

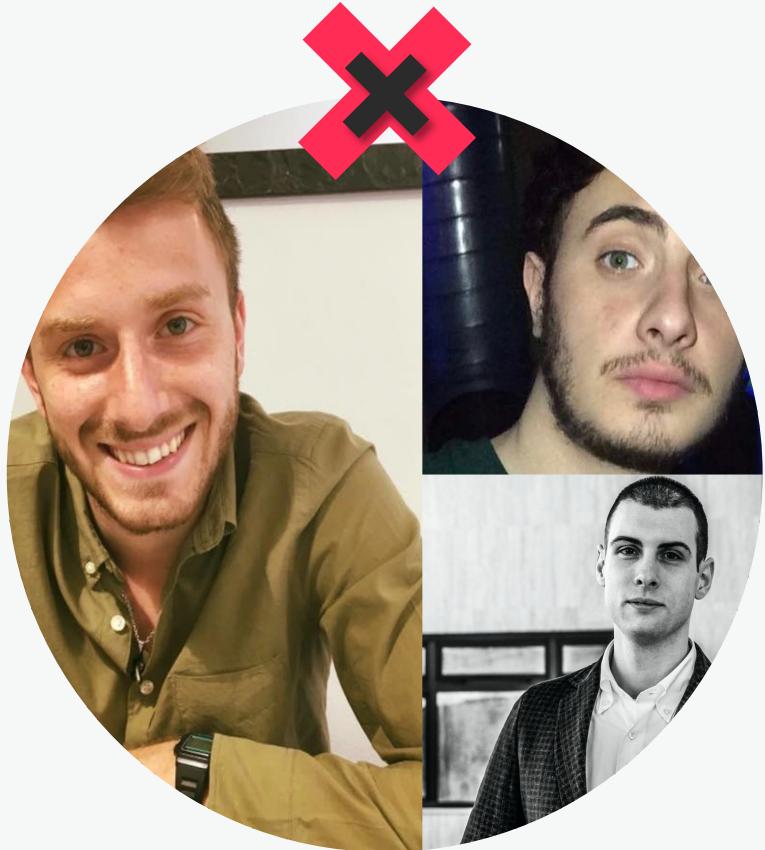
# Hoodie Music Downloader



SAPIENZA  
UNIVERSITÀ DI ROMA

---

*Mobile Application and Cloud Computing 2019/2020*



- [ Describing our Team ] -

## Our Team

Luca Tomei 1759275

Danilo Marzilli 1878501

Giovanni Trinca 1542534

# OUTLINE OF **TALK**

- 
- Introduction
  - Structure
  - Design
  - Look and Feel
  - Conclusion
  - Future Works

Hoodie is the best way to easily download and listen to songs:

- Stream
- Download
- Unlimited library
- Portrait and horizontal mode
- Trending music by genre → Charts



# MAIN FEATURES





- [ AN AGILE PROCESS ] -

## Our Approach

In order to develop the application correctly we decided to use a **SCRUM approach**, proceeding by sprints.



# Used Technologies

## Backend



### Node.js

Asynchronous event-driven  
Javascript runtime.

We use Node.js to **download**  
songs directly from Deezer.



### Firebase

Backend as a Service  
(Baas).

We store our **users**, their  
**searches**, all the stuff needed  
by our app.

## Frontend

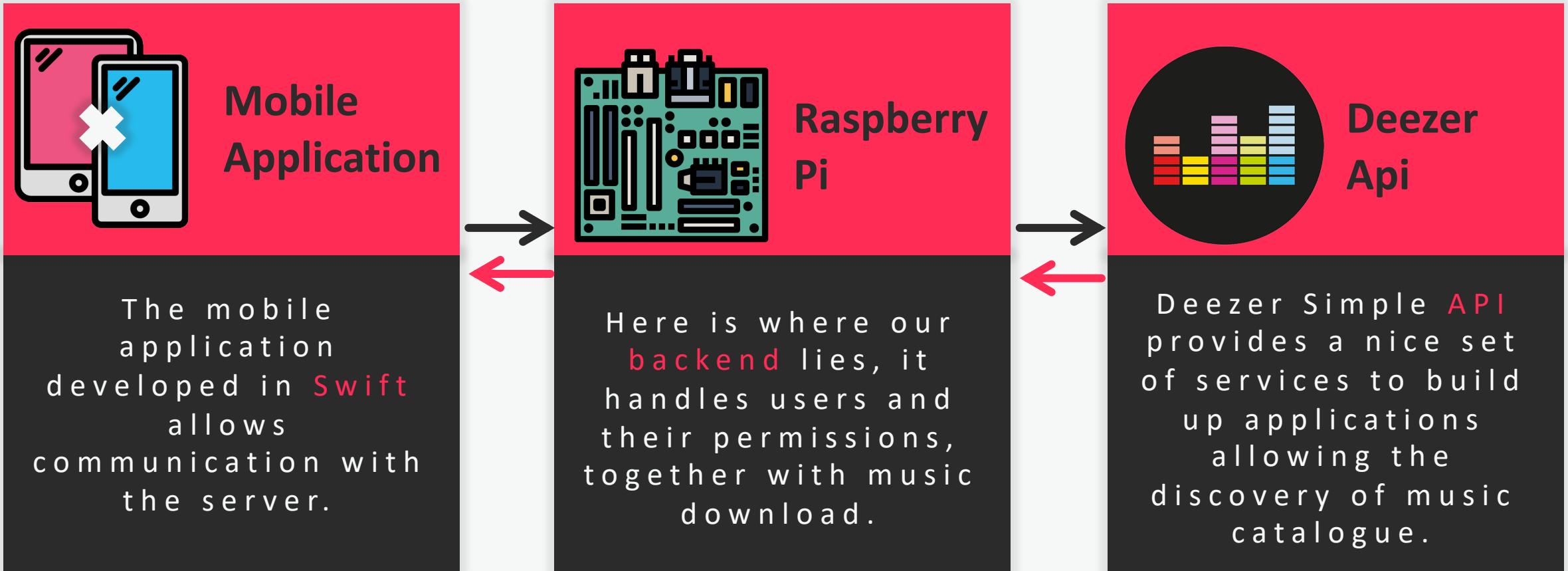


### Swift

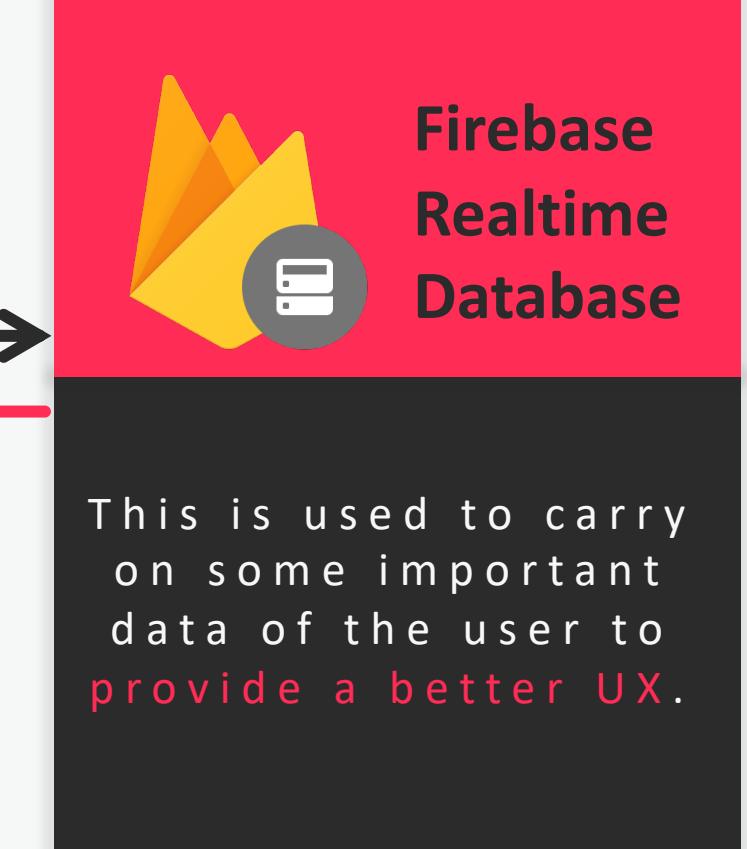
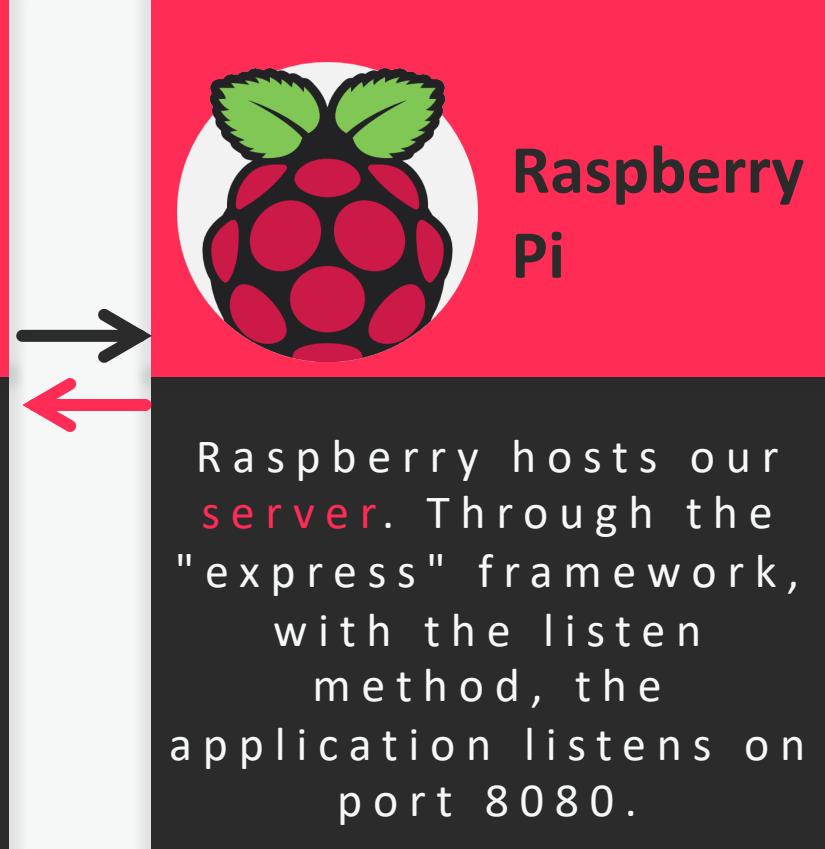
Powerful and intuitive  
**programming language** for macOS,  
iOS, watchOS and tvOS that  
support both **object-oriented** and  
**functional** programming.



# Application Architecture



# Backend Structure



# FIREBASE AUTHENTICATION



```
import FirebaseAuth

func signUp(email: String, password: String, completionBlock: @escaping (_ success: Bool) -> Void) {
    Auth.auth().createUser(withEmail: email, password: password)
{ (authResult, error) in
    if let user = authResult?.user {
        print(user)
        completionBlock(true)
    } else {
        completionBlock(false)
    }
}
```



```
func signIn(email: String, password: String, completion: @escaping (_ error: Error?) -> Void) {
    Auth.auth().signIn(withEmail: email, password: password)
{ authResult, error in
    completion(error)
}
```



```
func sendPasswordReset(withEmail email: String, _ callback: ((Error?) -> ())? = nil){
    Auth.auth().sendPasswordReset(withEmail: email) { error in
        callback?(error)
    }
}
```

- We use Firebase Authentication to allow users to log into our app using their personal **email and password**.
- These credentials will be passed to the Firebase Authentication SDK and after getting a response from their servers, the user's data will be saved to Firebase Realtime Database.



# GOOGLE AUTHENTICATION

```
import GoogleSignIn

func GoogleSign(_ signIn: GIDSignIn!, didSignInFor user: GIDGoogleUser!, withError error: Error!) {
    if let error = error {
        if (error as NSError).code == GIDSignInErrorCode.hasNoAuthInKeychain.rawValue {
            print("The user has not signed in before or they have since signed out.")
        } else {
            print("\(error.localizedDescription)")
        }
        return
    }

    // Save User Data
    let userId = user.userID
    let idToken = user.authentication.idToken
    let fullName = user.profile.name
    let givenName = user.profile.givenName
    let familyName = user.profile.familyName
    let email = user.profile.email

    // Store user data ...
    //...
}
```

```
func GoogleLogout(){
    do {try GIDSignIn.sharedInstance()?.signOut()}
    catch { print("already logged out") }
}
```

- Google Sign-In manages the **OAuth 2.0** flow and token lifecycle, simplifying our integration with Google APIs.

- OAuth 2.0 works with the following **four actors**:

- **authorization server:** responsible for authentication and authorization.
- **resource server:** in charge of serving up resources if a valid token is provided.
- **resource owner:** the owner of the data, that is, the end user of Hoodie.
- **client:** the Hoodie mobile app



# FACEBOOK AUTHENTICATION



```
import FBSDKCoreKit
import FBSDKLoginKit

@IBAction func didPressFacebookSignin(_ sender: Any) {
    let loginManager=LoginManager()
    loginManager.logIn(permissions: ["public_profile", "email"],
viewController : self) { loginResult in
        switch loginResult {
        case .failed(let error):
            print(error)
        case .cancelled:
            print("User cancelled login")
        case .success(let grantedPermissions, let
declinedPermissions, let accessToken):
            print("Logged in")
            self.goToMainView(message: "")
        }
    }
}
```

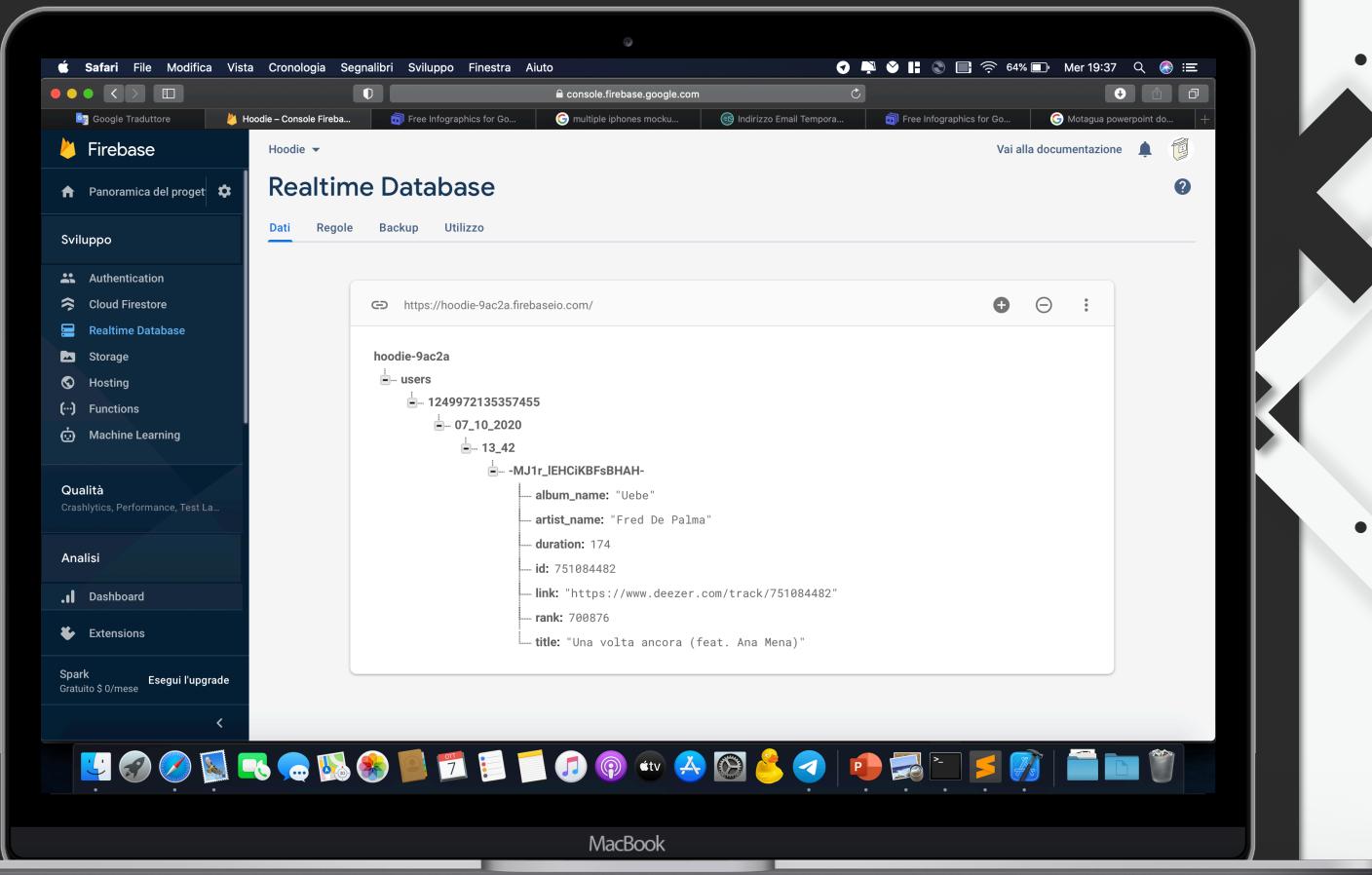


```
func facebookLogout(){
    do {try LoginManager().logOut()}
    catch { print("already logged out") }
}
```

- With the rise of social media, **Facebook login** integration has become one of the must have features in mobile apps. Despite the fact that every developer is integrating Facebook login into their apps, Facebook is doing a very poor job on updating their documentations.
- Using the Facebook authentication token we get from Swift, we can access the **Facebook API** and get the user's name and email to create an account.



# REALTIME DATABASE



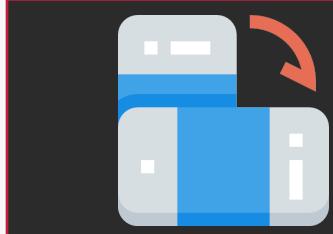
- The Firebase Realtime Database is a **NoSQL** cloud-hosted database. Data is stored as **JSON** and **synchronized** in real time to every connected client.
- All of our clients share one Realtime Database instance and automatically receive updates with the **newest data**.



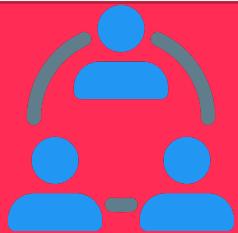
# iOS Application



Multi-Platform iOS Design



Responsive Layout



Multithreading

DispatchQueue + URLSession for https requests on separated thread  
(Codable protocol used to convert JSON to native Swift struct)

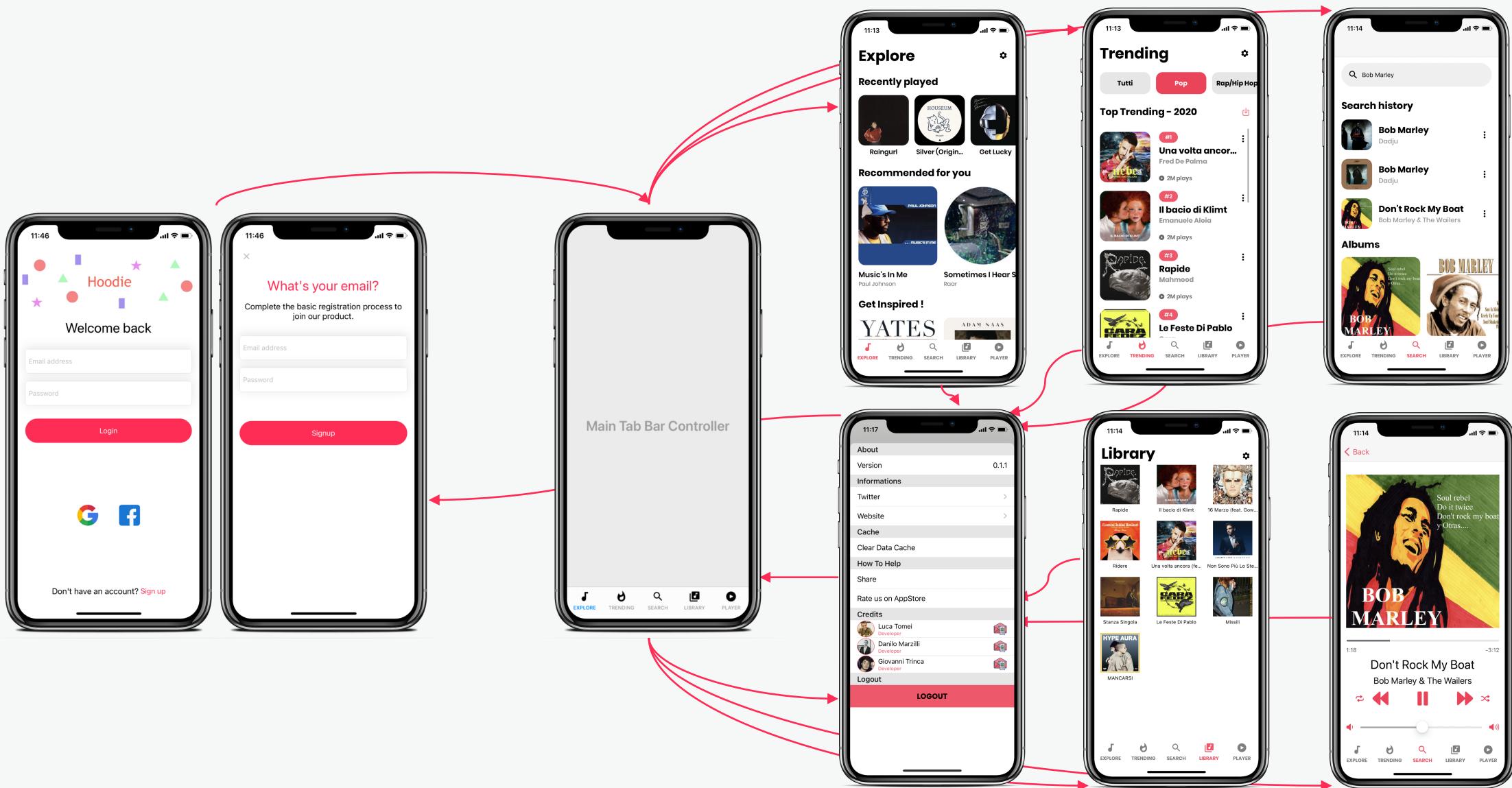
MVVM

Model View ViewModel Pattern Design

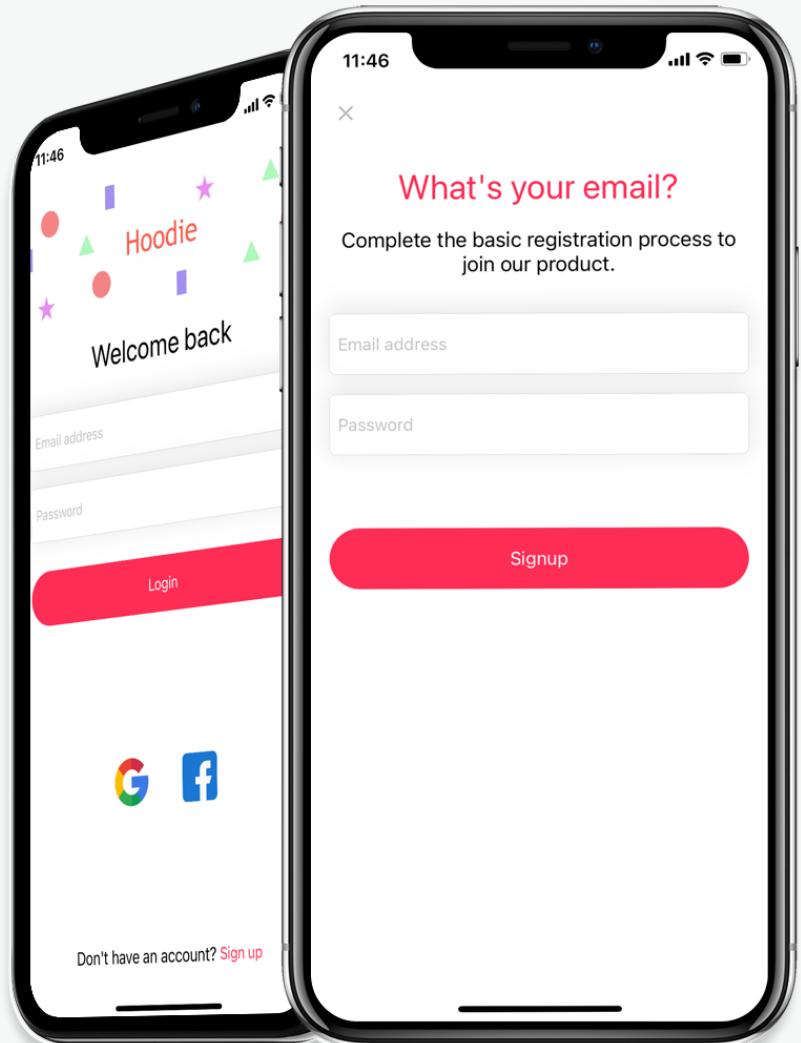


Design.

# Navigation Path



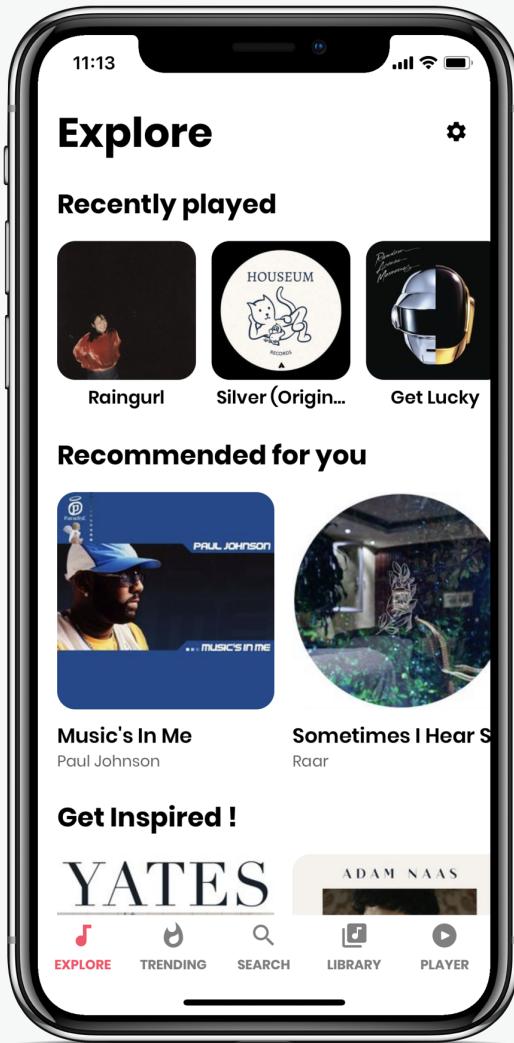
# Login & Signup



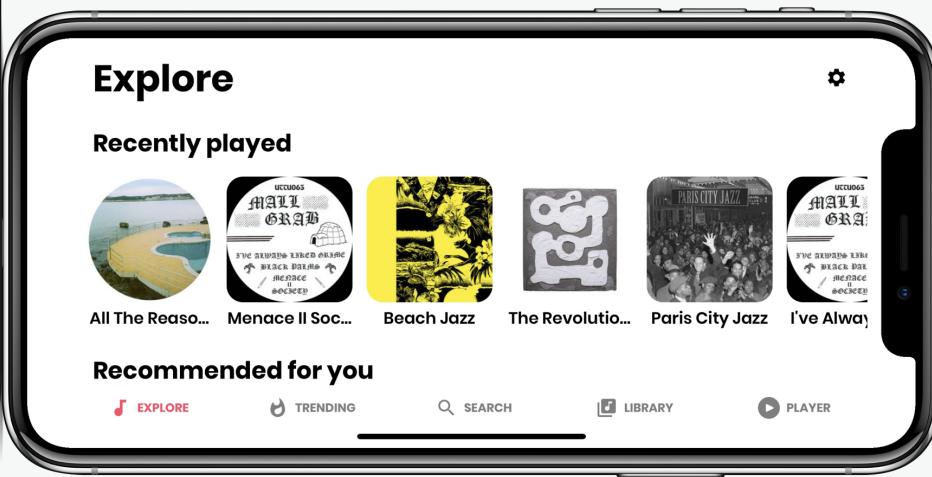
- Login methods supported:
  - Email and Password (Firebase Auth)
  - Google OAuth
  - Facebook
- After login or registration, the user is redirected to the main page of the application.



# Explore View

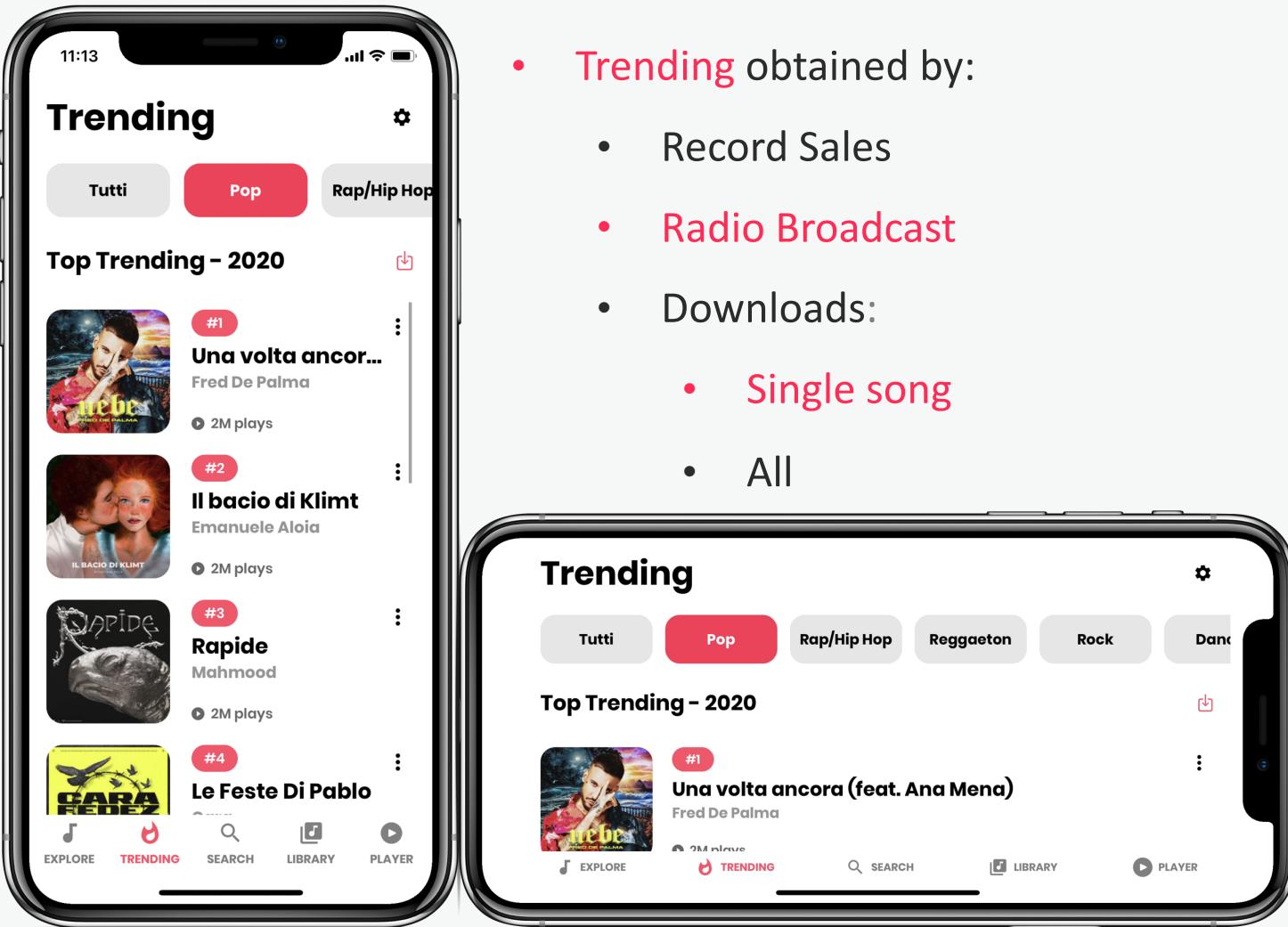


- Explore:
  - Recently listened songs
  - Related songs per genre



Look and Feel.

# Trending View

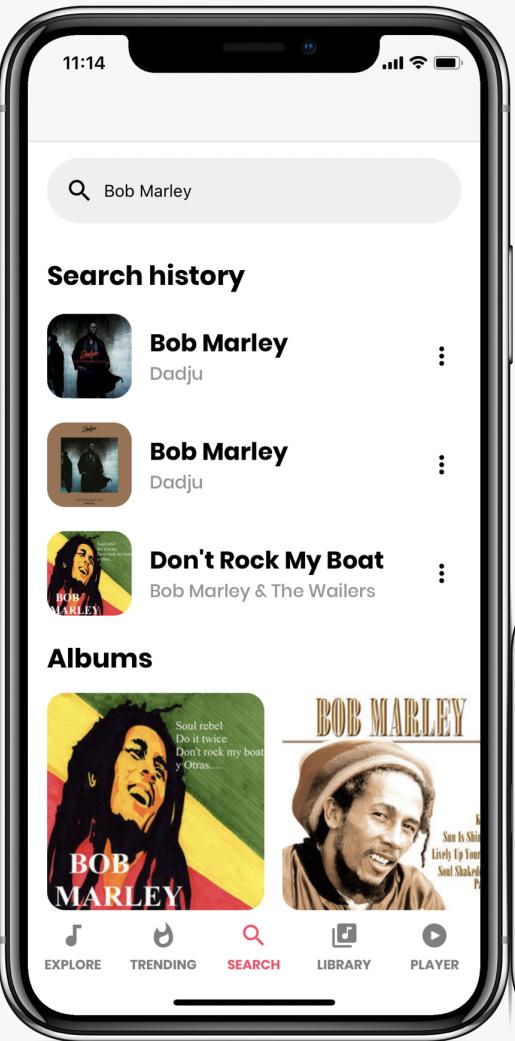


- Trending obtained by:

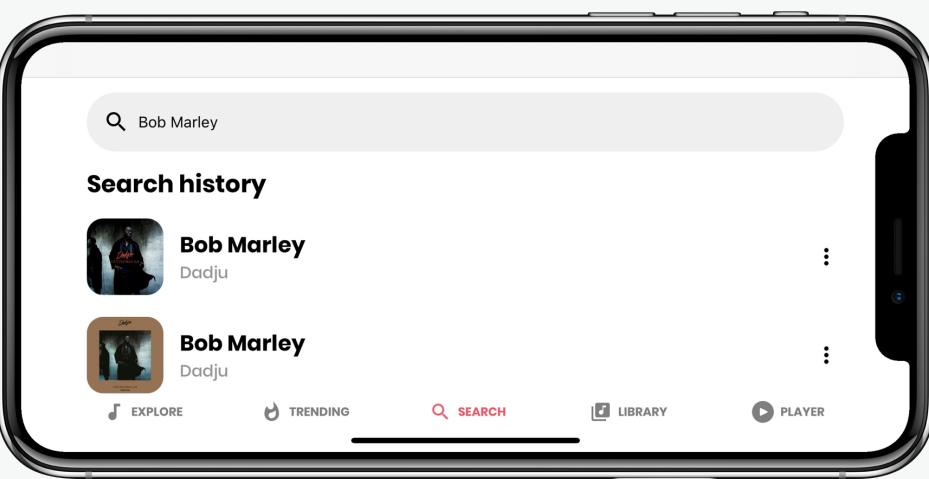
- Record Sales
- Radio Broadcast
- Downloads:
  - Single song
  - All



# Search View

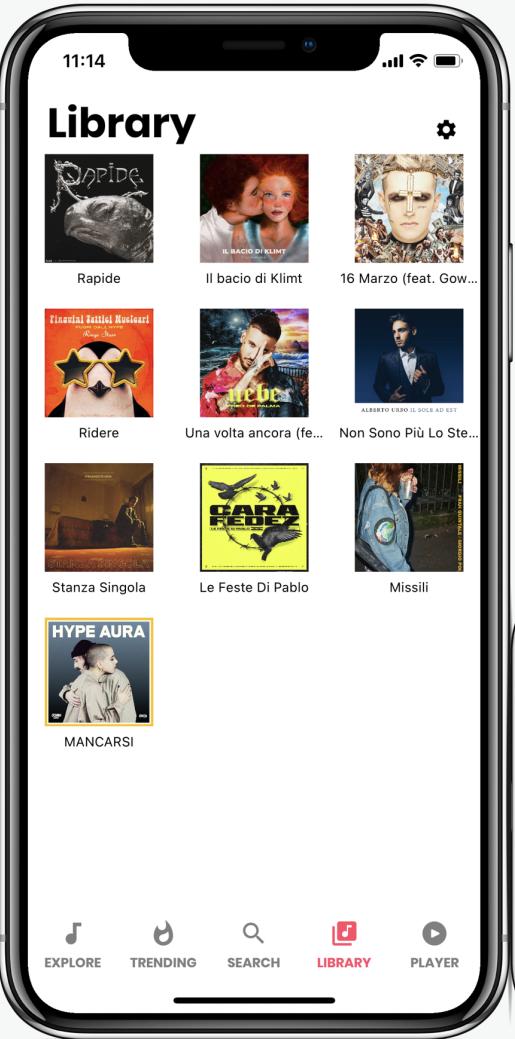


- Search (Deezer API):
  - Results by Ranking
  - Play a Song
  - Download a Song

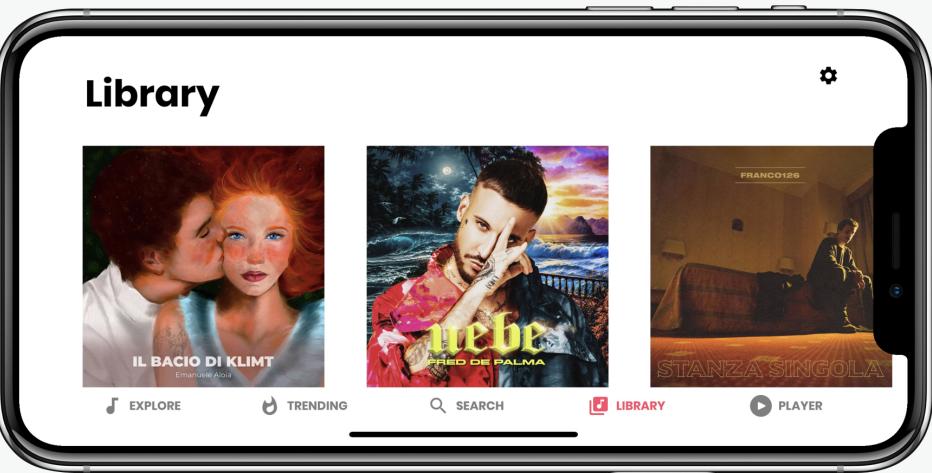


Look and Feel.

# Library View



- Library
  - Shows
    - Name of the song
    - Image of the album
  - User action
    - Play
    - Remove



Look and Feel.

# Player View

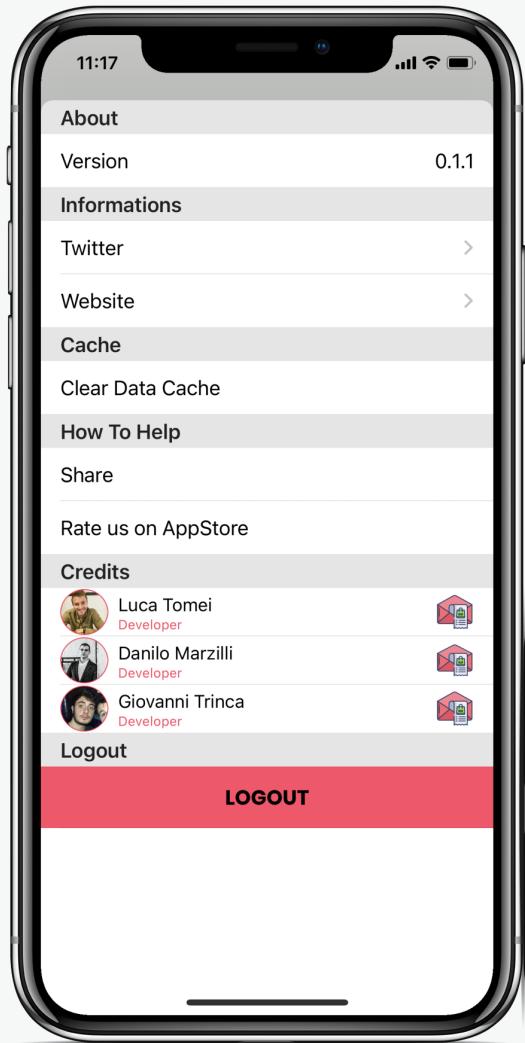


Our player shows the **music currently playing** accompanying the listening showing:

- The **name** of the album, the song, the author
- Cover **picture**
- Play/pause, next, previous, shuffle, repeat one, repeat all, volume slider **buttons**

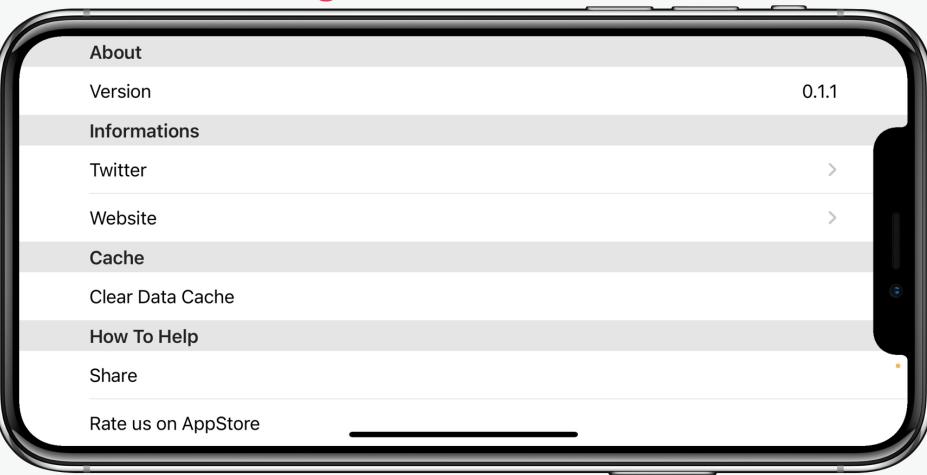


# Settings View



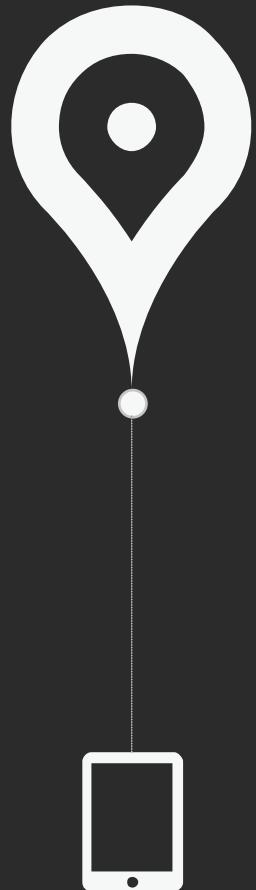
## Settings:

- Verify what **version** are you using
- See app **information** via Twitter or Website or email
- Clear **Data Cache**
- **Share** our application
- **Rate us** on AppStore
- **Logout**

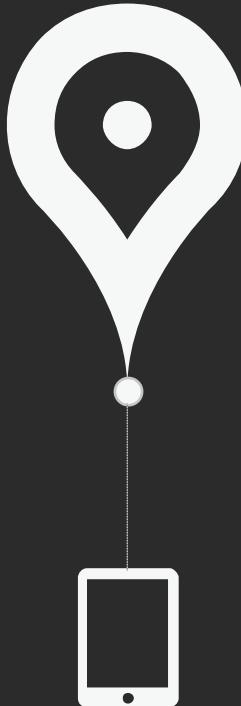


# Requirements

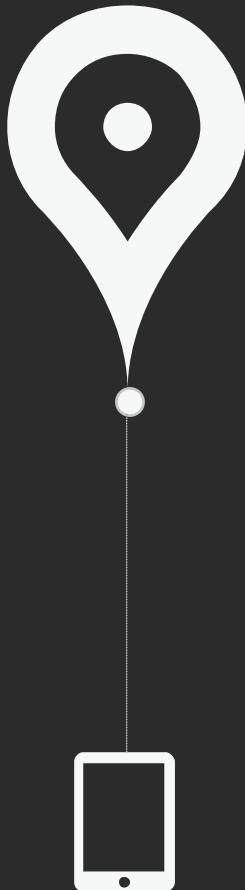
What we achieved



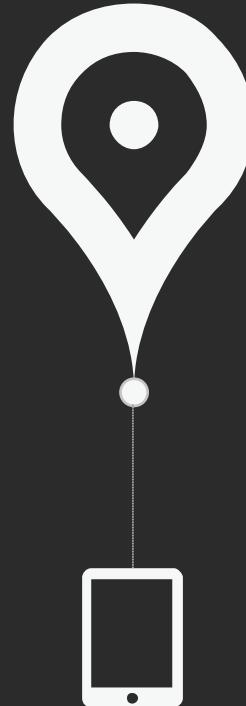
Responsive  
Design.



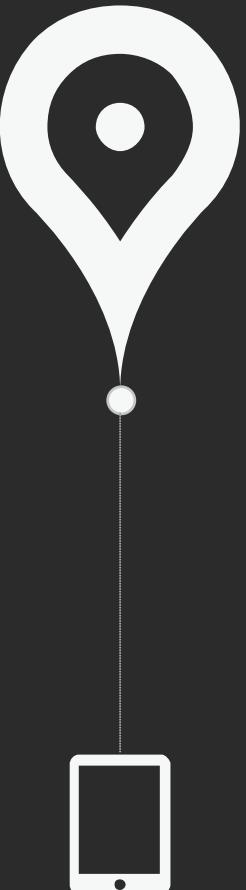
Auth.  
Service.



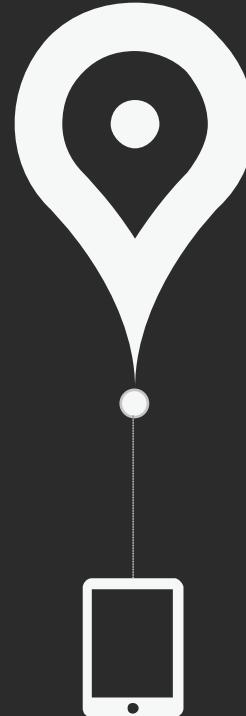
Multi  
Thread.



External  
WebAPI.



Storage  
Service.



Personal  
Backend.



Conclusion.



- [           WHAT NEXT?           ] -

## Future Works.

A possible extension of the application is to support dynamic playlists and self-created suggestions based on the musical preferences of the individual user through the use of the data saved on the Firebase Realtime Database.





- [ For your attention ] -

# Thank You

