smartparking

Design Document

Design and Implementation of Mobile Applications

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 Functional Requirements

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

- [R1]: Register and Login/Logout from the application.
- [R2]: Localize themselves on the map and listen to location changes.
- [R3]: View parkings from the map or from the list.
- [R4]: Filter parkings with his/her preferences.
- [R5]: Pay or extend their stop.
- [R6]: View their parking history or their active parking.
- [R7]: Report missing/malformed/outdated information on parkings.
- [R8]: Earn points from reports and stops and convert them in Bonus.
- [R9]: Receive notifications on their stop.
- [R10]: Personalize the visualization of the application.

2.8 Non-Functional Requirements

3 Use Cases Analysis

In this section we will provide a list of the most relevant Use cases of the system, for a total of 6 of them:

Use Case 1

Name	Register to SmartParking
Actor	Visitor
Requirement	[R1]
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) 2. The Visitor taps on the "Continue" button 3. The system checks data validity 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 2. The Visitor inserted an invalid e-mail 3. Some fields are not filled 4. The Visitor inserted a password that does not satisfy the security rules

Use Case 2

Name	User Login
Actor	User
Requirement	[R1]
Entry Condition	The User is on the Login Screen of the Mobile Application
Events Flow	1. The User enters the username 2. The User enters the password 3. The User taps on the "Log in" button 4. The system check the validity of the provided data
Exit Conditions	The User is logged in
Exceptions	1. The User is not registered 2. The User inserted a wrong username 3. The User inserted a wrong password 4. Some fields are not filled

Use Case 3

Name	Social Login/Registration
Actor	User
Requirement	[R1]
Entry Condition	The User is on the Login Screen of the Mobile Application
Events Flow	1. The User taps on the Google/Facebook button 2. The User allows SmartParking to use Google/Facebook for authentication 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 2. The User denies the permission to authenticate

Use Case 4

Name	Filter the Parkings
Actor	User
Requirement	[R4]
Entry Condition	The User is on the Home Screen
Events Flow	1. The User taps on the Filter icon on the Searchbar and goes to the Filter Screen 2. The User selects the options he wants 3. The User taps on the "Apply" button 4. The system checks for the resulting parkings 5. The system sends the User back to Home Screen
Exit Conditions	The User has successfully filtered the parkings
Exceptions	1. No parkings satisfy the filter parameters 2. The User tapped on the back button on the Filter Screen

Use Case 5

Name	Pay for a Parking
Actor	User
Requirement	[R5]
Entry Condition	The User is on the Home Screen after Login
Events Flow	1. The User taps on the Parking marker 2. The User taps on the ”Pay Parking” button on the modal and goes to the Details Screen 3. The User selects the end time of the stop 4. The User taps on the ”Pay with Credit Card”/”Pay with Paypal” button 5. The system redirects the user to the corresponding payment page 6. The User inserts his payment data and taps on the ”Pay” button 7. The system checks for the result of the payment 8. The system sends the User back to Home Screen
Exit Conditions	The User has successfully paid a stop
Exceptions	1. Some payment fields are not filled 2. Some payment fields are wrong 3. The payment failed 4. The User canceled the payment

Use Case 6

Name	Send Report
Actor	User
Requirement	[R7]
Entry Condition	The User taps on a Marker on the Home Screen
Events Flow	1. The User taps on the ”Report” button and goes to the Report Screen 2. The User taps on the Report he want to make 3. The User taps on the ”Send” button 4. The User shows the User how many points he/she earned and sends the User back to Home Screen 5. The system receives and stores all the data of the Report
Exit Conditions	The User has successfully sent a report
Exceptions	1. The User tapped on the back button on the Report Screen

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.

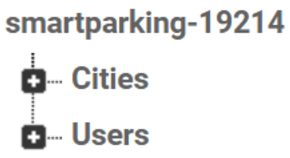
4.1 Front End: Mobile Application

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:

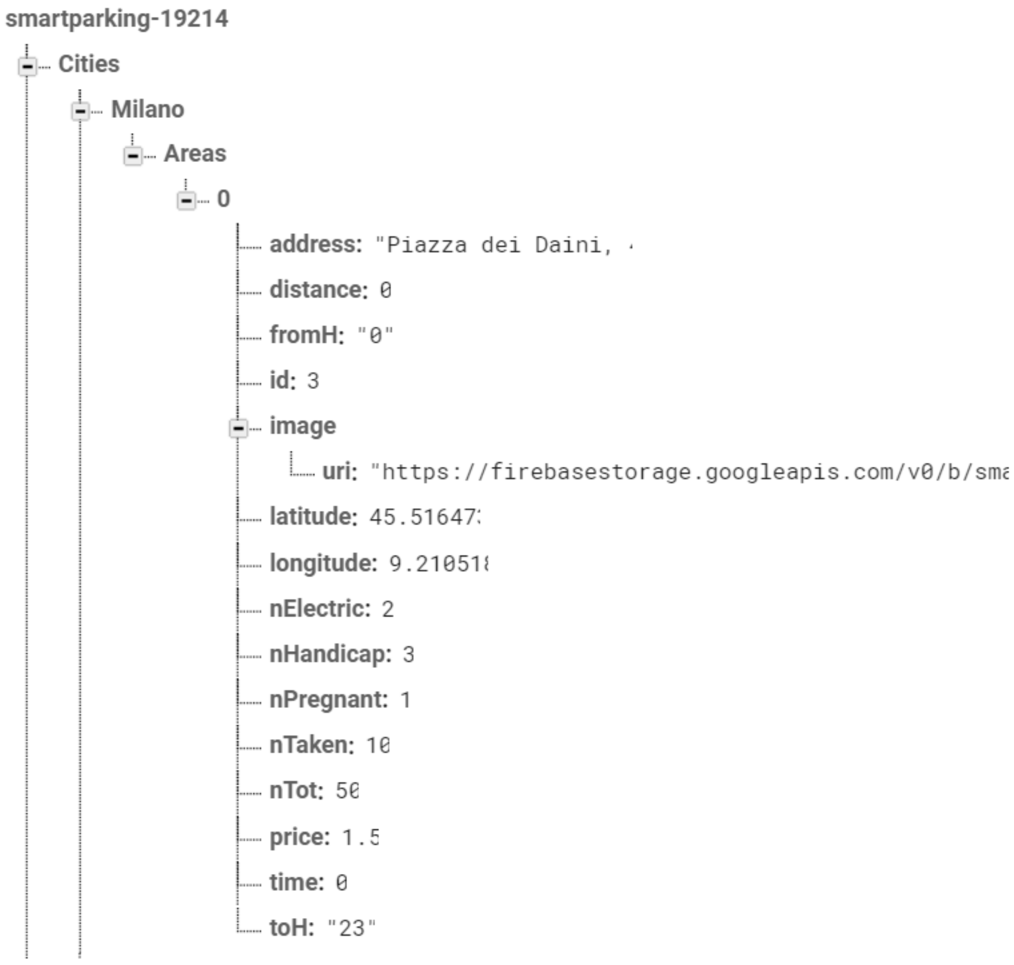
- Home/Map
- Parkings
- Profile

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**.

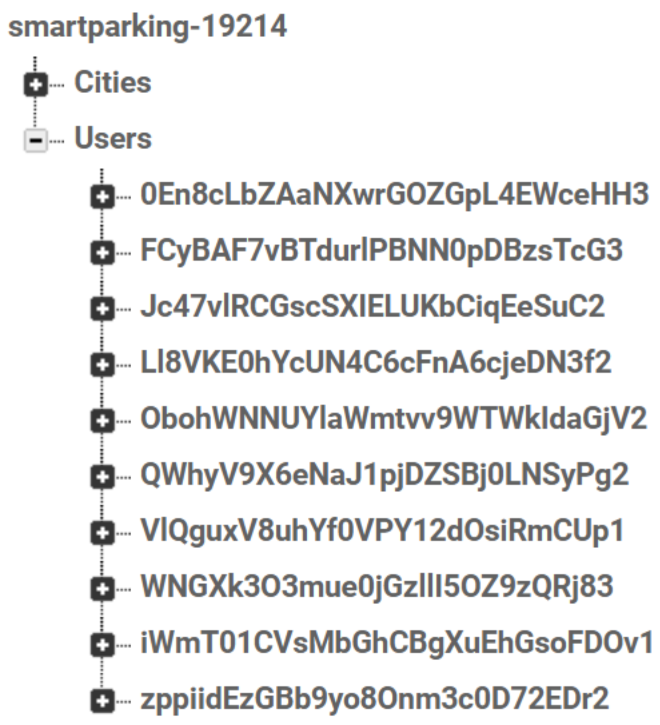


Cities In this folder are stored all the Cities included in the project, and each of them contains all the parking Areas that will be shown to the user.



```
smartparking-19214
  Cities
    Milano
      Areas
        0
          address: "Piazza dei Daini, 
          distance: 0
          fromH: "0"
          id: 3
          image
            uri: "https://firebasestorage.googleapis.com/v0/b/sma
          latitude: 45.51647
          longitude: 9.21051
          nElectric: 2
          nHandicap: 3
          nPregnant: 1
          nTaken: 10
          nTot: 50
          price: 1.5
          time: 0
          toH: "23"
```

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..



```
smartparking-19214
  Cities
  Users
    0En8cLbZAaNXwrGOZGpL4EWceHH3
    FCyBAF7vBTdurlPBNN0pDBzsTcG3
    Jc47vlRCGscSXIELUKbCiqEeSuC2
    Ll8VKE0hYcUN4C6cFnA6cjeDN3f2
    ObohWNNUYlaWmtvv9WTWkIdaGjV2
    QWhyV9X6eNaJ1pjDZSBj0LNSyPg2
    VlQguxV8uhYf0VPY12dOsiRmCUp1
    WNGXk3O3mue0jGzlll5OZ9zQRj83
    iWmT01CVsMbGhCBgXuEhGsoFDOv1
    zppiidEzGBb9yo8Onm3c0D72EDr2
```

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

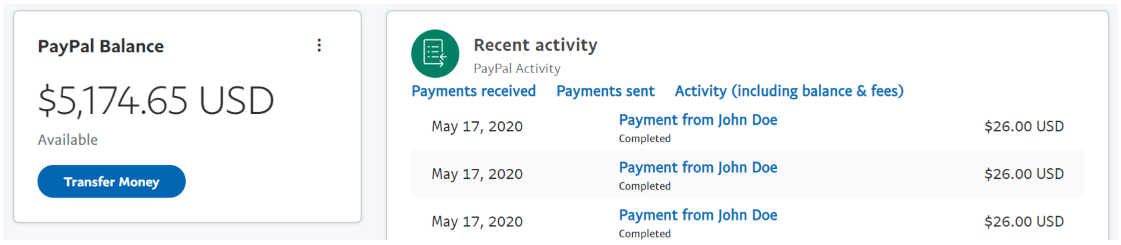
between objects such as if one object is modified, its depenedent objects are to be notified automatically. Observer pattern falls under behavioral pattern category. This is the natural Design Pattern used to implement the MVC Pattern: in our system, the intent of this pattern is to let the User execute 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



PayPal Balance

$5,174.65 USD

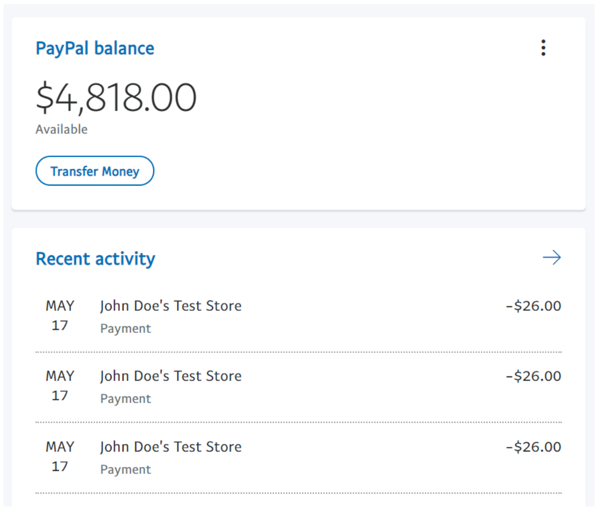
Available

Recent activity

PayPal Activity

May 17, 2020	Payment from John Doe Completed	$26.00 USD
May 17, 2020	Payment from John Doe Completed	$26.00 USD
May 17, 2020	Payment from John Doe Completed	$26.00 USD

Figure 1: Merchant account history of received payments



PayPal balance

$4,818.00

Available

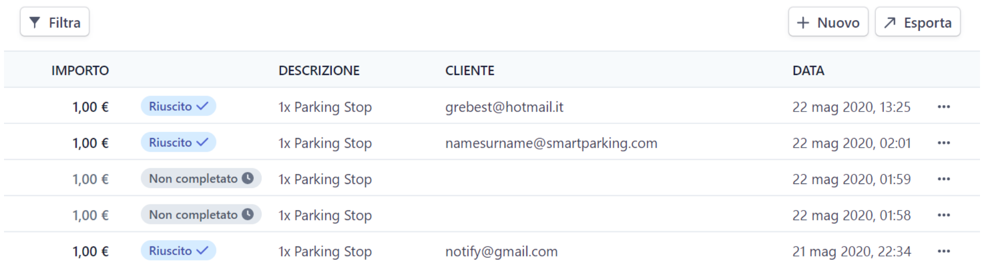
Transfer Money

Recent activity

MAY 17	John Doe's Test Store Payment	-$26.00
MAY 17	John Doe's Test Store Payment	-$26.00
MAY 17	John Doe's Test Store Payment	-$26.00

Figure 2: Customer account history of processed payments

5.4 Stripe



Filtra

Nuovo Esporta

IMPORTO		DESCRIZIONE	CLIENTE	DATA	
1,00 €	Riuscito ✓	1x Parking Stop	grebest@hotmail.it	22 mag 2020, 13:25	···
1,00 €	Riuscito ✓	1x Parking Stop	namesurname@smartparking.com	22 mag 2020, 02:01	···
1,00 €	Non completato	1x Parking Stop		22 mag 2020, 01:59	···
1,00 €	Non completato	1x Parking Stop		22 mag 2020, 01:58	···
1,00 €	Riuscito ✓	1x Parking Stop	notify@gmail.com	21 mag 2020, 22:34	···

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.

6.1 Colors

6.2 Fonts

6.3 Splash Screen

6.4 Welcome Screen

6.5 Login Screen

6.6 Registration Screen

6.7 Home/Map Screen

6.8 Parking Screen(s)

List Screen

Active Parking Screen

History Screen

6.9 Filter Screen

6.10 Profile Screen

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.

9 Software System Attributes

9.1 Reliability

The system guarantees a 24/7 service. Even if some Back end data has to change the system can be updated without downtimes, because every change in the Database will be immediately received and correctly updated from the Mobile Application.

9.2 Availability

The system is not used for critical services so the occurrence of faults is tolerated. However, given that the Mobile Application relies on safe and highly tested libraries and Third Party components and APIs, we can estimate the system availability to be at least at 99%.

9.3 Security

The only Security aspect that the system must handle is the credentials transmission and storage: Firebase does that, and obviously no password data is stored in the Database. Also for in-app payment, no sensible data is stored and Stripe/Paypal APIs provide a secure transmission of data.

9.4 Compatibility

The system will be hopefully used by a lot of people, so the Mobile Application must be compatible to as many devices as possible. Moreover, Hardware Limitations are not so tight because nowadays nearly every smartphone should satisfy them, making Compatibility easier to reach. In addition, nearly every smart device can run the application: every iOS or Android tablet and every iOS or Android smartphone.

9.5 Scalability

The architecture is simply scalable as the number of users grows during the time. Enlarging the structure of the system is be an easy task: Firebase helps us a lot in this case, because the number of possible users is huge and the parking Areas and Cities can be expanded in a really easy way, by simply adding them one after the other in the Database (also thanks to the NoSQL structure).