!VOID LAB

fill the void.

Porto Oculto

Software Architecture Document

Revision: 0.6

Client: Coletivo Oitoo and OASRN

Date of Revision: 31.05.2019

Software Architecture Document

REVISION					
REV.	**DATE**	**DESCRIPTION**	**BY**	**REVISION**	**APPROVED BY**
0.1	25.02.2019	SAD initial version	Andre	Erica	Erica
0.2	15.03.2019	Android architecture details	Felipe	Erica	Erica
0.3	17.03.2019	Document formatting	Mila	Erica	Erica
0.4	22.03.2019	Inserted patterns and divided mobile architectures	Felipe	Erica	Erica
0.5	23.03.2019	Change after Company Internal Review	Mila	Erica	Erica
0.6	31.05.2019	Final review for validation	José	Erica	Mila

1. Introduction

1.1. Purpose of the Document

The main purpose of this document is to present to the client Coletivo Oitoo the description and details of the various components of the software architecture.

1.2. Definitions and Abbreviations

Initials	**Description**
API	Application Programming Interface
DB	Database
OS	Operating System
MVC	Model-View-Controller
MVVM	Model-View-ViewModel
OASRN	Ordem dos Arquitectos - Secção Regional Norte

Table 1 - Definitions and Abbreviations

2. Technical Solution

2.1. Execution Infrastructure

The software is a mobile application for Android and IOS devices, so the main infrastructure for the product will be user's own devices.

The version minimum version required of each platform will be as follow:

- Android: 7.0 (Android Nougat) – SDK version 24
- iOS: 10.0

The target version of each platform will be as follow:

- Android: 9.0 (Android Pie) – SDK version 28
- iOS: 10.0+

The support infrastructure, the one that will handle all data, images, maps, and authentication has more components, as described below.

2.2. Security

System will use Firebase Security rules to handle requests to the Cloud Firestore Database based on the user-level in the system.

All security relevant activities in the software are logged in the ***logs*** node in the database with the ID, TIMESTAMP, oldStatus and newStatus.

All setttings saved to the device should be encrypted with Base64 to avoid identification of sensitive data of the app.

Application code should be obfuscated to difficult reverse engineering of source code.

2.3. User Authentication

Firebase Authentication will handle the logins for both iOS and Android applications as it provides a secure and reliable service and SDK for both required platforms for this project.

There will be 3 types of users with different kinds of access:

- ○ Common users - This is the basic and standards account that allows the users to upload pictures of the buildings or fields they find abandoned/depreciated.
- ○ Premium(?) users - This is a payment account which allows the user to have access to more premium and detailed information.
- ○ Admins - This type of account has access to all functionality and is responsible for moderating certain aspects of the application.

2.4. Graphical Web Interface

Restrictions - The client had no restrictions regarding technologies.

Requests - The client asked for a simple, clean and minimalist app. The user should be able to use the app and submit pictures in a quick and simple manner.

2.5. Image Storage

In order to store the images taken by the users in the device we'll use Firebase Storage. It provides a cloud file storage designed to work with Android and IOS via the SDK Firebase provides. The services integrate easily with the authentication solution and its security policies available.

2.6. Handling of Exceptions

Exceptions in code will be handled using the failure callbacks for events in the Firebase SDK that contain the derived messages for each provider in case of sign-in and to the communication with the database.

Exceptions in business logic (activity flow) will be handled using informative screens, alert messages and screen dialogs.

2.7. Database

Firebase Cloud Firestore it's a NoSQL document-based database highly scalable and easily integrable with the authentication solution and provides SDK for both required platforms for this project.

3. Visions of Architecture

3.1 Layer view

The Porto Oculto project will be an application mainly for smartphones. For this, there will be developed two applications, one for iOS and one for Android. It will be natively developed for each one of these, iOS version will be developed in Swift 4.2 and the Android app is developed in Java 9. Like it was said previously, an initial version will use the firebase as the main database and the google maps API will serve as a base for the maps required in the app.

3.1.1. Presentation Layer

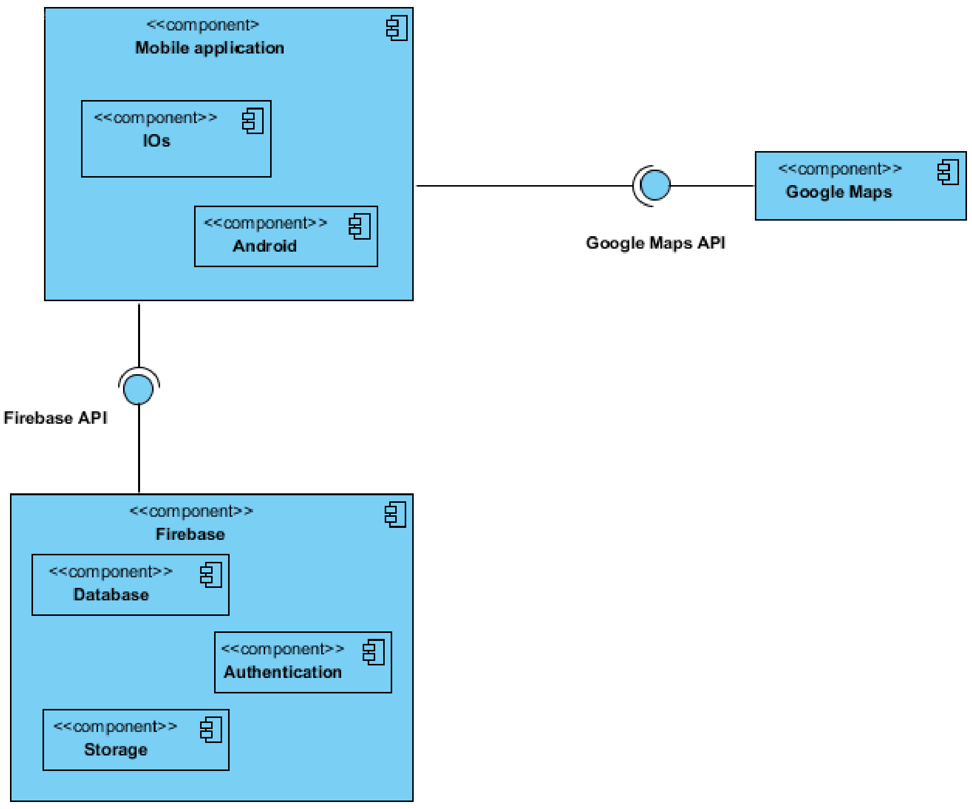

Diagram 1 - Component Diagram

3.1.2. View Layer

As a native mobile application, everything related to the view layer uses the native implementation of the UI components the platform provides, except for a few custom components that inherit from the native ones. This layer handles the interaction of the user with the application and communicates with the service layer to pass and receive information and update the UI accordingly.

3.1.3. Service Layer

The service layer is the middleman between the view and data layers. When a user interacts with the application, the service layer will receive this information and process and if necessary, request the required information from the data layer and then send the result to the view present it to the user.

3.1.4. Data Layer

The Firebase Cloud Firestore it's the central data storage. As a NoSQL document-based database, it gives us more freedom when to model the data structure necessary for the application. Firebase SDK for Android and iOS handles all communication with the database and implements the security features required to protect the application data.

3.2 Logical View

3.2.1. Process Overview iOS

The iOS mobile application will follow an architectural pattern known as Model-View-Controller (MVC). The MVC architectural pattern is used for developing user interfaces and divides an application into three interconnected patterns, this is done to separate internal representations of information from the ways information is presented to and accepted from the user. The MVC design pattern decouples these major components allowing for efficient code reuse and parallel development. The components have the following objectives:

- Model - The central component of the pattern. It is the application 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 a chart, diagram or table. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants.

- Controller - Accepts inputs and converts it to commands for the model or view.

The MVC not only defines the division of the application into these components but also defines the interactions between them.

- The model is responsible for managing the data of the application. It receives user input from the controller.
- The view defines a presentation of the model in a particular form.
- The controller responds to the user input and performs interactions on the data model objects. The controller receives the input, optionally validates it and then passes the input to the model.

3.2.2. Process Overview Android

The Android mobile application will follow an architectural pattern known as Model-View-ViewModel (MVVM). The MVVM architectural pattern is the recommended pattern to be used by Google and allows decoupling views from logic, making it easier to create unit tests and use observable objects.

- Model - This holds the data of the application. It cannot directly talk to the View. Generally, it's recommended to expose the data to the ViewModel through Observables.
- View - It represents the UI of the application devoid of any Application Logic. It observes the ViewModel.
- ViewModel - It acts as a link between the Model and the View. It's responsible for transforming the data from the Model. It provides data streams to the View. It also uses hooks or callbacks to update the View. It'll ask for the data from the Model.

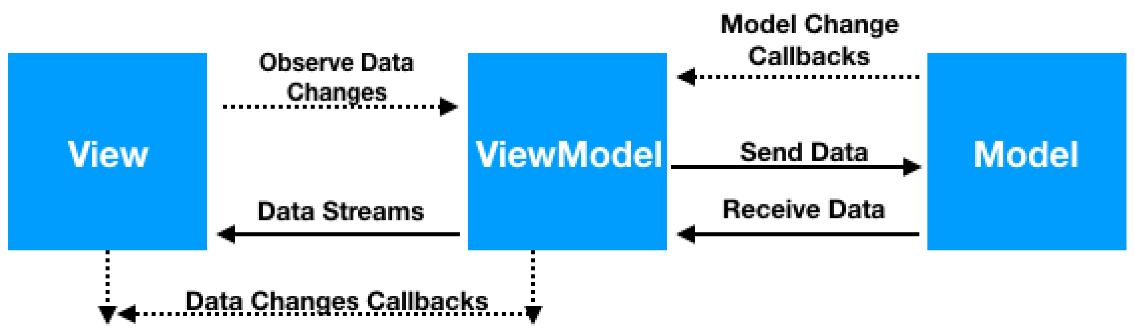

Figure 1 - MVVM Architecture

3.2.3. Data Access Object Pattern

The Data Access Object (DAO) Pattern is used to separate low level data accessing API or operations from high level business services. The business component that relies on the DAO uses the simpler interface exposed by the DAO for its clients. The DAO completely hides the data source implementation details from its clients. Because the interface exposed by the DAO to clients does not change when the underlying data source implementation changes, this pattern allows the DAO to adapt to different storage schemes without affecting its clients or business components. Every request to Firebase will be implemented through the DAO pattern.

3.2.4. Observer Pattern

The Observer Pattern allows us to notify other objects without making assumptions about who these objects are, which means, they aren't tightly coupled. This way our views are separated from the logic components and we can create unit tests easier and we have components and tests more easily maintainable.

4. Environment

4.1. Approval Environment

The approval environment is structurally identical to the testing environment. The difference between these environments lies only in the care with which the data is treated in each one. In the homologation environment, the data mass must represent the real world data as closely as possible, so that the quality assertion is effective.

4.2. Production Environment

The production environment is analogous to the type-approval environment. The difference between these environments lies in the number of nodes that make up each of the clusters as well as in the computational power of these nodes.

5. Conclusion

In this document, the architectural components of the Porto Oculto were presented, as well as the technologies to be used for the application to meet the needs of the customer.