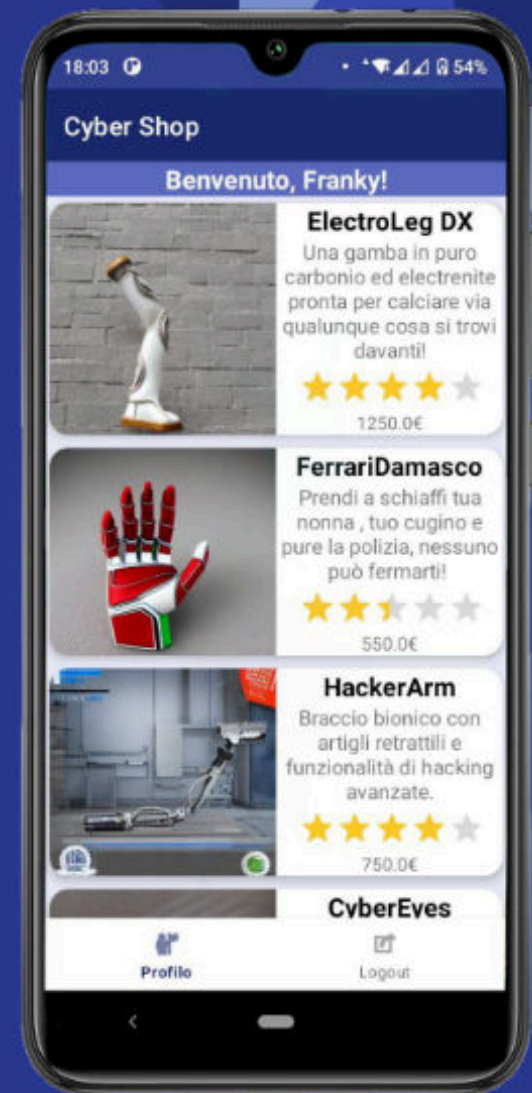


# CyberShop

Siena Lorenzo 1000015814

Computer Engineer



# What is CyberShop?

Cybershop is an app for managing a fictional cyberpunk-themed business in 2079.

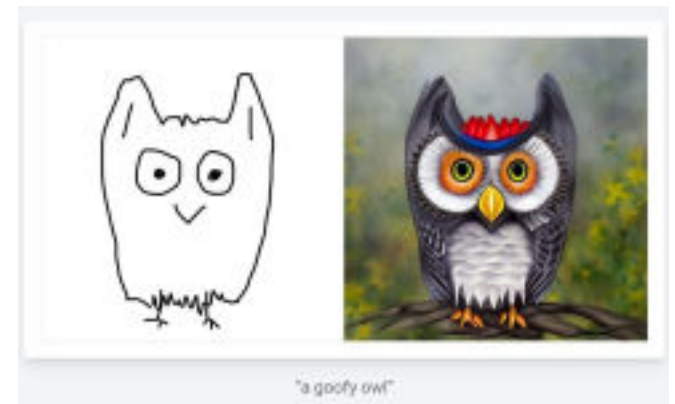
Its goal is to sell cybernetic prosthetics that can be viewed in the app through augmented reality or in 3D in the form of STL files, a CAD format, which, once purchased, can be printed at home using your own 3D printer enabled for printing biocompatible electronic implants and circuits.

All images were generated with an AI tool: **Scribble Diffusion**

(derived from ControlNet) capable of transforming prompts

and simple hand-drawn drawings into neural network-generated images.

The project is Open Source and licensed under the MIT license.



# What is CyberShop?

The app itself meets the project specifications and allows:

User registration and authentication.

Access to a home page with the products available in the store.

The ability to add a product to the user's wishlist and/or review it by giving 1 to 5 stars.

Access the authenticated user's profile to view their wishlist.

Access to an admin dashboard where you can add, edit, and delete products directly from the app, while viewing your store's basic data and statistics.

# Architecture, paradigm and development

**Several possible approaches to application development were evaluated:**

Activity and fragment with layout reuse.

A single Activity architecture and many fragments to be reached with a Navigator. But due to the complexity of the design and the generality of the object-oriented paradigm, where the possible implementations are virtually infinite, a KISS (Keep it simple, stupid!) approach was chosen, using one Activity and one layout per screen, maintaining the application state through Intents in order to display the necessary information.

# The database

The following FireBase services were chosen to save the information to be loaded and displayed dynamically by the client:

**Authentication:** to manage email, user UID and authentication.

**Realtime Database:** list of products ,users and wishlist.

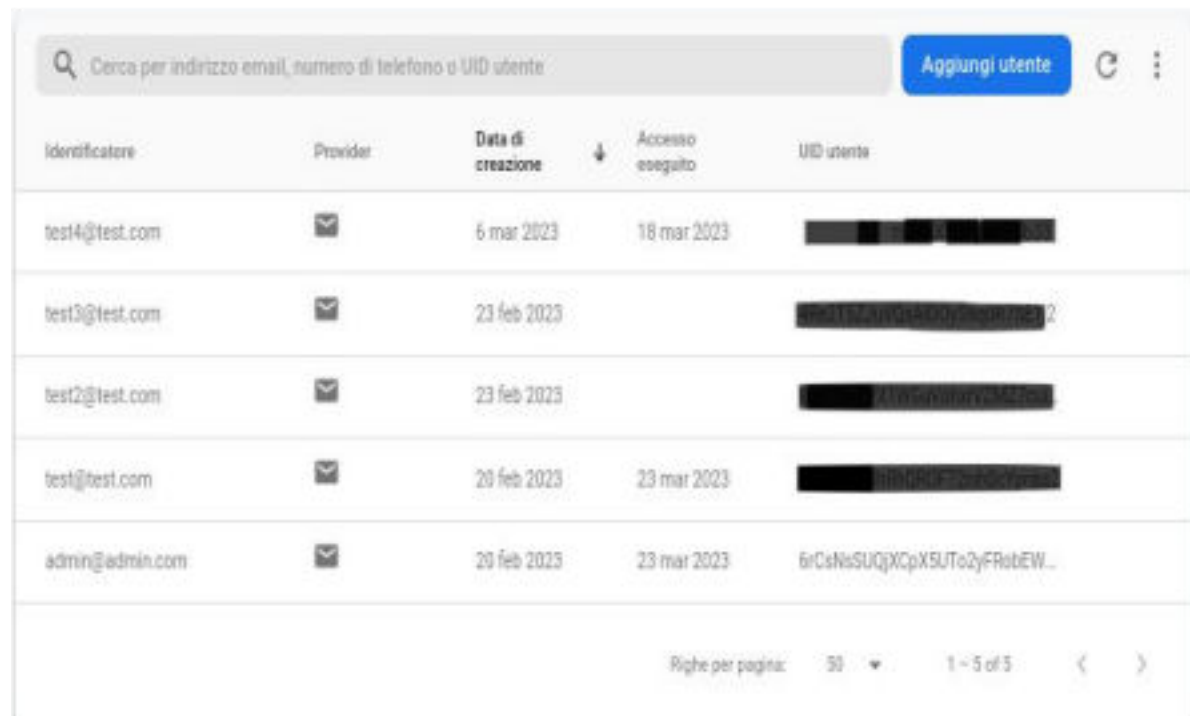
**Storage:** where the product image and STL are saved

# Authentication database

Profiles are saved here for use in the application. For testing purposes, each user has the following password: **password** except the administrator of the shop that accesses with

user: **admin@admin.com**  
password: **administrator**

It is possible to register  
and `SignupActivity`  
e logger then `LoginActivity`



The screenshot shows a web interface for managing an authentication database. At the top, there is a search bar with the placeholder text "Cerca per indirizzo email, numero di telefono o UID utente" and a blue button labeled "Aggiungi utente". Below the search bar is a table with the following columns: "Identificatore", "Provider", "Data di creazione", "Accesso eseguito", and "UID utente". The table contains five rows of user data. The last row is for the administrator user, "admin@admin.com", which has a unique UID. The first four rows are test users with email addresses like "test14@test.com". At the bottom right, there is a pagination control showing "Righe per pagina: 50" and "1 - 5 of 5".

Identificatore	Provider	Data di creazione	Accesso eseguito	UID utente
test14@test.com	✉	6 mar 2023	18 mar 2023	[redacted]
test13@test.com	✉	23 feb 2023		[redacted]
test2@test.com	✉	23 feb 2023		[redacted]
test1@test.com	✉	20 feb 2023	23 mar 2023	[redacted]
admin@admin.com	✉	20 feb 2023	23 mar 2023	6rCsNsUQjXCpX5UTo2yFRbtEW...

# Realtime Database

"products"

Here is the list of products in the shop.

The **product** branch has as its children random **UID** keys generated during product creation with the following attributes:

**name:** string

**description:** string

**price:** float

**urlImage:** Path to Firebase that identifies the resource to be used as an image for the product

**urlStl:** Path to Firebase that identifies the resource to be used as a 3D file in STL format for the product



# Realtime Database

"products"

## reviews:

here are the ratings from  
1 in 5 of users according to the  
structure

**UID user :rating** the average rating of the product is  
calculated by the class

RecyclerViewAdapterLong  
when the product is displayed  
as card.

**admin:** **true** for admin only

and **false** by default when creating a new user

**mail:** which matches the email saved in

Authentication **name:** user's real name

**surname:** user's real surname

**wishlist:** a list containing the product key and the entire product  
duplicated by the branch "products".

N.B. This type of redundancy is a design pattern known as

**Denormalization** and one best practices per Firebase

both for the class methods `FirebaseRecyclerOptions` which  
belongs to `FirebaseUI` and which accepts entire objects when building  
and does not allow using a list of keys.

Added to this is the fact that when scaling the application, database  
access times in the event of joins become a significant latency and that  
the Nosql nature of Realtime does not allow joins.





# Realtime Database

“user\_wish”

To solve the problem of multiple product paths and therefore multiple ratings that could compromise data consistency and the average rating calculation, a look-up table has been implemented that takes into account the products in the wishlist per user.

The **user\_wish** branch holds a child node identified by the **UIDuser**. This node contains a product map, where a **true** value signifies that a product is on the user's wish list.

When a user adds a product to the wishlist, an entry is created;  
when it is deleted, the node is deleted.

This look up table is consulted when assigning a rating to a product and must be propagated to all the necessary paths.



# Realtime Database

“multipath update”

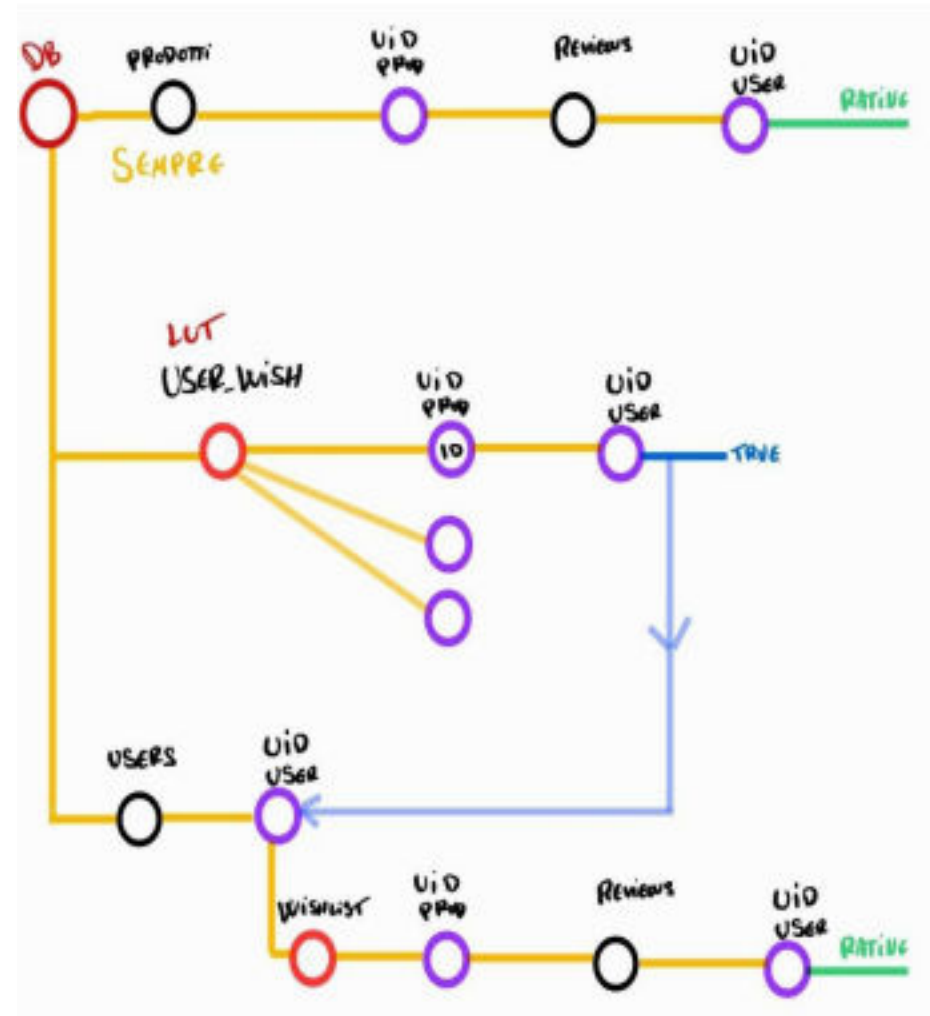
PATH:

“products/product”

“users/(UID\_USER)/wishlist/product”

“user\_wish/(UID\_USER)/(UID\_product:true)”

On the right you can see the routes for the “multipath update” which happens as an **atomic operation** in FullProductActivity synchronizing products and ratings.



# Storage database

Here are saved the images and stl to be loaded in the previews, and to be downloaded



The screenshot shows a Google Drive interface with the address bar displaying 'gs://cybershop-b79a9.appspot.com'. In the top right corner, there is a blue button labeled 'Carica file' (Upload file) and a plus icon. Below the header, there is a table with four columns: 'Nome' (Name), 'Dimensioni' (Size), 'Tipo' (Type), and 'Ultima modifica' (Last modified). The table contains two rows representing folders: 'FILE\_STL/' and 'IMAGE\_STL/'. Both folders are listed with a size of '-' and a type of 'Cartella' (Folder). Each row has a checkbox in the 'Nome' column.

<input type="checkbox"/>	Nome	Dimensioni	Tipo	Ultima modifica
<input type="checkbox"/>	FILE_STL/	—	Cartella	—
<input type="checkbox"/>	IMAGE_STL/	—	Cartella	—

Product positions in RealTime database, **urlImage** and **urlStl** reflect the format


*gs://xxxxxxx.appspot.com/IMAGE\_STL/file\_name.png*

*gs://xxxxxxx.appspot.com/IMAGE\_STL/file\_name.png*

# Storage database

"IMAGE\_STL"

Images in IMAGE\_STL are loaded by the application using the Glide plugin, which takes the storage location and sets its reference to the ImageView, which then requests it.

gs://cybershop-b79a0.appspot.com > IMAGE_STL					Carica file
<input type="checkbox"/>	Nome	Dimensioni	Tipo	Ultima modifica	
<input type="checkbox"/>	tender_v1.png	408.56 KB	image/png	6 mar 2023	
<input type="checkbox"/>	ceramic_leg.png	534.42 KB	image/png	28 feb 2023	
<input type="checkbox"/>	cuore_elettrico.png	488.69 KB	image/png	28 feb 2023	<div>cuore_elettrico.png</div>  <div>Nome <a href="#">cuore_elettrico.png</a></div> <div>Dimensioni 498.370 byte</div> <div>Tipo image/png</div> <div>Data di creazione 28 feb 2023, 18:50:52</div> <div>Data di aggiornamento 28 feb 2023, 18:50:52</div> <div>Posizione file gs://cybershop-b79a0.appspot.com/IMAGE_STL/cuore_elettrico.png</div>
<input type="checkbox"/>	cuore_elettrico2.png	583.2 KB	image/png	28 feb 2023	
<input type="checkbox"/>	cyber_sar.png	589.12 KB	image/png	28 feb 2023	
<input type="checkbox"/>	cyber_eyes.png	458.96 KB	image/png	28 feb 2023	
<input type="checkbox"/>	cyber_hand_gun.png	362.84 KB	image/png	28 feb 2023	
<input type="checkbox"/>	electricty.png	671.85 KB	image/png	28 feb 2023	
<input type="checkbox"/>	ethereal_tent.png	423.47 KB	image/png	28 feb 2023	
<input type="checkbox"/>	ironman_hand.png	416.35 KB	image/png	28 feb 2023	

# Storage database

"FILE\_STL"

gs://cybershop-b79a9.appspot.com > FILE\_STL

Carica file

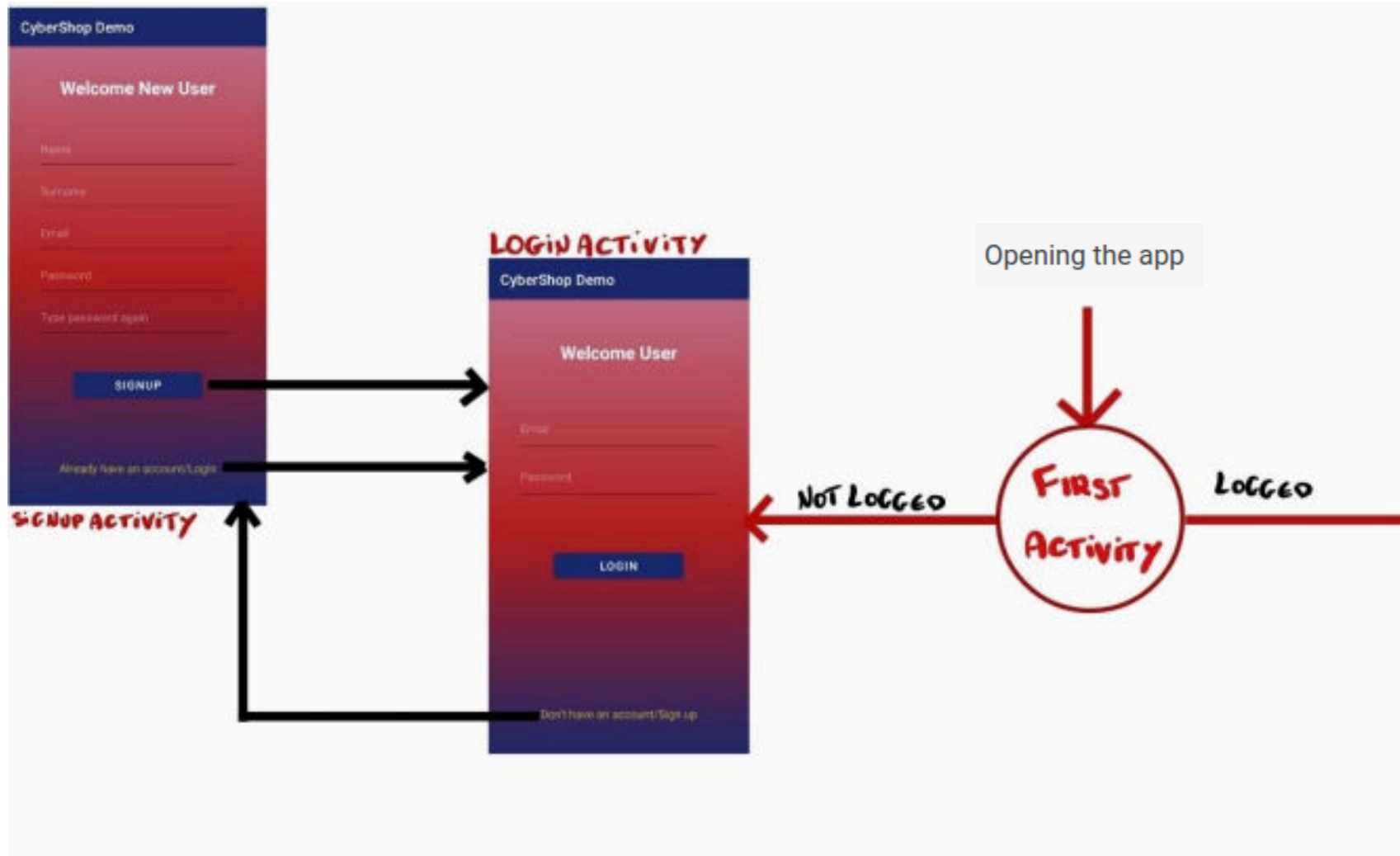
<input type="checkbox"/>	Nome	Dimensioni	Tipo	Ultima modifica
<input type="checkbox"/>	 bender_v1.stl	3.61 MB	model/stl	28 feb 2023
<input type="checkbox"/>	 d20.stl	226.06 KB	model/stl	28 feb 2023

Here are 2 test stl files ready to use and to print, that leave room for future implementations not present in this demo, as the library ARCore is **not** implemented as per specifications.



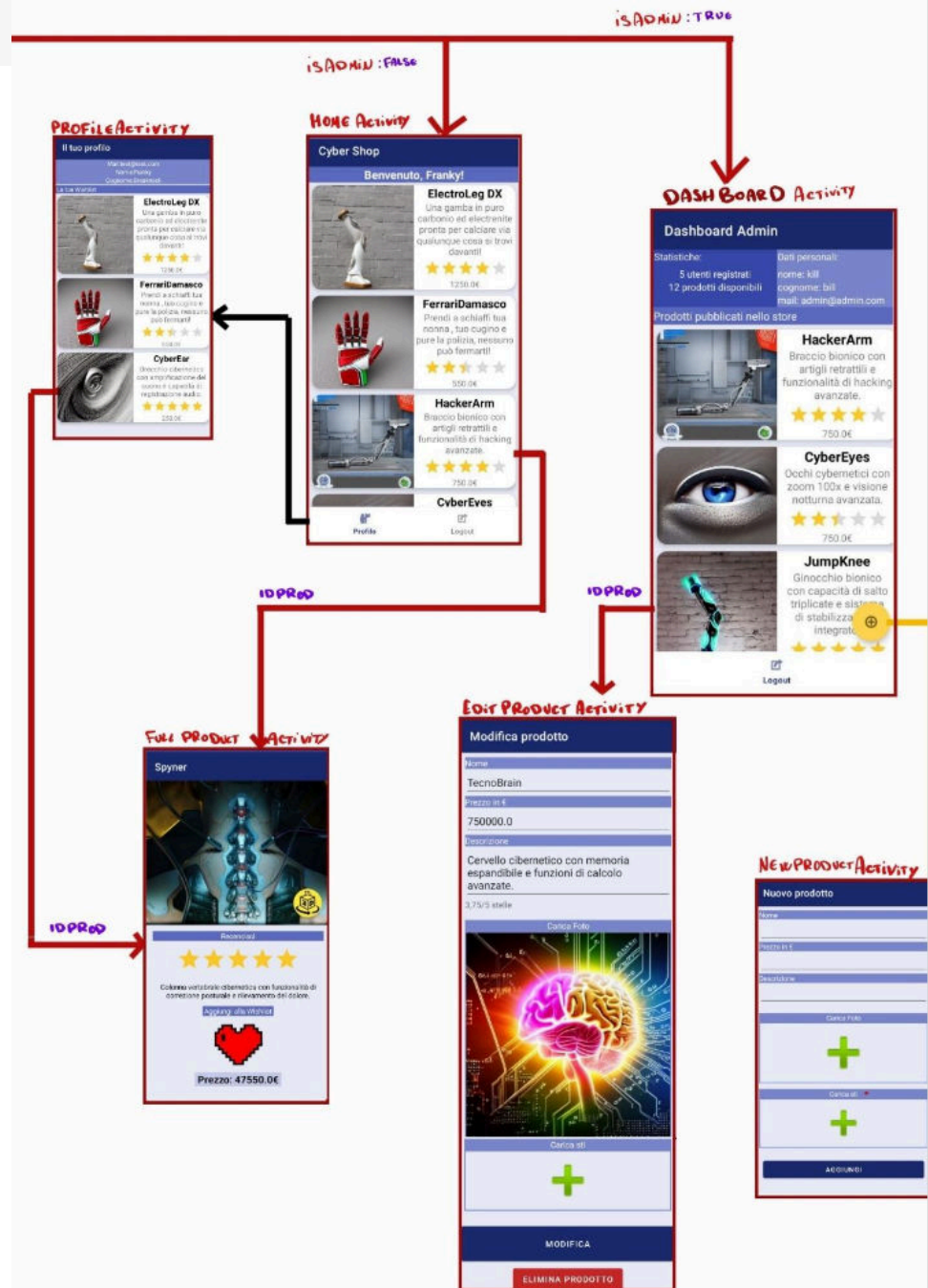
# Navigation structure

## Not Logged in



# Navigation structure Logged in

logged user



# FirstActivity

An entry activity that redirects the user into 3 possible states:

Logged in with User rights sent to the *HomeActivity*

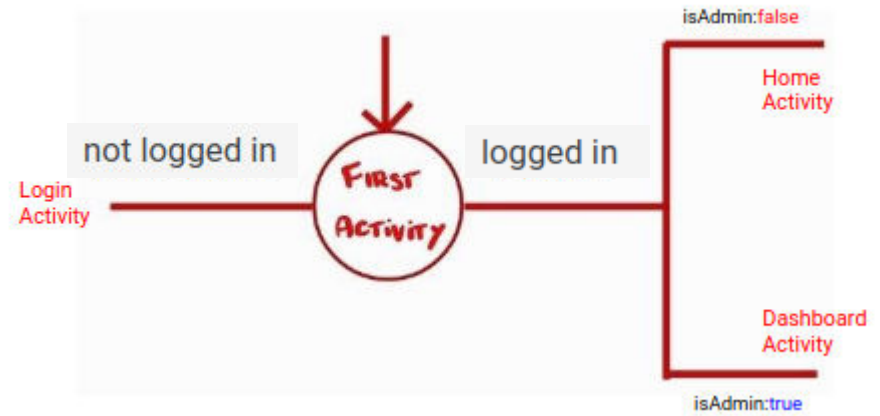
Logged in with Superuser rights sent to the *DashboardActivity*

Not logged in sent to the *LoginActivity*

## DATABASES INVOLVED:

**FirebaseAuth and Realtime Database**

It checks whether the current user is authenticated, exists in the database, and is an admin.





# LoginActivity

Activity that sets 2 listeners on the login button and on the signup text

If clicked **login**:

It checks the credentials entered and, based on the user type, sends an intent to the home or the dashboard.

If clicked **Don't have an account/Sign up**

it sends the user to the registration page.

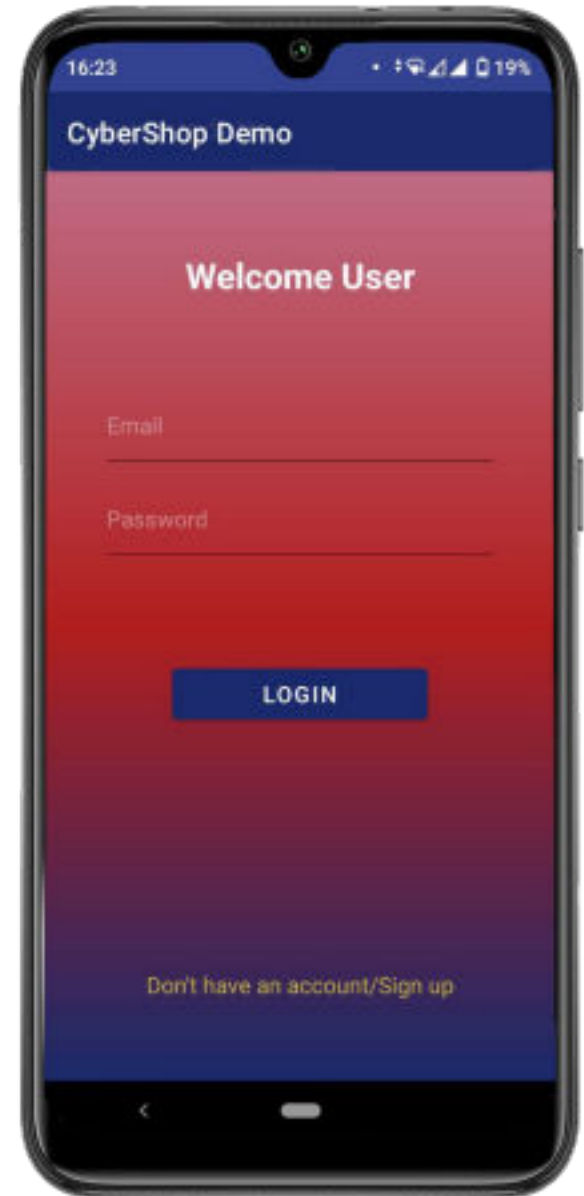
## Databases involved

[*FirebaseAuth*]

for User Authentication.

[*Realtime Database*]

for checking that the user is properly registered.



# SignupActivity

Registration activity that sets 2 listeners on the signup button and on the login text

When the user clicks on **signup**:

Check that the data entered is not empty and that it is valid. If so, redirect to the login page.

If clicked **Already have an account/Login**:  
sends the user back to the login page.

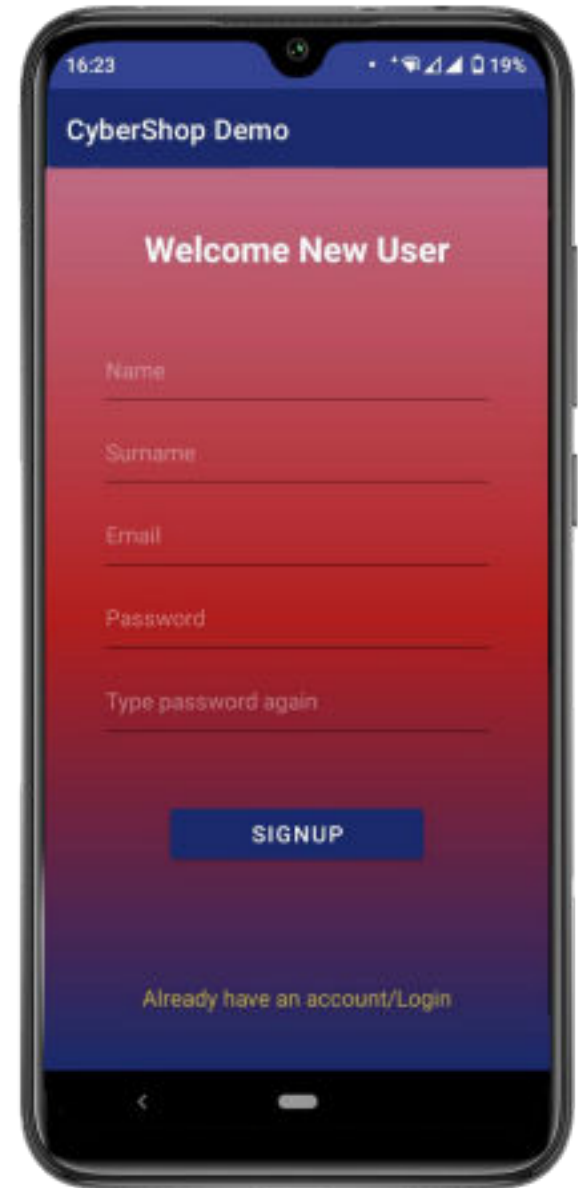
## Databases involved

[*FirebaseAuth*]

for User creation

[*Realtime Database*]

for checking that the user has been created



# HomeActivity

Initialize a menu to be placed below that leads to: user profile [ProfileActivity] or to logout [LoginActivity]

Extracts the list of products from the database and inserts it into the recyclerView where it displays the products one by one in the form of a card [long\_card.xml]

Implement and set up a listener for each card that sends the user and product id clicked to [FullProductActivity] so you can view it in full detail.

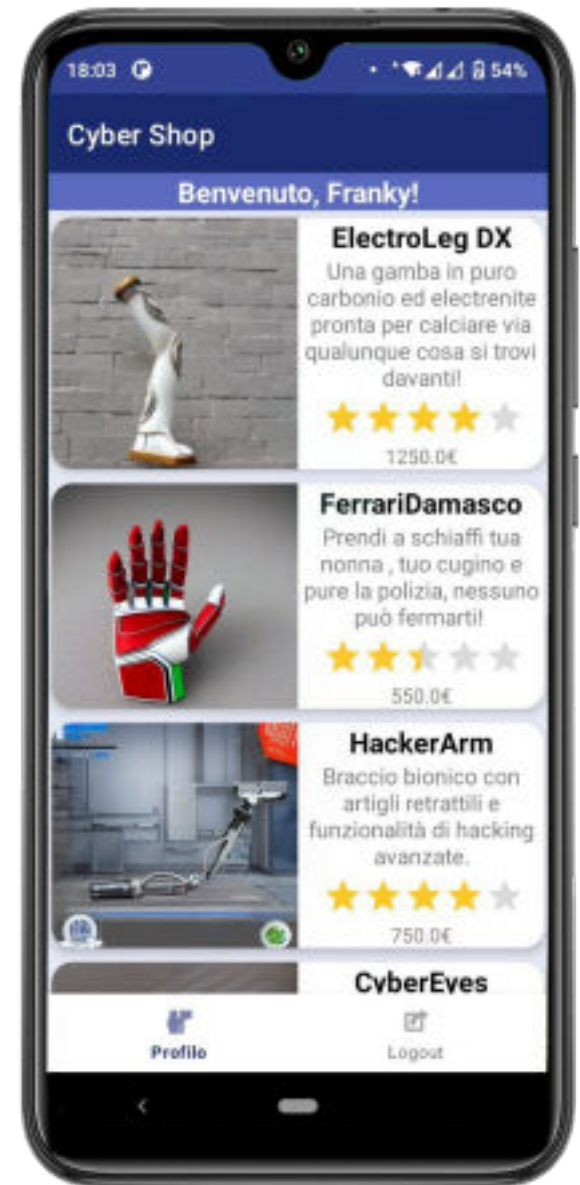
## Databases involved

[FirebaseAuth]

Read the username to greet the user in the top screen

[Realtime Database]

Reads the “products” list and displays them on the screen.



# FullProductActivity

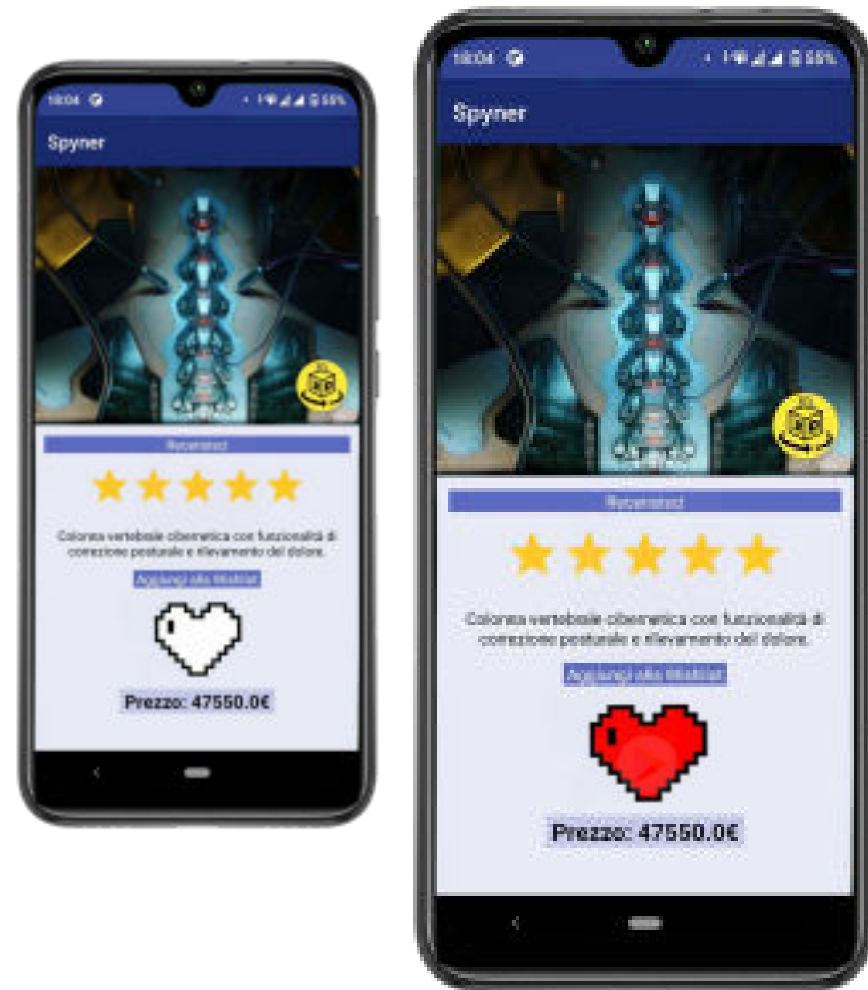
Activity that has the task of displaying a product in the shop (if it comes from HomeActivity) or in the wishlist (if it comes from ProfileActivity)

It receives from the previous Activity an intent with the id of the product to display

From here you can:

- View **name**, **description**, **image**, **price** and previous **reviews** with 1 to 5 stars, with 0 stars if never reviewed.
- View\* the product's STL in 3D through a **fab**.
- Add the product to your wishlist
- Add a new product review [1-5] stars

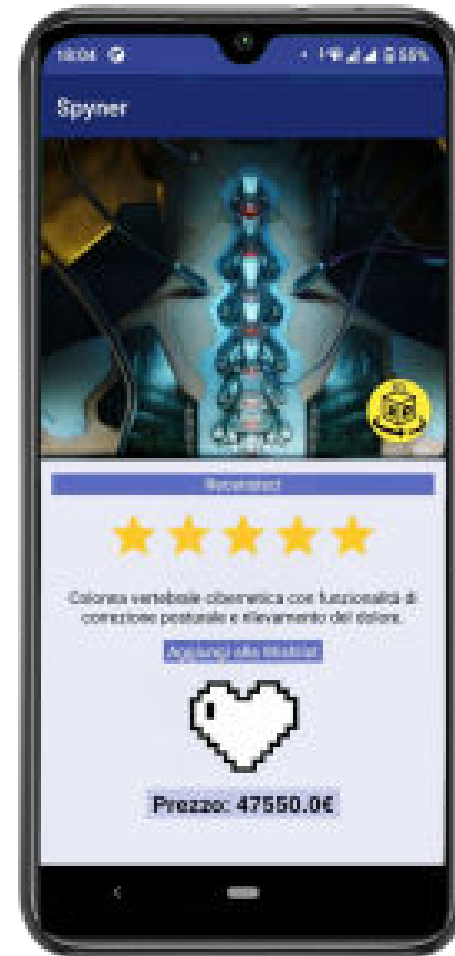
\*[Feature not present in this demo as ARCore is not supported nor implemented as per specifications, by pressing the fab a Snackbar message warns the user, The stl file is in any case available and functional in the database [Storage] for later implementation]\*




# FullProductActivity

The Activity also sets listeners to:  
Add a review to the  
product from 1 to 5 stars,  
via **onRatingChanged**  
checking and filling the  
bar with the previous  
review, if present.

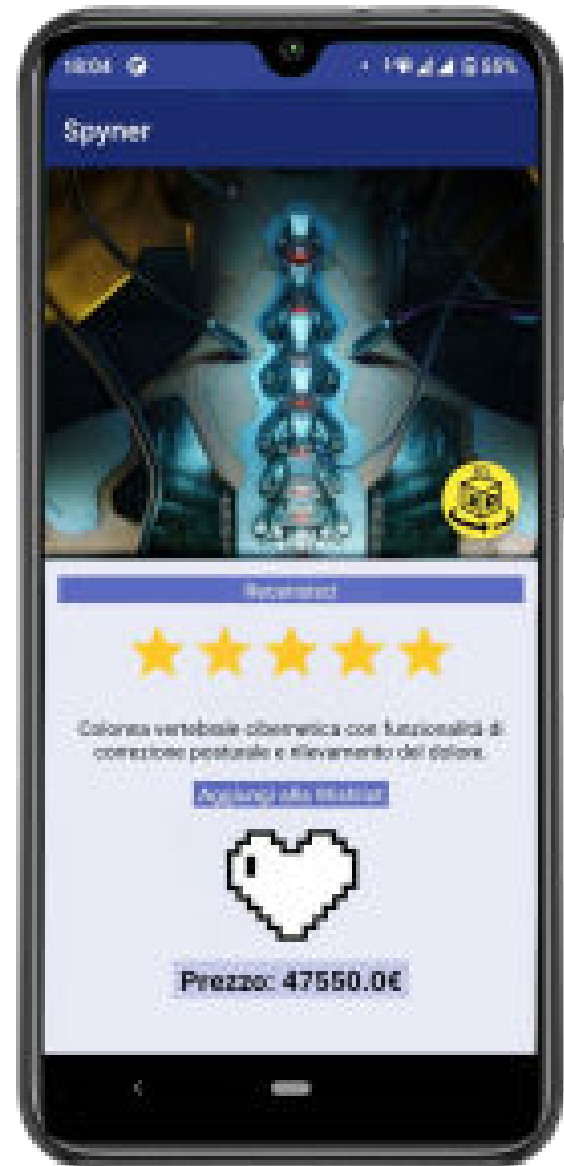
(Full implementation  
in the documentation for the  
FullProductActivity class)



# FullProductActivity

Add the product to the wishlist through **buttonWishlist.setOnClickListener** checking whether the product is already in the wishlist and consequently the associated “heart” icon. 

(full implementation in the FullProductActivity class documentation)



# FullProductActivity

**Here the Databases involved are:**

[*FirestoreAuth*]

To get the uid of the logged in user.

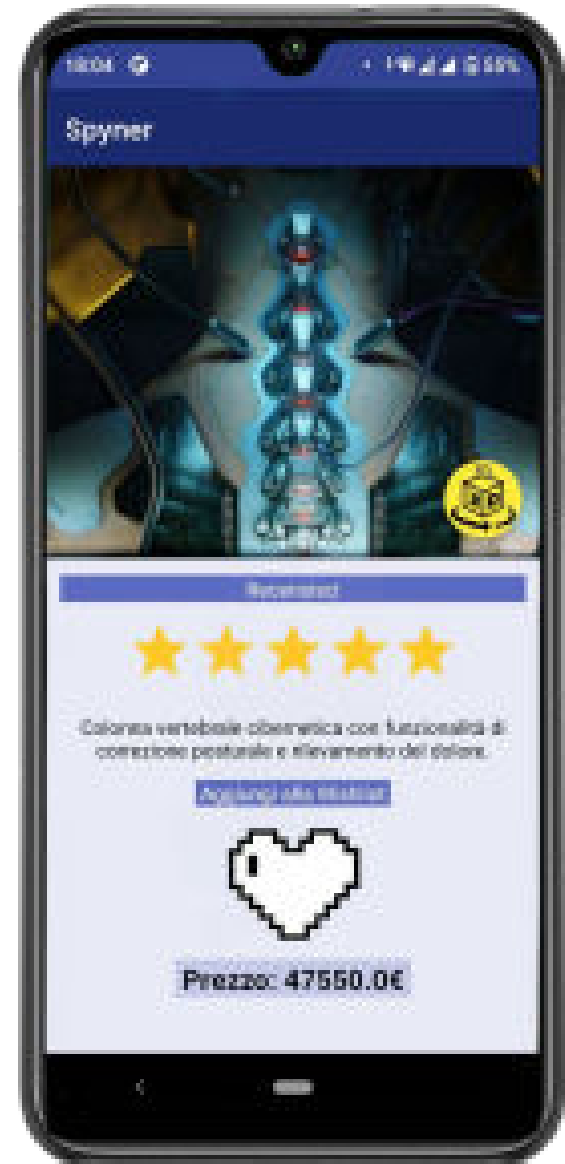
[*Realtime Database*]

To view the product and to add/remove it from the logged in user's wishlist.

To maintain the lookup table in the path “user\_wish/(UID\_PROD)/(UID\_USER)”

[*Storage*]

To display the product image and, if applicable, the STL to be displayed with Glide.



# ProfileActivity

Extracts the list of products in the wishlist from the database of the user and inserts it into the recyclerView where it shows one by one the products in the form of cards

(long\_card.xml)

Implement and set up a listener for each card that sends the user and product id clicked to [FullProductActivity] so you can view it in full detail.

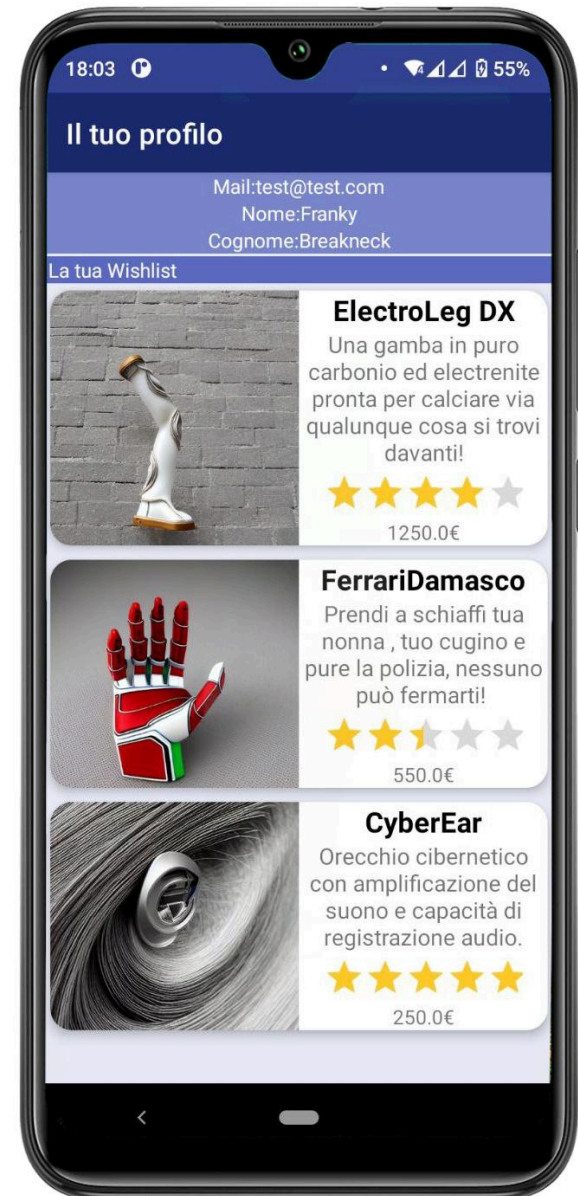
## Databases involved

[FirebaseAuth]

View and print: name, surname and email.

[Realtime Database]

Read the list of products from "users/(UID\_USER)/wishlist/product" and displays them on the screen.





# DashboardActivity

This is the home page for the admin which displays:

Administrator's data: name, surname, and email.

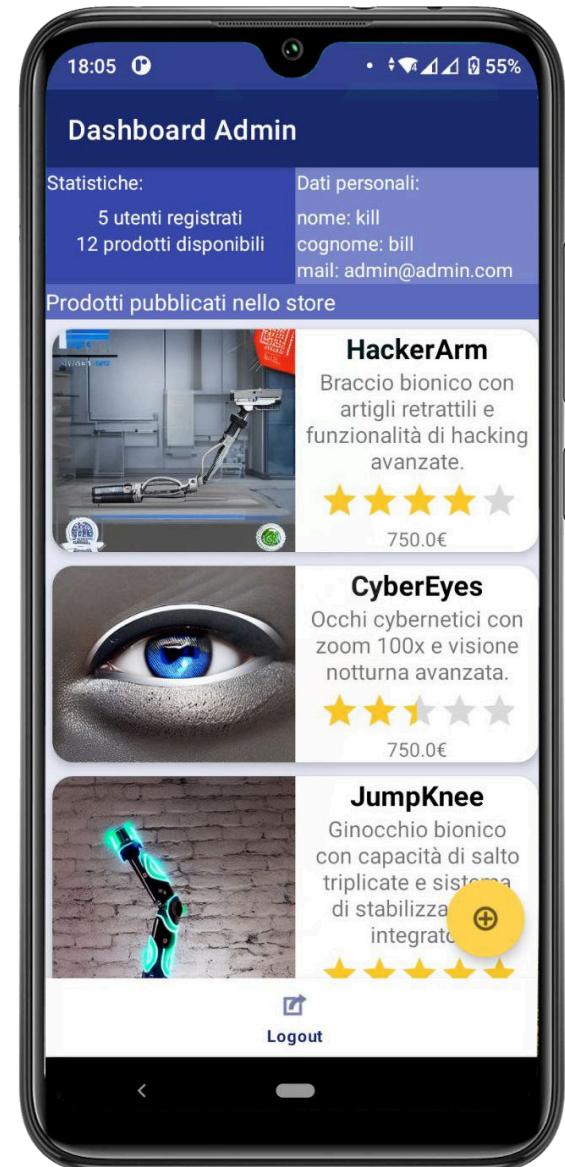
Store statistics: number of products in the store and number of registered users.

From here you can manipulate the shop database and products to:

1) Modify or delete a product by clicking on an item in the product list which, once clicked, redirects to `EditProductActivity` [with the product id to edit]

2) add a product to the store by clicking on the **fab (+)** floating above the store's product list, being redirected to `NewProductActivity` .

Clicking the navigation bar at the bottom logs you out.



# DashboardActivity

## Databases involved

*[FirebaseAuth]*

The user's privileges are checked on the page

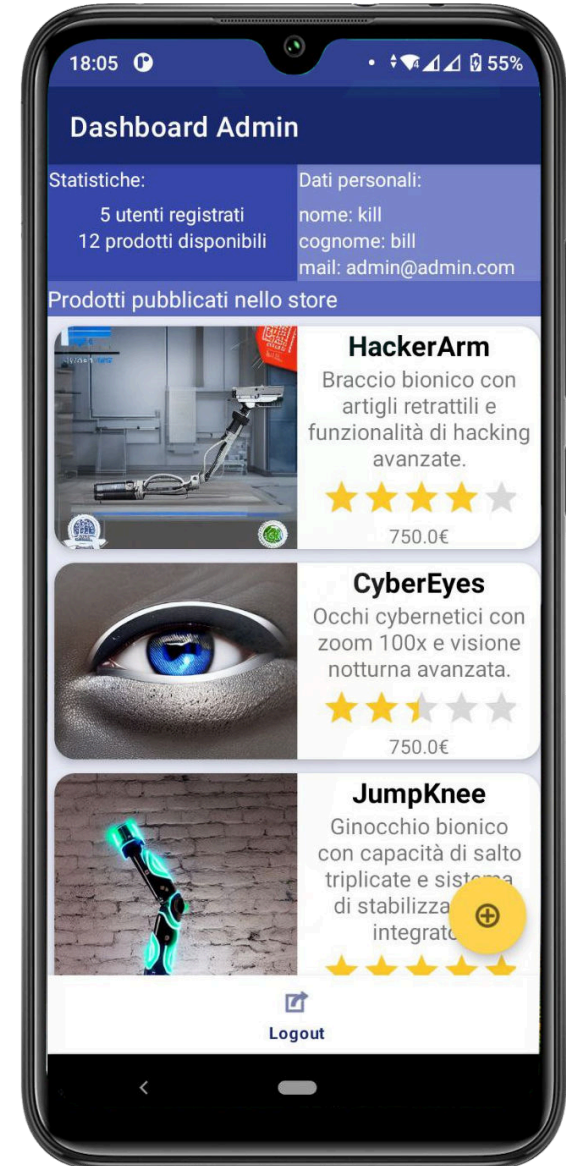
*[Realtime Database]*

The products are displayed

The admin's first name, last name, and email address are displayed.

The number of registered users is counted.

The number of products in the store is counted.



# NewProductActivity

After clicking on the fab **(+)** in the activity dashboard this activity opens, where you can add:

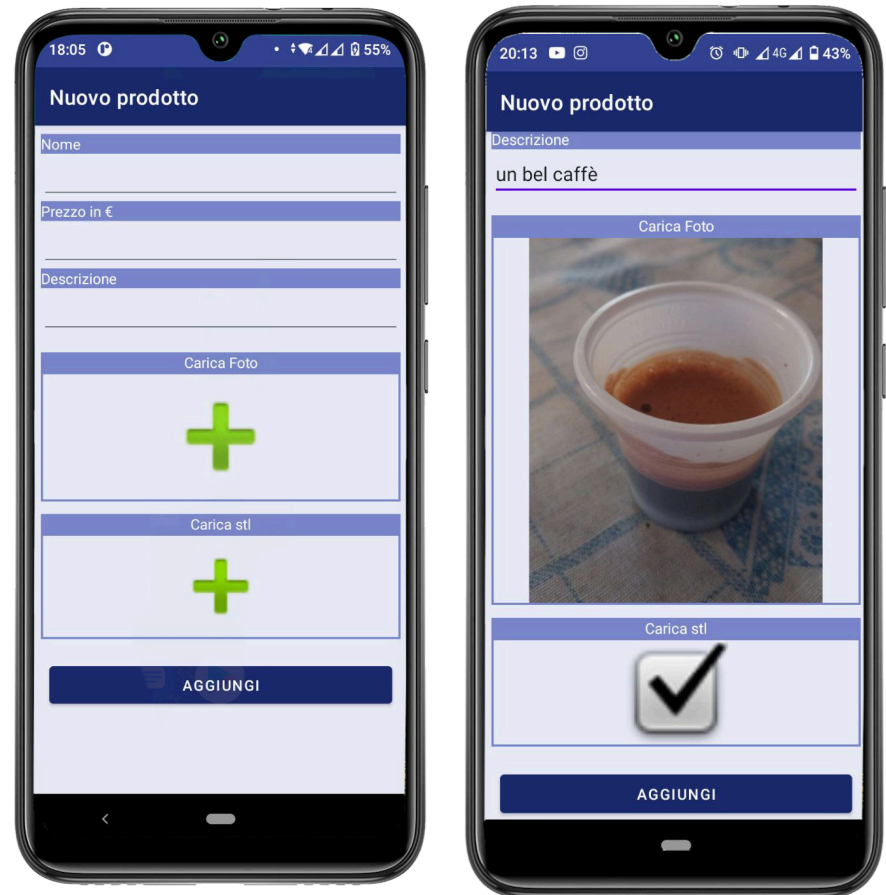
**name**,  
**description** ,  
**price** ,  
**image** of the product  
and the **file stl** of the product [which will go into Storage].

Under **Upload Photo** there is an ImageButton **previewImage** with a listener that invokes **selectImage()** to choose the image from the device memory that:

Set the file path locally to **mImageUri** through a **onActivityResult** and that sets the image in the **previewImage**

Under **Upload stl** there is an ImageButton **previewStl** with a listener that invokes **fileChooser()** to choose the stl file from the device memory that:

Set the file path locally to **mStlUs** through **onActivityResult** and what sect **[✓]** in the **previewStl**



# NewProductActivity

At the bottom it is present **ADD** button with a listener set that once clicked invokes **addNewItemToDatabase()**  
That :

Check that the data entered are all valid and not null by shifting the focus of the user if some field is not present and finally contact the db in order to:  
insert it into the database [*Realtime Database*] the new product with references **mStorageRef** and **mStorageRef2** and to insert the image and stl into [*Storage*].

If the activity is destroyed because the phone is rotated through **onSaveInstanceState** they get recharged and checked **mStlUs** and **mImageUri** so you don't have to re-enter them.

If the product is inserted correctly you will be redirected to **DashboardActivity** with a confirmation toast.



# NewProductActivity

## Databases involved

[*Realtime Database*]

Contacted to add the new product

[*Storage*]

Contacted to add the image in

**gs://cybershop.../IMG\_STL/file.png**

and the Stl file in

**gs://cybershop.../FILE\_STL/file.stl**



# EditProductActivity

After clicking on a product in the list in the `DashboardActivity` this activity opens where it is possible to **modify** the product using the received product id:

**name, description, price, image** of the product and the **file stl** of the product [which will go into Storage]

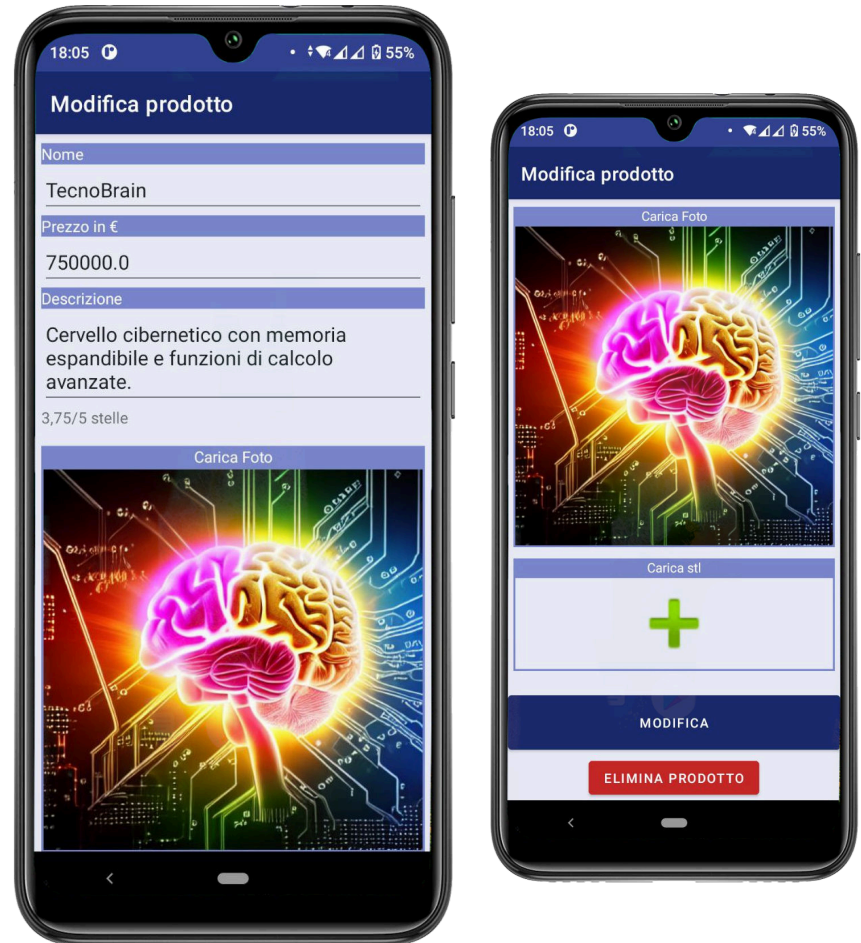
Fields are **precompiled** with the product data to be modified.

Under **Upload Photo** there is an ImageButton **previewImage** with a listener that invokes **selectImage()** to choose the image from the device memory that:

Set the file path locally to **mImageUri** through an **onActivityResult** and that sets the image in the **previewImage** replacing the previous reference

Under **Upload stl** there is an ImageButton **previewStl** with a listener that invokes **fileChooser()** to choose the stl file from the device memory that:

Set the file path locally to **mStlUs** via an **onActivityResult** and set [✓] in the **previewStl** replacing the previous reference





# EditProductActivity

It's present an **EDIT** button with a listener set that once clicked invokes **modItemToDatabase()** That :

Check that all the data entered is valid, shifting the user focus if any field is missing, and contact the db to insert into the database [*Realtime Database*] the modified product keeping the id with the references **mStorageRef** and **mStorageRef2** to insert the image and stl into [*Storage*] in case of modification.

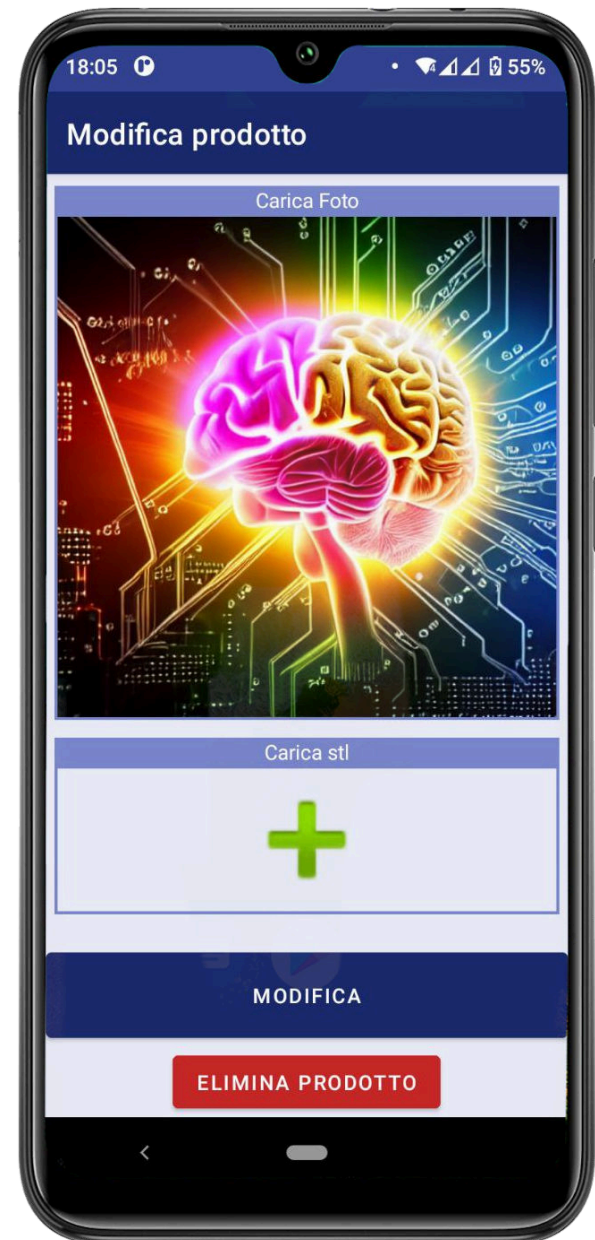
Obviously, when changing the stl and/or image files, the old files in storage will be deleted.

To maintain a tight consistency of cases and optimize access to [*Storage*] during editing there are 4 possible cases

1. Don't change either stl or png
2. Change only the stl
3. Change only the png
4. Change both

If no data is changed, the initial product data will be used.

If the product is modified successfully, user will be redirected to DashboardActivity with a confirmation toast.



# EditProductActivity

It's present a **DELETE PRODUCT** button with a listener set that once clicked invokes **deleteItemToDatabase(id)** That :

will query the database for the product id and delete it from the entry in [Realtime Database] and its associated files in [Storage]

## Databases involved

[Realtime Database]

Query the database to read, modify and delete the product

[Storage]

is contacted for:

uploading the product image to edit

delete the stl and png file to replace

add image in **gs://cybershop.../IMG\_STL/nome.png**  
and the Stl file in **gs://cybershop.../FILE\_STL/nome.stl**





# Style and colors

For the colors it was chosen the material palette 50,100,200 of a shade of purple.

The naming convention was followed:

**NameActivity -> activity\_name.xml**

Each Activity works both vertically and horizontally without content overflow thanks to careful layout design.

So it's fully responsive.

Thank you for your attention.

All code, resources , android studio project and documentation are present inside the github repository.