a non logged user should see the "Add product" button). Finally, also the application flows (reported in section: "Application flows") have been tested.

4.1.3 UI testing - iPad



(a) iPad 1 **(b)** iPad 2

Figure 53: UI testing - iPad

In those test (figure: 53) we have tested the UI of our application, only for the iPad views, tests of iPhone views are skipped. We have tested the general working of our application from the point of view of an external user. We have tested the same things as for iPhone but in this case the views are different, so different tests are needed. Since the iPad supports both landscape and portrait orientation, the tests have been adapted to support both orientation. A possibility it would have been to duplicate the tests modifying only the "setUpWithError" function, but it would have lead to a complete duplication of all the iPad tests.

4.1.4 UI testing - Apple watch

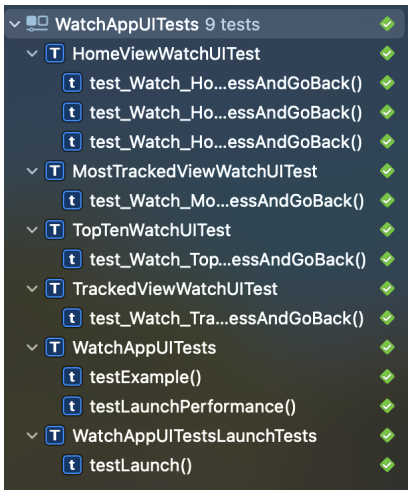


Figure 54: UI testing Apple watch

In those test (figure: 54) we have tested the UI of our application, only for the Apple watch views. We have tested the general working of our application from the point of view of an external user.

4.1.5 TestFlight

We have also created a test campaign through Apple TestFlight platform in order to have some people testing the application in a real scenario.
11 people tested the application functionalities alongside the implementation phase. In particular 11 beta have been released, the first 7 ones were released with the scope of testing new implemented functionalities, while the last 4 have been published in order to correct bugs and make improvements.
The last one is the current version (version 1.0 build 11) and it represents the pre-released one.

4.2 Further implementations

In this section will be presented some possible further implementations or improvements:

- **Internationalization of Amazon website:** let the user access the Amazon website, corresponding to the user language.
- **Internationalization of the application:** this means not only the translation of the texts (in order that each user have access to the product available in its language), but the conversion of the currency and the correspondent price, based on the location of the user.

- **Improve user suggestion:** this means adding a ML layer in the back-end that suggests to the user some products in which they could be interested in.
- **Support more devices:** this can include a better and native support for MacOS since now the iPad application can be cross compiled keeping its UI.