POLITECNICO
MILANO 1863

Design and Implementation of Mobile Applications - Polaris project VINCENZO MANTO, ROBERT MEDVEDEC

# Design Document

| | |
|---|---|
| **Deliverable:** | DD |
| **Title:** | Design Document |
| **Authors:** | VINCENZO MANTO, ROBERT MEDVEDEC |
| **Version:** | 0.1 |
| **Date:** | 02-01-2022 |
| **Download page:** | https://github.com/robertodavinci/android-dev-travel-app/tree/main |
| **Copyright:** | Copyright © 2022, VINCENZO MANTO & ROBERT MEDVEDEC – All rights reserved |

# Contents



 

## List of Figures

## List of Tables

 

# 1 Introduction

## 1.1 Purpose

This document provides a detailed description, mainly of the architecture and the UI, of the 'Polaris' mobile application.
'Polaris' application is a system used to enhance travelling experience through idea sharing among people, exploration of other users' travels, documentation and easy manipulation of trip destinations and ideas. The system itself is made so the user has every single information regarding his travel in one place, without having to remember or worry about any details.
'Polaris' is mainly an Android mobile application, with a possibility of being also expanded as a web application and/or iOS application in the future.

 

## 1.2 Scope

'Polaris' is an application that helps all travellers around the world to easily and efficiently manage their travels and thus enhance their travelling experience. The target audience for this application is everyone who uses a smartphone and has at least some experience in using mobile applications, as some of the patterns and application usages might not be intended for the novices in the field. Thus the target audience ranges from 15 to 50 years old, although there are no strict boundaries.

The application is mainly intended to be used when a user is planning and making their own trip and has a liberty to organize their free time and places to visit.

Another important usage of the application is sharing trip ideas with friends and other users around the world. Planning trips by itself is a daunting and time consuming task, and with the lack of adequate applications on the market, we wanted to create something that is going to allow users to easily share and modify their previous trips and therefore improve the overall travelling experience for others.
The most important functions of the application are arranging a trip, finding points of interest in the area, organizing visits to those points, exploring accommodation and restaurants in the area, and crafting your own views of the travel by providing additional comments on the whole experience.

 

## 1.3 Definitions, Acronyms, Abbreviations

### 1.3.1 Definitions

- **Application**: a computer (mobile) program that is designed for a particular purpose.
- **Smartphone**: a mobile phone that performs many of the functions of a computer, typically having a touchscreen interface, internet access, and an operating system capable of running downloaded apps.
- **Google Maps**: a web mapping service developed by Google, used both as a standalone app and as an integrated mapping solution in most of the apps.
- **iOS**: operating system developed by Apple, used by their portable devices like iPads and iPhones.
- **Android**: most popular operating system for smartphones and tablets, developed by Google and partners.
- **Backend**: the part of a computer system or application that is not directly accessed by the user, typically responsible for storing and manipulating data.

### 1.3.2 Acronyms

- **API**: Application programming interface, computing interface which defines interactions between multiple software intermediaries
- **UI**: User interface
- **GUI**: Graphical user interface
- **DB**: Database
- **REST**: Representational state transfer - software architectural style used in web services

### 1.3.3 Abbreviations

- **App**: Application.

 

# 2 Overall Description

## 2.1 Product perspective

### 2.1.1 Internal structure

The proposed system uses an application on the server side that is connected with the APIs to the Android on the phone.

The Way of communication between the parts of the system and logic behind the system is presented in the following figure:

IMAGE HERE

**Data storing**

The database part is divided into two separate entities. Most of the authentication part is done directly through Google Firebase and Firestore frameworks, which offer great support for Google and Facebook accounts login integration, as well as native support for email based accounts. Controlling authentication via Google services is fast, easy, and seamless, so we have decided to keep most of our logic, as well as account data, stored here.

User account details, such as username, ID, and user preferences, are stored in the Google Firestore database, and are fetched after every login, so that the local user account data is consistent with the one on the servers.

ARE WE GOING TO STORE IMAGES HERE AS WELL??

The other part of the database, which stores the data about the destinations, trips, points of interests, and other features related directly to the workflow of the app, are stored in another, separate database. This decision was due to the fact that Google Firestore is a NoSQL database, so putting relational data there would make it inefficient and slow compared to the traditional SQL database.

This is what made us decide to transfer this data to completely other place and use a different technology than with the accounts. Account data storing and fetching works exceptionally well and fast with Google services, so we had no real reason to move it from there, especially since that data is more sensitive and Google services provide high level of protection.

Android framework also supports numerous features for working locally. We have used these features to have an additional "offline" mode of our application, which allows full functionalities despite the user not having internet connection. We use Android Room to store all the desired trips downloaded and edited by the user, as well as the ones that the user has created locally and decided not to publish yet. "Offline" mode is then synced as soon as the user gets Internet connection, if any additional changes have been made to the local or public trips.

Finally, shared preferences are used to store the user's local information such as username, theme preferences, display name, and other features that help user while navigating the app. These are updated with the data located in the Google Firestore framework.

**APIs & App Logic**

Authentication used for the application is done directly through Google Firbase. Account creation can be done directly through email, after which it has to be confirmed via confirmation email. The other

 

way of creating account is logging in with either Facebook or Google account. This way account gets automatically created, or joined if it has already been previously created with the same email. This allows users to not have to worry with which account they have logged in before, as it is all connected to the same email. Users would have to be logged in to either their Google or Facebook account before using those methods to log in to the app.

Google and Facebook "button" directly communicate with their respective APIs and only after the conversation has successfully ended do communicate with Google Firebase servers.

Another thing that is used from Google framework is the Google Maps API. From that API, several different features are used, such as Maps presentation, location searching, cities searching, distance calculation, and others. Google Maps are dynamically presented in the app and allow users to search through them either by text or by touch.

Another important thing is a small PHP script that is running on the server. It serves as a backend of the app, handling query fetching, control, and execution, as well as handling and displaying data that is coming and going through Google Maps API.

### Frontend

On the frontend side, we have decided to use all of the latest technologies recommended by Google for Android app development.

Kotlin is used as the programming language, which allows better data and variable control in Android environment than the previous official language Java.

From the pure design perspective, Android Compose is used instead of the previous template/xml based design. Compose allows for creation of certain elements, as well as their dynamic modelling, which allows for reusability and code cleanliness. Compose allows for elements to by dynamically inserted into specific places, as well as to be dynamically changed, hidden, or manipulated in a much faster and easier fashion. It also allows for faster development since the previews of those compose components can be rendered much faster.

As the base of the system, bottom bar with different pages has been used. More on that part is explained in latter chapters of this document.

The app supports both online and offline mode, with shared preferences used for storing local display preferences of the app.

 

### 2.1.2 Scenarios

**Scenario 1**

Marco wants to go to a trip to France. His friend, Federico, already went on a trip in a similar area and visited points of interest that Marco likes. Marco asked Federico to tell him all of the places he visited, how did he travel, and the order in which he visited these places. Federico made some notes for Marco, but it still left Marco with the problem of having everything in his phone organized and in one place. That's when Marco found out about the app and asked Federico to put everything he remembers from the trip here. That way Marco had all of the locations in the correct order, as well as comments about certain places and other detailed information all in one place, which allowed him to easier follow the trip during his time in France.

**Scenario 2**

Rebecca, the friend of Marco and Federico, heard about their trip to France and decided to go there as well. However, she is not as interested into visiting so many French castles like Marco and Federico, and wants to add some other destinations to her trip. However she still wants to keep the first half of the trip, where guys visited the southern part of France. She also uses the app and copies the trip, after which she edits it and replaces castle destinations with beaches on the northern part of the country. She renames the trip and publishes it with a different name. This way Rebecca saved some time and effort on planning the entire first half of the trip, whilst being able to modify the experience in the latter half.

**Scenario 3**

Giovanna wants to on a trip but she already visited France. She needs some new ideas. By using the "explore" function in the app she finds some of the top rated trips around Europe - her favourite of those is a trip to Iceland. Giovanna follows all of the guidelines in the trip and has an amazing time.

## 2.2 Product functions

Functions of the system provide easy and intuitive ways to use the app. They are somewhat connected and have overlapping features. Nevertheless, the users may use only certain parts of the system and still get the full functionality they need from the app. These functions are mentioned in several places in the document, but their most thorough explanation can be found here.

### 2.2.1 Adding a trip

Adding a trip is the most essential part of the system. The function features several parameters and allows for high level of customizability in order to create a fully unique travel experience.
Each point of interest is featured as a specific "destination", although even places that are not regarded as specific destinations can be inserted into a trip. Destination fetching is done through the Google Maps API, with the users also having the ability to add and edit their own destinations. The function features the following:

- Selecting a starting point, that is then connected to the nearest city on the map
- Adding other destinations to the trip
- Writing a trip description
- Adding preferred way of travel between destinations

 

- Identifying trip price level
- Adding comments to trip destinations
- .....

### 2.2.2 Editing a trip

Editing a trip can be done on two different types of trips - public or private. If the trip is public, either published by the user editing it or someone else, by starting to editing an exact copy of that trip is created, which can then be published again under different name and different trip ID.
If the trip is private and has not yet been published, then a new trip is not created, but rather the trip ID stays the same. If the trip is then published and edited again, the new edited version of the trip has a new trip ID and is a whole new entity.
Editing a trip features all of the same functions that adding a trip does, which means that everything from a small comment to the whole trip can be changed.

### 2.2.3 Finding a trip

Finding a trip can be done in several different ways. The first way features a search bar which then searches a trip by the starting point name or by the city which is the closest to that destination. The second way is a search by the trip ID or a hyperlink, which can be directly received from other users. The third way is by accessing a specific user's account page and scrolling through their published trips. Finally, trips can also be found by looking at the interactive map, selecting the area of the desired starting point of the trip, and finding the trip through the distinct trip name and photo.

### 2.2.4 Following a trip

This function mainly uses a specific trip and sets it as an active trip of the user. This makes it easily accessible by the user at all times by using the bottom navigation bar, and allows him to follow certain steps of the trip without losing progress. This function also allows the user to change the current active trip and still keep the progress of an old trip, so that the progress can be easily restored when that trip is again set as an active trip.

### 2.2.5 Updating account settings

Only a few settings can be changed in the user account. List is the following:

- Changing a username
- Changing a profile picture
- Switching between dark and light colour scheme
- Setting preferred price level
- Switching between offline mode and online mode

### 2.2.6 Offline function

The system allows the users to be disconnected from the Internet and still use the app fully. Individual trips can be downloaded and store in the phone internal storage, and then accessed at any time. If any changes are made to the trip during the offline time, a new iteration of the trip gets created and published as soon as the connection is restored and the user has come back to the online mode. Multiple trips can be downloaded and kept in the storage of the phone at all times.

 

# 3 Architectural Design

## 3.1 Overview

Architectural design of the application is based on the three-layer model used in most applications. The three layers are the presentation layer, or frontend, the application layer, or middleware, and the data layer, or backend. Each of those layers does its own part of the job and communicates with other layers in order to present the correct information to the user.
The architecture is also a typical client-server implementation where server holds the data and the client is accessing it through requests (besides the offline mode where the user stored some of the data from the server locally and is accessing it without using online requests).

### 3.1.1 Backend architecture

**User database**

As previously mentioned, user database is located in the Google Firestore service. This service uses real-time NoSQL database. This database is organized in collection->document system. Everything starts with one collection, which can hold as many documents (entities) as possible. Each document can have an unlimited number of attribute fields, which can be with a specified type or without one, and at most one collection. Then the cycle repeats again.

Working with NoSQL has its advantages when it doesn't have a lot of relational and connected data. We can extract only the small amount of data we want, it is very fast, and also very efficient. Since accounts are not connected in any way, using NoSQL database seemed like a viable choice since it saves our users time and data.

The architecture of this database is the following: the first collection contains all the users as their unique IDs, where they are represented by a document. The first level of the document holds only the info of the display name and the ID. The second collection is made for storing user preferences, and every user has their own. In that collection there is a document that holds all of the parameters needed for the user's usage of the app, such as colour mode, economy level, thumbnail URL; and two optional ones, real name and real surname

The following figure represents the structure of the explained database.
IMAGE OF THE NoSQL architecture

### 3.1.2 Middleware architecture

In order to connect the data on the server to the screen of the phone and to allow the user to properly see the data, we have implemented a complicated layer of functions and classes in order to create easy-to-use and esthetically pleasing experience for the user.

## 3.2 Runtime view

In the following section, sequence diagrams that represent the most important use cases are shown and explained in detail. We have focused only on the most commonly used use cases in the app, as well as their crucial parts. We do not go into too much detail when it comes to specific parts of interaction between the layers, but rather want to present the vague idea and how it all works.

 

### 3.2.1 Create a trip

Create a trip use case starts with opening the app and pressing on the "Create a trip" button. A new screen opens up which allows us to change certain attributes about the trip and add different destinations. Every trip needs to have a name, which doesn't have to be unique, since there is an ID that is automatically allocated to the every trip. Every trip needs to have at least two destinations. We can search through destinations via the search bar or by using the interactive map and selecting a destination from the map. After selecting a destination, we add it to the trip. Destinations can be reordered and removed from the trip at any time, which is not shown in the diagram due to simplicity and concision. The price level of the trip should also be defined before saving the trip. Trip can also be discarded. In the end, if the user is satisfied with its trip, they can publish it in which case it will be stored directly in the database. Local database is not used for storing unpublished trips, since this would mean that by losing local data, all of the unpublished trips would be lost. All of the trips are stored in the online database and are can be found by other users only if published.

IMAGE OF CREATE A TRIP

### 3.2.2 Edit a trip

Edit a trip use case starts with opening the app and choosing one of two ways of searching for a trip. One way is searching for a trip name/ID directly through the search bar, and another is by searching for a trip on the interactive map. After the trip is found, it is somewhat "copied" to the instance of the user. User can then edit every single bit of the trip - change the name, change the price level, and add/remove destinations (again, destination removal is not shown in the graph due to simplicity). Finally the user can choose to save the trip without publishing it, or publish it so that everyone else is able to find it. Either way, the trip gets a brand new ID only if it has been changed in any way (changing only the trip name and/or price level is not regarded as a change). Again, due to data safety, in either case it is stored online.

IMAGE OF EDIT A TRIP

### 3.2.3 Add a destination

Add a destination use case starts with opening the app and choosing one of two ways of adding destination - either by using the dynamic Google Maps API, or by using a search function which searches for specific places and locations in the Google database. By using a Google Maps way, the user can specify the exact location, while by using the search way, the user can pick of the previous existing locations in the database. After picking a location, the user can change a name and/or add an specific image of this location. This way the user is allowed to personalize locations for their trip needs. After saving all of the data in the form of a destination, the new destination is published and can be found by other users and added to their trips. There are no local and unpublished destinations.

IMAGE OF ADD A DESTINATION

### 3.2.4 Search for a trip/explore

## 3.3 Testing

 

# 4 User Interface Design

UI

 