smartparking

# Design Document

Gregorio Galletti - Ibrahim El Shemy

A.A. 2019/2020 - Prof. Luciano Baresi

## Contents

# 1 Document Structure

- `Introduction`: This section introduces the Design Document. It explains the Purpose, the Scope and the conventions of the document.
- `Architectural Design`: This section describes the components used for the system and the relations between them, providing information about their deployment and how they works. It also specifies the architectural styles and the design patterns chosen to design the system.
- `Third Party interaction`: This section introduces the interaction of SmartParking with the listed Third Party APIs and services and their functionalities.
- `User Interface Design`: This section provides an overview on how the User Interface will look like. This section will be accurate enough to explain all our decisions about the design and the UI of the Mobile Application.
- `Implementation and Integration`: This section contains the order of the system's subcomponents implementation and their integration.
- `Testing`: This section describes the test cases submitted to the Application.

# 2 Introduction

## 2.1 Purpose

This document represents the Design Document (DD) for SmartParking mobile application. The purpose of this document is to provide an overall guidance to the architecture of the software product and the interaction between all the components of the system to be developed, following the requirements and the goals that the software must satisfy.

## 2.2 Scope

SmartParking is a crowd-sourced application where users can view all the street parkings around them, together with detailed information. Users can also filter the parkings in several ways: searching for a specific address, a specific type, a maximum distance, etc... Moreover, users can pay the fee directly from the app, chosing the payment type and how much they want to stop.

## 2.3 Definition, Acronyms, Abbreviations

### 2.3.1 Definitions

- `User`: any client of the service, a person that logs in the system and uses it.
- `User Device`: any compatible device with the SmartParking application, mainly smartphones.

- `App`: abbreviation for the SmartParking Mobile Application.

### 2.3.2 Acronyms

- DD: Design Document.
- API: Application Programming Interface.
- GPS: Global Positioning System.

## 2.4 Revision History

- 3/04/2020: First Version of DD Document.

## 2.5 Reference Documents and Used Tools

**Reference Documents**

**Used Tools**

- *Github*: https://github.com/
- *TexMaker*: https://www.xm1math.net/texmaker/
- *Draw.io*: https://www.draw.io/
- *AdobeXD*: https://www.adobe.com/it/products/adobexd.html
- *LucidChart*: https://www.lucidchart.com/

## 2.6 Domain Assumptions

- [D1]: Users' devices support the GPS technology.
- [D2]: Users' devices support receiving push notifications.
- [D3]: Users' devices always have an internet connection available during the interaction with the system.
- [D4]: User's devices support the Mobile application (including Google Maps services).
- [D5]: It is possible to retrieve the current time from the users' devices.

### 2.7 Goals

The system provides to users a simple and user-friendly interface to:

- [G1]: Register and Login/Logout from the application.
- [G2]: Localize themselves on the map and listen to location changes.
- [G3]: Find and choose parkings from the map or from the list.
- [G4]: Pay or extend their stop.
- [G5]: View their parking history or their active parking.
- [G6]: Report missing/malformed/outdated information on parkings.
- [G7]: Earn points from reports and stops and convert them in Bonus.

## 3 Use Cases Analysis

**UC1**

| | |
|---|---|
| **Name** | Register to SmartParking |
| **Actor** | Visitor |
| **Goals** | [G1] |
| **Entry Condition** | The Visitor has installed the Mobile Application on his/her device |
| **Events Flow** | 1. The Visitor fills all fields in the Registration Screen(s)<br>2. The Visitor taps on the "Continue" button<br>3. The system checks data validity<br>4. The system stores the new User's data |
| **Exit Conditions** | The Visitor has registered to SmartParking |
| **Exceptions** | 1. The Visitor inserted an e-mail that is already used for an account<br>2. The Visitor inserted an invalid e-mail<br>3. Some fields are not filled<br>4. The Visitor inserted a password that does not satisfy the security rules |

**UC2**

| | |
|---|---|
| **Name** | User Login |
| **Actor** | User |
| **Goals** | [G1] |
| **Entry Condition** | The User is on the Login Screen of the Mobile Application |
| **Events Flow** | 1. The User enters the username<br>2. The User enters the password<br>3. The User taps on the "Log in" button<br>4. The system check the validity of the provided data |
| **Exit Conditions** | The User is logged in |
| **Exceptions** | 1. The User is not registered<br>2. The User inserted a wrong username<br>3. The User inserted a wrong password<br>4. Some fields are not filled |

**UC3**

| | |
|---|---|
| **Name** | Social Login/Registration |
| **Actor** | User |
| **Goals** | [G1] |
| **Entry Condition** | The User is on the Login Screen of the Mobile Application |
| **Events Flow** | 1. The User taps on the Google/Facebook button<br>2. The User allows SmartParking to use Google/Facebook for authentication<br>3. The system stores the User's data |
| **Exit Conditions** | The User is registered and logged in |
| **Exceptions** | 1. The User has no Google/Facebook accounts<br>2. The User denies the permission to authenticate |

# 4 Architectural Design

Our application is composed by two main components: a Front End, developed with the React Native framework, and a Back End that relies on Firebase, and in particular on Firebase Realtime Database.
These two components are highly connected: in fact, the mobile application has the role to show to the user all the data stored in the Database, and also to respond to the user's

behaviour.

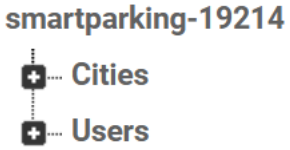
## 4.1 Front End: Mobile Application

The Mobile Application is a React Native application composed by several screens, each one will be described in the User Interface Design Section. The choiche of React Native frameworks comes from the fact that, being SmartParking an application that will be mainly used "in real time", smartphones will be the target devices: it's more reasonable that a user is using his smartphone when driving (obviously respecting the street law code) rather than a tablet or a smartwatch. A **cross-platform** framework has been identified as the smartest choiche in this case, in order to reach the largest number of users.
This Application will obviously be a **multi-thread** application. This because we need to be able to handle all user actions and at the same time to keep the Google Map and the shown parkings updated, without having blocking instructions and so performing all the actions in parallel.
The Mobile Application is composed of a total of X screens (better described in Section Y), and the most important of these are:

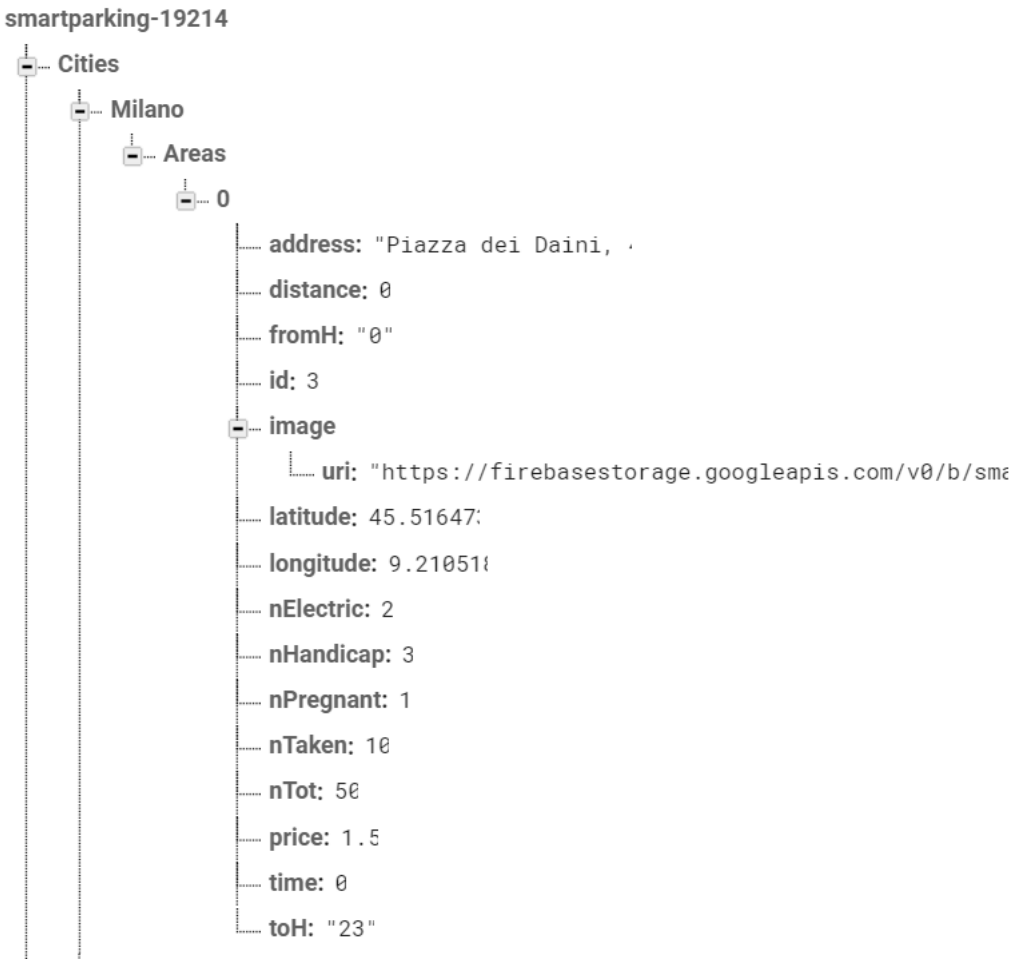
- Home/Map
- Parkings
- Profile

## 4.2 Back End: Firebase

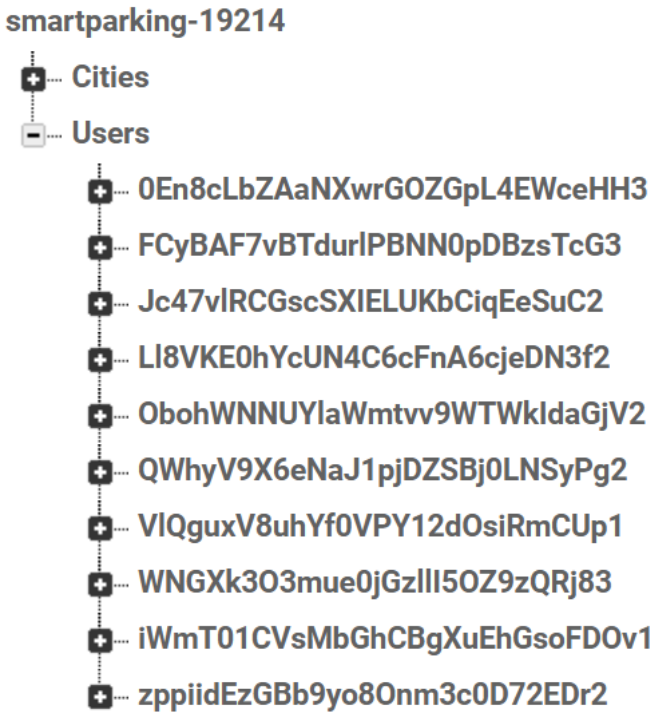
The Firebase back end is where all the parkings, users and reservations data are stored. The back end is also encharged of authentication and user session management (a well implemented and tested functionality of Firebase).
The main data that has to be stored is the parking Area data: every city has some areas, and each of them includes a number of parking spots of different type: everything must be saved.
Since Firebase RD is a NoSQL Database we have a JSON-style structure, where we have 2 main folders: **Cities** and **Users**.



**Cities** In this folder are stored all the Cities included in the project, and each of them contains all the parking Areas that will be shown to the user.




**Users** In this folder are stored all the Users registered to SmartParking (by their ID), and each of them contains all the needed information about them such as e-mail, name, preferences, etc..




## 4.3 Architectural styles and patterns

### 4.3.1 Design Patterns

- `Model-View-Controller Pattern`: is a software design pattern commonly used for developing user interfaces which divides the related program logic into three interconnected elements. This is done to separate internal representations of information from the ways information is presented to and accepted from the user. Following the MVC architectural pattern decouples these major components allowing for code reuse and parallel development.
  In SmartParking, we can say that the `Mobile App` represents both the **View** and the **Controller** of the system, while the entire `Back end` and in particular the Database represents the **Model**. That is, the App shows the relevant data to the user, and provide the possibility to choose between some options to modify the shown data: it behaves like a Controller, calling fetch methods to update the Server-side data.
  It is worth to mention that, in some cases, the MVC Pattern with a local copy on the client side has been used: this because relying on fetch methods everytime the user modifies some data would be too heavy from a computational point of view, and also too much internet-consuming.

- `Observer/Observable Pattern`: is used when there is one-to-many relationship

between objects such as if one object is modified, its depenedent objects are to be notified automatically. Observer pattern falls under behavioral pattern category. This is the natural Design Pattern used to implement the MVC Pattern: in our system, the intent of this pattern is to let the User execute some queries through the UI and after searching the Database, the result is shown back in the UI. If changes occurs in the database, the UI is notified and updates the UI.
We can say that React Native helped us a lot in implementing this pattern, due to the intrinsic presence of this. All RN components, in fact, are "linked" to state values, and modifying them results in modifying also the UI to which the value is linked.

# 5 Third Party interaction

## 5.1 Google

## 5.2 Facebook

## 5.3 Paypal

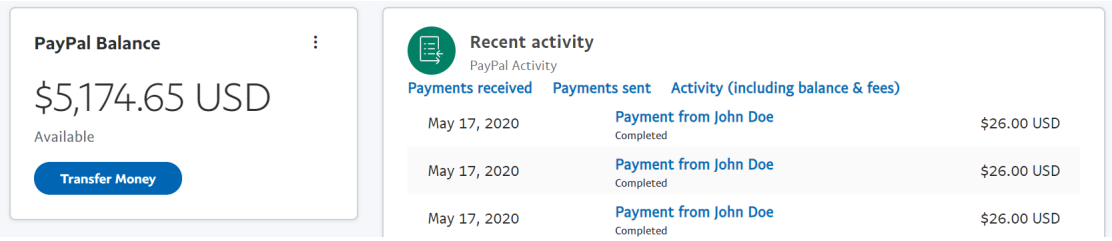



Figure 1: Merchant account history of received payments

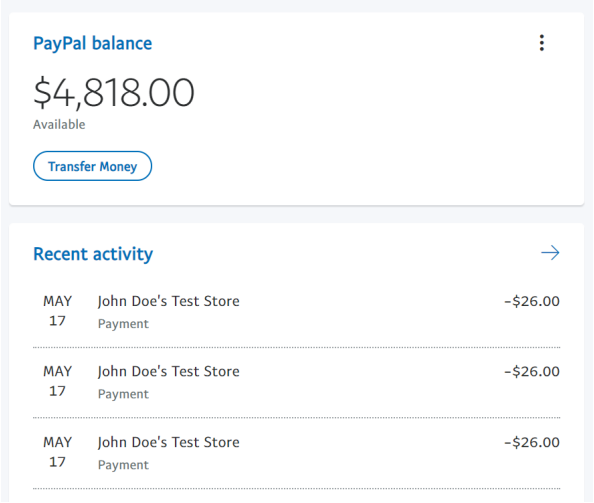



Figure 2: Customer account history of processed payments

## 5.4 Stripe

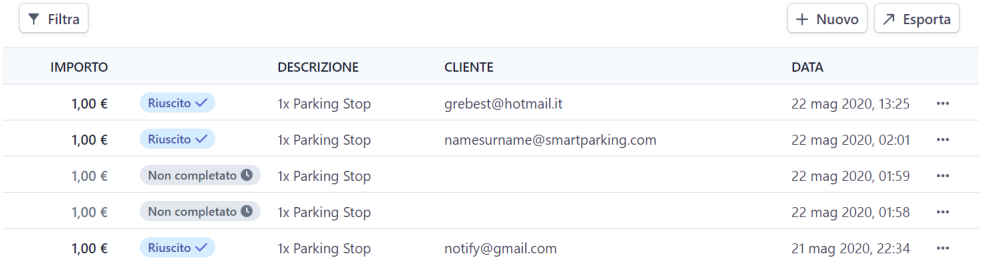



Figure 3: History of payments received or deleted by users

## 5.5 Redux

## 5.6 Google Maps APIs

# 6 User Interface Design

In this section we will describe, page by page, all the Screens of the Mobile Application and their functionalities, along with the decisions we made to provide a nice and pleasant User Experience.

# 7 Implementation and Integration

We can split this section in three main phases, written in order of implementation:

## 7.1 Front End implementation

First of all we created all the needed screens with a very basic design: we were focused on providing a good User eXperience flow first by identifying all the best components to insert, and only after this phase we started thinking about the color, shape and less-relevant details. This is a very common development flow in mobile applications, and works very well when managing a lot of screens and user flow possibilities.

## 7.2 Back End implementation

We created the Back end skeleton of the system, by configuring Firebase Realtime Database in order to store every needed information. In particular, we started by organizing and adding all the involved Cities (in the very first version only Milan and Sondrio, the group members' cities), and then adding to them all the parking Areas.

## 7.3 Front End and Back End integration

This phase was relatively short, because

# 8 Testing

Nothing.