Epicture

The goal of the project is to create a social application implementing Imgur API, a Picture social Network.

Mockup

SplashScreen

Login

A WebView is opened to log the user to the Imgur Service

The User can connect with several possibilities :

- Facebook
- Twitter
- Google
- Yahoo

DiscoverList

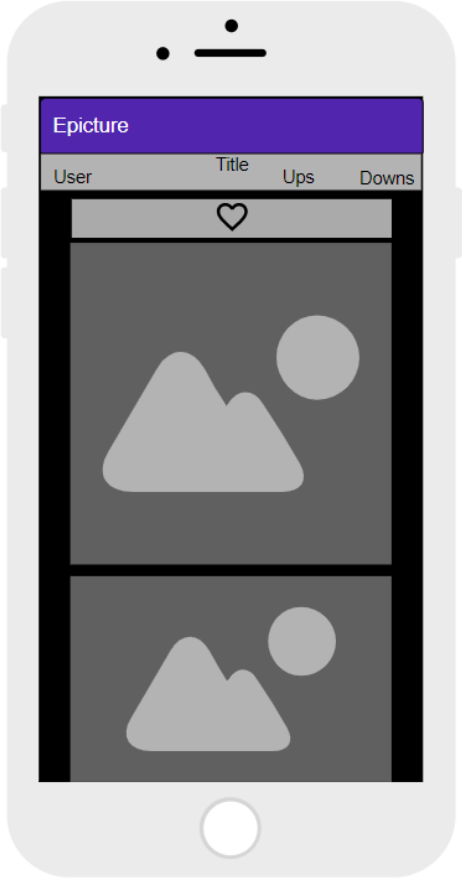
To begin the explanation, there is a navigation bar in the bottom side of the application to switch of pages easily

In the Discover page, the user can scroll as far as he wants to go.

Severals gallery are displayed with an image, the title, the ups and downs vote.

It's possible to click on it to get the gallery's details.

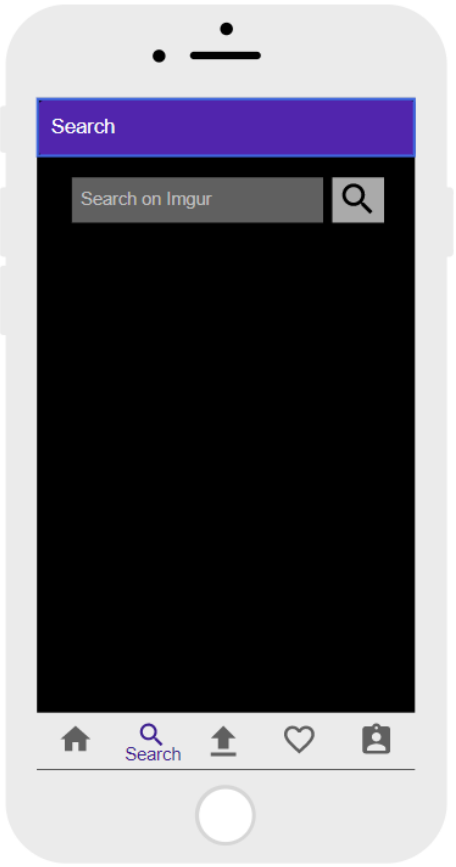
Gallery Details



Here the user will still be able to see the title, ups and downs vote but he will also see all images of the selected gallery and the who posted it.

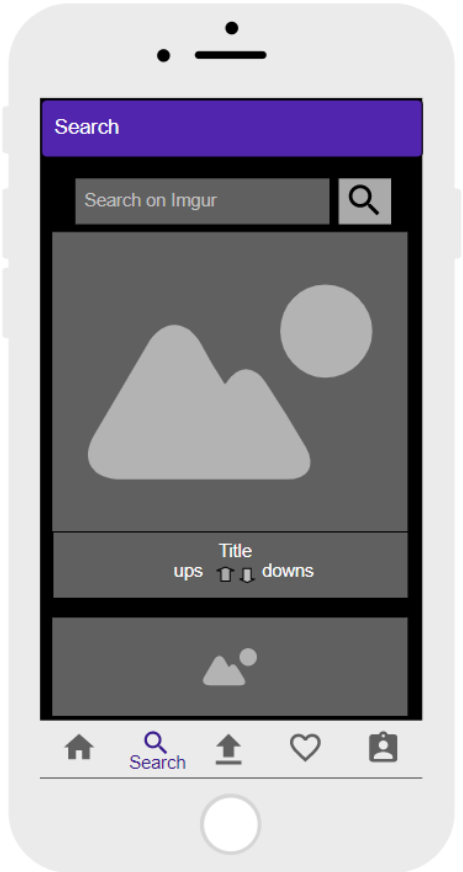
The user can click on the favorite button with a heart on the top of the page to add the gallery in his favorites

Search



Here is the search page,

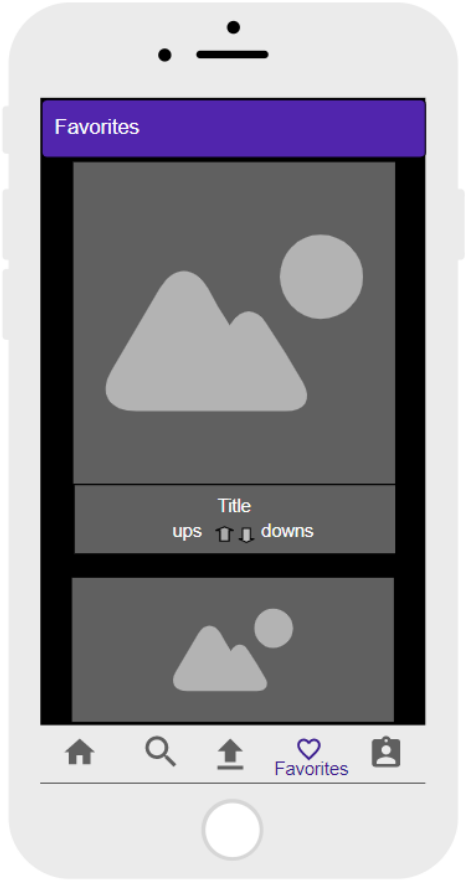
The user can write in the search bar then he can click on the button to launch the searching task



When it's done, severals galleries that correspond to the research will appear,

Same as for the discovery part, the user will be able to click on every gallery to see their details

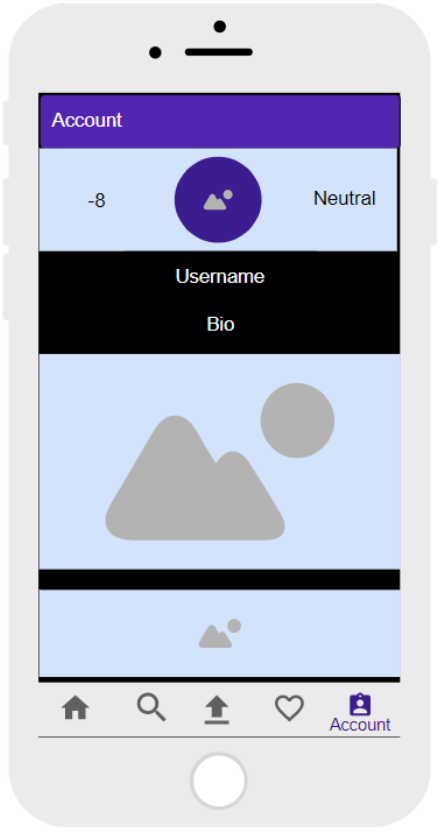
Favorites



This is the page where every favorites gallery of the user are displayed

The user can also click on the galleries

Account



To end the tutorial of the application, we will speak about the account page.

On this page are displayed information about the user. (The cover image, the avatar, the username, it's biography and notoriety)

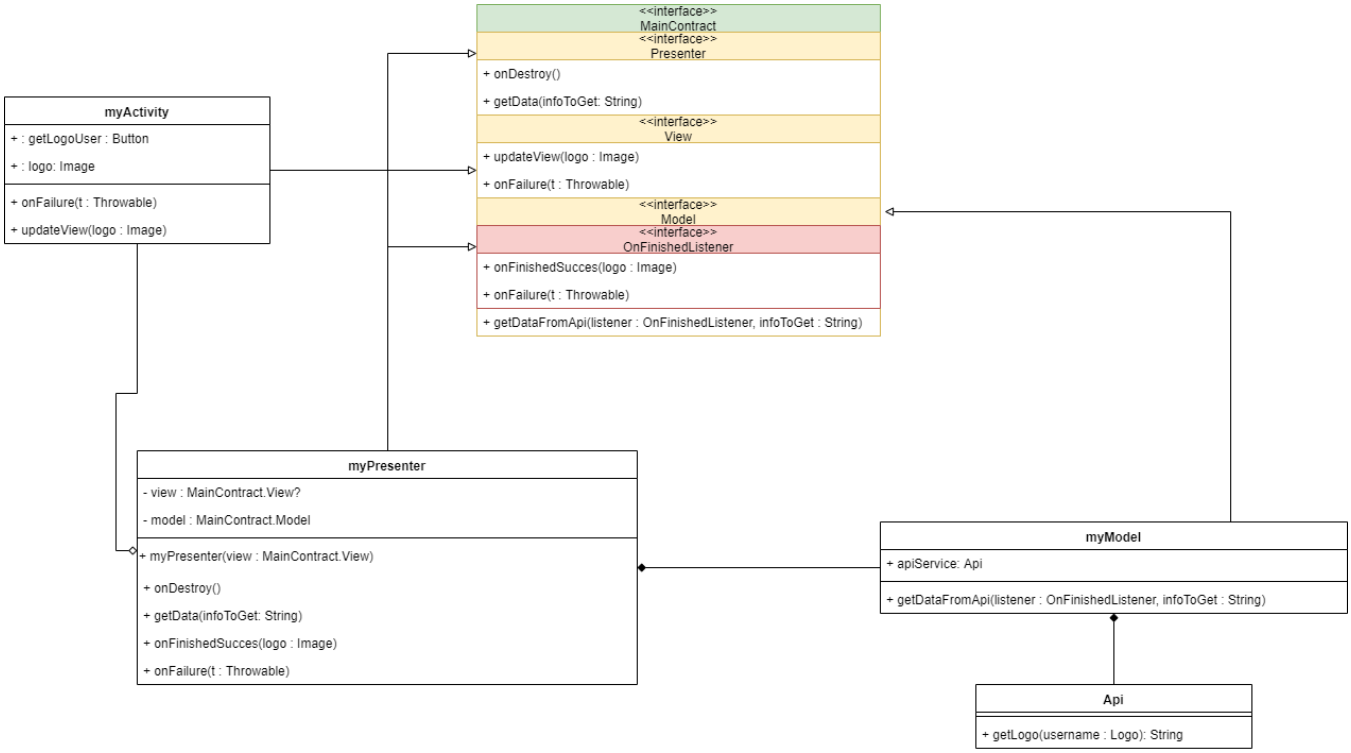
And bellow the information, are displayed images that the user posted on imgur

Architecture

The project use a MVP architecture, a Model View Presenter.

Each activity is defined by a base interface : The contract. Inside this, we define the several functions of the activities like the display, the getter of data from the api. They are divided in three, the Model Interface, View interface and Presenter interface.

Here is a base MVP architecture to show you how it's done :



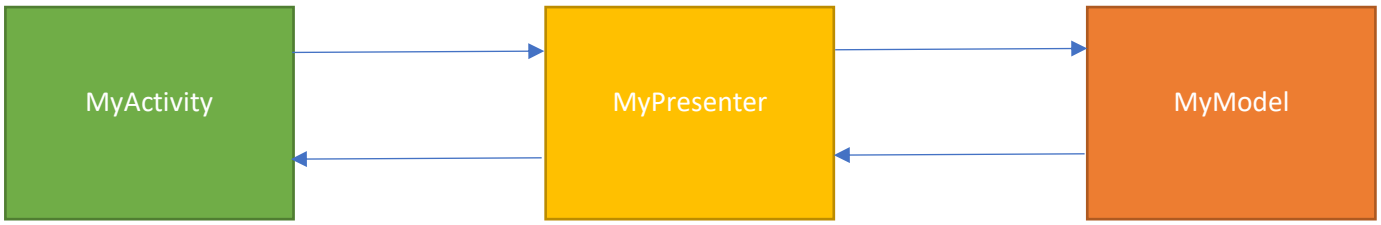
The model contains all the network api implantation, in this case it's where the data is fetch.

The View contains all the graphical aspect of the activity, for exemple our Application aspect.

The Presenter is the intermediate between our View and our Model, the view and the model cannot communicate between each other. The presenter will indicate to the model wich data to get and wich aspects to update for the view.

Here, the OnFinishedListener interface is the CallBack part for the network abstraction.

During the execution, all the data are transfered in this cycle :



For the project, each activity is separate in module where we have implemented a MVP abstraction.

Api Imgur Implementation

The api is integrated with the Retrofit2 Library

Gallery :

```
@GET( value: "gallery/{section}/{sort}/{window}/{page}")
fun gallery(@Path( value: "section") section : String,
            @Path( value: "sort") sort : String,
            @Path( value: "window") window : String,
            @Path( value: "page") pageNo : Int,
            @Query( value: "showViral") showViral : String = "true",
            @Query( value: "mature") showMature : String = "true",
            @Query( value: "album_previews") albumPreviews : String = "false",
            @Header( value: "Authorization") idClient : String = "Client-ID " + Constants.clientID
            ) : Call<ResponseApi<List<Gallery>>>
```

The gallery request returns a list of gallery with some filters :

- Section : **hot** | **top** | **user** by defautlt **hot**
- Sort : **viral | top | time | rising** by default **viral**
- Page : the **number** of the page wanted

AccountBase :

```
@GET( value: "account/{username}")
fun accountBase(@Path( value: "username") username : String,
                @Header( value: "Authorization") idClient : String = "Client-ID " + Constants.clientID
                ) : Call<ResponseApi<User>>
```

The accountBase request returns the User's information like his score, coverImage, avatar and more, just need to indicate the username in the request

GetAccountImages :

```
@GET( value: "account/me/images")
fun getAccountImages(@Header( value: "Authorization") bearer : String = "Bearer " + Constants.accessToken
                    ) : Call<ResponseApi<List<Image>>>
```

Returns a list of post of the connected user.

AccountGalleryFavorites :

```
@GET( value: "account/{username}/gallery_favorites/{page}/{favoritesSort}")
fun accountGalleryFavorites(@Path( value: "username") username: String,
                            @Path( value: "page") page : Int = 0,
                            @Path( value: "favoritesSort") favoritesSort : String = "newest",
                            @Header( value: "Authorization") idClient : String = "Client-ID " + Constants.clientID
                            ) : Call<ResponseApi<List<Gallery>>>
```

Returns a list of the favorites post of a specific user :

- Username : the user to check
- Page : the **number** of the page wanted

GallerySearch :

```
@GET( value: "gallery/search/{sort}/{window}/{page}")
fun gallerySearch(
                  @Path( value: "page") pageNo : Int,
                  @Path( value: "window") window : String,
                  @Path( value: "sort") sort : String,
                  @Query( value: "q") toSearch : String,
                  @Header( value: "Authorization") idClient : String = "Client-ID " + Constants.clientID
                ) : Call<ResponseApi<List<Gallery>>>
```

It creates a research request and returns a list gallery specific to the request.

- Page : the **number** of the page wanted
- Window : **day | week | mounth | year | all** default to **all**
- toSearch : the key-word for the research

FavoriteAlbum :

```
@POST( value: "album/{albumHash}/favorite")
fun favoriteAlbum(@Path( value: "albumHash") idGallery : String,
                  @Header( value: "Authorization") bearer : String = "Bearer " + Constants.accessToken
                ) : Call<ResponseApi<String>>
```

Put a gallery in the user's favorite. The user needs to be connected.

Tests

Android Studio

To launch the test, open the project with android studio. Then go app/java/com.example.epicture/ and right click on the OAuthUrlUnitTest and Run the file

Externals Dependencies

The project use several external librairies :

- Retrofit2 : An easier way to integrate API inside our code
    - https://square.github.io/retrofit/

- Glide : A dynamic image printer
    - https://github.com/bumptech/glide

- Gson : A json parser
    - https://github.com/google/gson