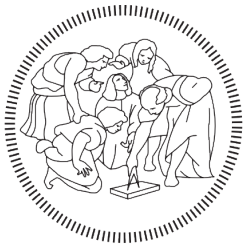
POLITECNICO DI MILANO

School of Industrial and Information Engineering

Computer Science and Engineering



POLITECNICO
MILANO 1863

Design and Implementation of Mobile Application Project: Safe Car Design Document

Course Professor: Prof. Luciano BARESI

Authors:

Mattia CRIPPA 1039725

Alberto PIROVANO 10396610

Academic Year 2017–2018

Abstract

SafeCar is an application that has been designed to help the driver in improving its driving style. On one side it offers the possibility to inspect the historical user data, and on the other side it provides a hint generation engine during the trip . The core algorithm of the application communicates in an asynchronous way with a physical object that has to be plugged into the car. This object is the interface between the car and the application, and once connected with it the application is able to access and process the data about the navigation. During the trip the application analyses these data, processing them to profile the driving style. In addition it generates an index representing a summary of a specific moment of the trip. Based on this index, another algorithm generates a hint message. This procedure is performed in loop, so the application generates hints continuously during the trip and through these hints the user can obtain some indications about how to improve his driving style.

Contents

List of Figures

Chapter 1

Introduction

The *Design Document* is meant to provide documentation that developers can use in implementing the entire system, it includes a general description of the architecture and the design of the system to be built.

1.1 Purpose

The purpose of the *Design Document* is to provide a description of the system detailed enough to understand which are the components of the system, how they interact, which is their architecture and how they will be deployed. The level of the description is high enough for all the stakeholders to capture the information they need in order to decide whether the system meets their requirements or in order to begin the development work.

1.2 Scope

This document provides a detailed description of *Safecar* software design and architectural choices. Every portion of the document is designed to be comprehensible, but in order to fully understand the topic, the reader should already have a general idea of the system.

1.3 Definitions & Acronyms

We are going to use a set of specific terms, each of them referring to a specific abstract or physical object:

1. **Plug**: It is a smart object that the user has to buy and that has to be plugged inside a specific port of the car. It must have Bluetooth functionalities enabled
2. **Trip**: An abstract concept data structure containing all the information about a travel
3. **Badge**: An abstract object that refers to a specific unlocking condition. It can be locked or unlocked

In the document some technical terms are often used whose definitions are here reported:

1. **Layer**: A software level in a software system
2. **Tier**: An hardware level in a software system

We also need some application specific terminology:

1. **DSI**: Driver Safety Index
2. **ALG1**: High-level hint generation algorithm
3. **ALG2**: DSI computation engine
4. **ALG3**: Low-level hint generator agent

1.4 Document Structure

This document is intended for individuals directly involved in the development of *Safecar* application. This includes software developers, project consultants, and team managers. This document is not meant to be read sequentially; users are encouraged to jump to any section they find relevant. Below is a brief overview of each part of the document:

1. **Introduction:** This section gives general information about the *Design Document* of *Safecar* application

2. **System Overview:** This section contains an overview of the application and its primary functionalities. It also contains assumptions and constraints followed during the design of the software

3. **Architectural Design:** This section exposes in details the design chosen for the architecture of the system to be

4. **User Interface Design:** This section provides the detailed design information for each component in the current delivery

Chapter 2

System Overview

The aim of our project is to develop an Android application which can be used in the real world.
In order to use this application, the driver has to register as a user of the service. The *Registration and Login* procedures can be performed via the custom application functionality or via Google Plus APIs. The user can change his/her password, drop the account and modify data related to his/her profile whenever he/she wants. At this point the application flow develops in two different ways:

- If the user's smartphone is not in the radio scan area of the Plug, he/she can only inspect user related data about his/her history usage of the application. He can consult the performed trips, his/her profile data or the gained badges

- If the user's smartphone is close to the Plug, he/she can pair his/her device with that Plug and, after the procedure has succeeded, he/she can start a new trip. Now the *Driving Experience* actually begins. After this moment, the application will follow the user in his/her driving experience by asking to the plug data about the navigation, for example the acceleration rate or the frequency with which the user is breaking. On the base of these data, the driver will be provided with several hints about how to improve his/her driving style, until he/she decides to end the trip

When the trip ends, the application will provide an after trip report graphic

showing the followed route on a custom map.
The application is comprised of two main features:

1. **After Trip data presentation:** The user can inspect the details of the trip he/she just completed. In this screen the user can see:
   - The route he/she followed, shown in a custom Google maps widget
   - A general report about the completed trip providing the departure, the arrival, the date of the trip, the kilometres traveled, the time that the trip took and the *Driver Safety Index* (DSI), that estimates the quality of the drive
2. **During Trip functionalities:** This functionality, is provided by the core *Safecar's* algorithm. This algorithm is an engine that has the role of estimating the *Driver Safety Index* based on navigation data coming from the car. These data come from a smart object, a plug, that has to be inserted in the custom car gate and has the capability of sending cleaned and custom driving style data to the connected device

Based on these data, the algorithm computes the *Driver Safety Index* (DSI). The engine architecture is easily exchangeable, given that the application is built in a parametric way. Building a more complex algorithm is only a matter of replacing the specific piece of code (a method) with another one that takes as input the same data and generates a same type index, but by implementing a different, maybe more complex, logic. In order to provide the user with an almost continuous feedback about his/her driving style, this engine recomputes the DSI index once every 30 seconds. This computation is done in a dedicated, separate thread that generates the current piece of advice to be sent to the user by simply switching on the value of the DSI. This hint is conveyed by filling a specific screen on the application.

2.1 System constraints

These are the constraints which must be met in order to allow the application to work correctly:

- The user of the mobile should have partial internet access:
  - The user application must have the internet access at least during the login and the logout procedure. In fact, during the actual use of the application it doesn't need the data connection
- User's device should support bluetooth:
  - The application needs the bluetooth for the interactive part, the one in which the app follows the user during the navigation. During the offline part, it doesn't need any bluetooth capability
- User's device should support geolocalization:
  - The application needs the geolocalization for the interactive part, the one in which the app follows the user during the navigation. During the offline part, it doesn't need any geolocalization capability

Chapter 3

Architectural Design

This section exposes *Safecar* Architectural Design in a complete e comprehensible way.

3.1 Layers

The software architecture that has been chosen follows the principles of the *Model-View-Controller* architectural pattern. Therefore three main software components have been identified, i.e: the Model, the View and the Controller.

3.1.1 View

The role of this Layer is that of processing Clients commands, and converting them into requests addressed to the Controller layer.

3.1.2 Controller

In this second Layer are included all the software components that implement the system logic, the authentication logic and the high level data primitives. The application logic is handled by the Firebase cloud authentication services and by some ad hoc authentication components.

3.1.3 Model

The third and last Layer is the Model, that should guarantee a high level interface which stores and manages all the *Safecar* relevant data. This is provided

by Firebase cloud database.

3.2 Tiers

The system can be divided in two different Tiers, which are the Clients and the Cloud Database

1. **Clients**: This is the Android mobile application. The view and the controller layers are mapped on this Tier. A thin model is also present on the Client Tier

2. **Cloud Database**: This Tier hosts the Database that allows the service data persistence. The Model layer is entirely mapped to this Tier

This architecture, with three different Layers and two different Tiers, makes it possible to have a Client-Server style with a Fat-Client and a Thin-Server. This is due to the mapping of the View, the Controller and the thin model to the Client and the complete Model to the Server.

3.3 Cloud and Local Model

Once the user logs in, the application downloads all the user related data from the cloud model and dumps them into a local copy called Local Model.
This wrapper is build through the Realm real time database APIs. In particular, when the user modifies some piece of data, the application updates its local model item, leaving the cloud model unchanged.
For example, this happens when the user decides to drop a trip, add a trip, drop a plug or add a new plug. For this, there is no need of internet connection after the login and before the logout procedures, in fact all the modifications are performed locally.
When instead the user decides to log out, the application understands which are the data items that have been modified and uploads their modifications to the cloud model in order to make it consistent with the local one. After pushing the modifications the controller drops the local model and exits the user session.

3.4 Application components and their interaction

The following section details all the architectural aspects that characterize the software to be. Here it is explained which are all the components of this application and how they interact in order to have a complete functional software.

3.4.1 Login component

This component handles the user's login procedures and provides either the Google+ authentication service or the email and password authentication service.

3.4.2 Sign in component

It manages the Sign in procedures, that is the generation of a new user object.

3.4.3 Reset password component

It is in charge of handling the procedure of password change for the user that has been registered through the email and password login procedure.

3.4.4 Logout component

It ensures the consistency between the Cloud Model and the Local Model.

3.4.5 Personal data component

This is responsible of computing and handling the modifications to the profile information.

3.4.6 DSI computation engine

This is the most important component, in fact is in charge of:

- Ordering and placing the trips contained in the local model into in the four dedicated tabs. This is done by using four software components,

respectively *DateComparator*, *DSIComparator*, *KMComparator* and *DurationComparator*

- Handling the sampling of position pins during the trip. It also stores them into the current trip object. This is done by the *TripHandler* thread, an asynchronous task. It is spawned when the trip starts and it autonomously handles it
- Through a call to the first algorithm, namely *ALG1*, it computes the DSI index through the *DSIevaluator* asynchronous task
- After that, *Safecar* exploits the feature of the second algorithm, namely *ALG2* to compute the current hint

3.4.7 Plug component

This component implements the logic for pairing, managing and using the abstract interface of physical plugs.

3.4.8 Badges component

This component implements the procedure that, by using the user's data, computes the badges the user is currently holding.

3.5 External services

Safecar is fully integrated with several external services

3.5.1 Firebase database

Safecar makes use of the great Firebase cloud database to store the data about trips, users and plugs. It represents the persistent storage of the application.

3.5.2 Firebase authentication

Firebase is also great for the authentication capabilities that it offers. *Safecar* exploits the Google authentication services but it can also straightforward integrate Facebook, Twitter and GitHub.

3.5.3 Bluetooth core service

In order to pair the user's device with the selected plug, *Safecar* uses the core *Bluetooth Adapter* to interface with the physical device plugged in the car.

3.5.4 Geolocalization core service

During the trip, *Safecar* periodically samples location data in order to be able to precisely track the trip's route.

3.5.5 Realm real-time database

In order to ensure consistency, *Safecar* uses two databases services, one in the cloud and the other local. Realm is used to ensure database queries to be fast, efficient and precise.

3.5.6 Gilde image loader

In order to implement the feature of loading profile images from disk, the application uses a third-party set of APIs, Gilde.

3.5.7 Picasso image cropper

To provide fancy cropping features to the profile images, *Safecar* exploits the power of another third-party set of APIs, Picasso.

3.6 Algorithms

The application wraps the customizable algorithms into pluggable blocks. It is then possible to change those blocks with different and maybe more complex algorithms.
The important fact is that the new algorithm will respect the following interface:

- **ALG1**: It takes as an input the MAC address of the current plug and returns the current DSI. Now we can face two cases:

- The application is run on a device paired with a real plug. In this case *ALG1* has to:
  1. Register itself to the plug
  2. Connect with the plug
  3. Periodically ask the plug for some kind of data
  4. Clean and parse the data and put them into a custom data structure
  5. Pass the parsed data structure to *ALG2*
  6. Get the DSI from *ALG2*
  7. Pass the DSI to *ALG3*
- The application is run on a device that has no access to a real plug. In this case *ALG1* has to:
  1. Stash the current input
  2. Call *ALG2*
  3. Get the DSI from *ALG2*
  4. Pass the DSI to *ALG3*

- **ALG2**: It takes as an input the parsed data from *ALG1* and returns a DSI. It can be of two types:
  1. Memory based:
     - It computes the current DSI from the current input
     - It has to store all the previous DSI scores into a data structure and has to generate the current DSI by combining them in some way. An example can be to construct a discounted weighted average of the DSI over time to generate a value of the DSI that could be related to the current sample but also not blind with respect to the past driving style
  2. Blind:
     - It computes the current DSI only from the current data structure in input

- **ALG3**: It is an agent that takes as input the computed DSI and returns an hint string. It can be viewed as an agent that takes as input a percept and returns a decision. It can be:

    1. Simple reflex: It matches the percept on a rule set and returns the decision related to the matched rule
    2. Model based reflex: It stored the percept sequence and, through a model, matches a rule
    3. Goal based: Based on the current percept, on the model and on a goal specification, it returns the best hint to reach the goal. In this case the DSI computation can be skipped and this algorithm can work on the factored representation returned by *ALG1*. This algorithm is a tree search or a graph search.

Chapter 4

User Interface Design

Following are some mockups that give an idea of the structure of the application pages.

4.1 Splash Screen and Registration & Login Pages

These mockups show an example of the *Splash Screen* and *Registration & Login* pages of the application. The *Registration & Login Page* allow the user of the application to register as a user of the service, via the custom application functionality or external providers.

Figure 4.1: Splash Screen

Figure 4.2: Login

4.2 Home Page

These mockups give an idea of the application*Home Page*. The *Home Page* presents a *TabView* through which the user can navigate in order to inspect his history trips and a *Navigation Drawer Menu* through which the user can reach other pages of the application.

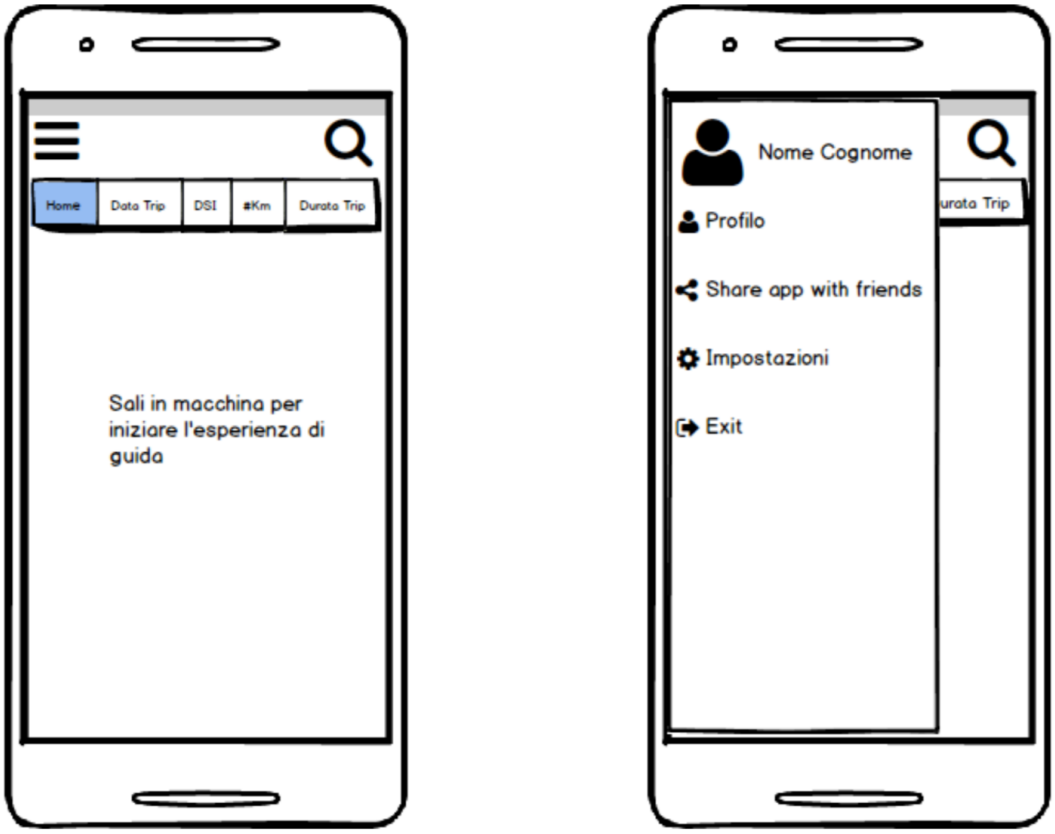


Figure 4.3: Home Page

4.3 During The Trip Page

These mockups show an idea of the *During Trip Page* of the application. The *During Trip Page* presents a *View* where *Hints* produced by the application are loaded and viewable by the user. The user can interact with the application using two buttons: the first is a *Pause/Resume* button which allow the user to pause the trip, without stopping it, if he wants to take a break from the driving session and, then, resume it; the second, instead, is a *Stop* button which allows the user to end the driving experience.

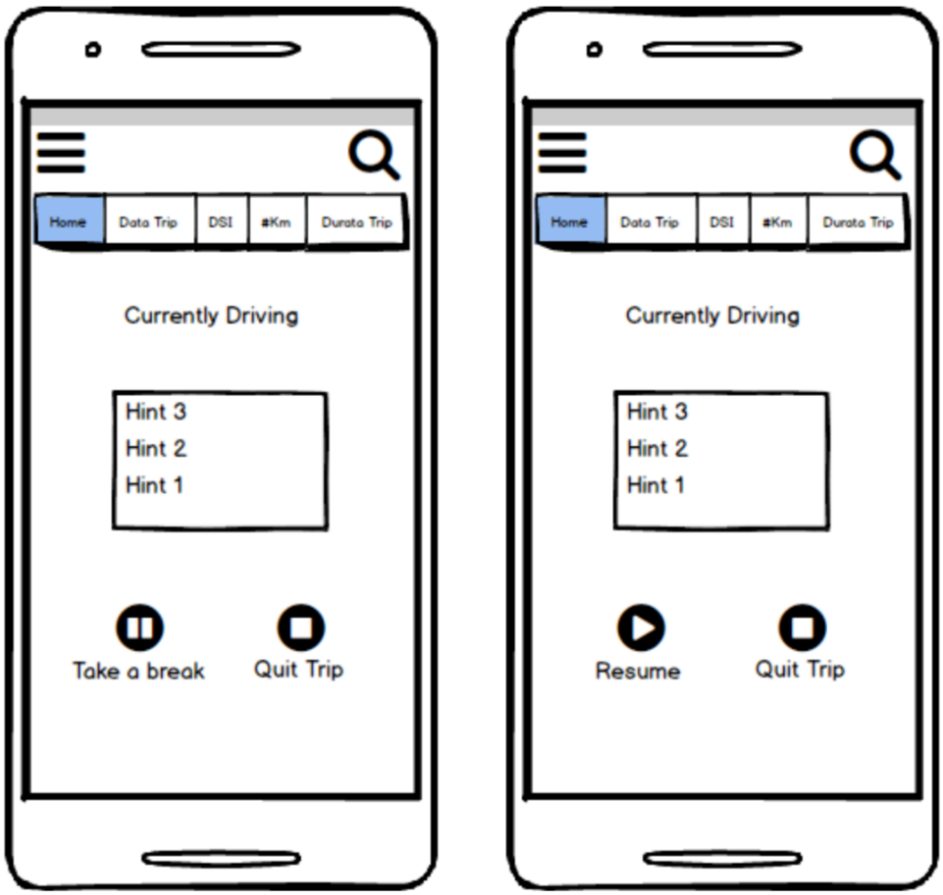


Figure 4.4: During Trip

4.4 Report Page

This mockup shows an idea of the *Report Page* of the application. The *Report Page* displays when the user inspects one of his history trips or when he completes a driving session pushing the *Stop* button. This page allows the user to see all the details about his/her trip including data like: the trip's date, duration and length, the DSI score and the path he/she made.

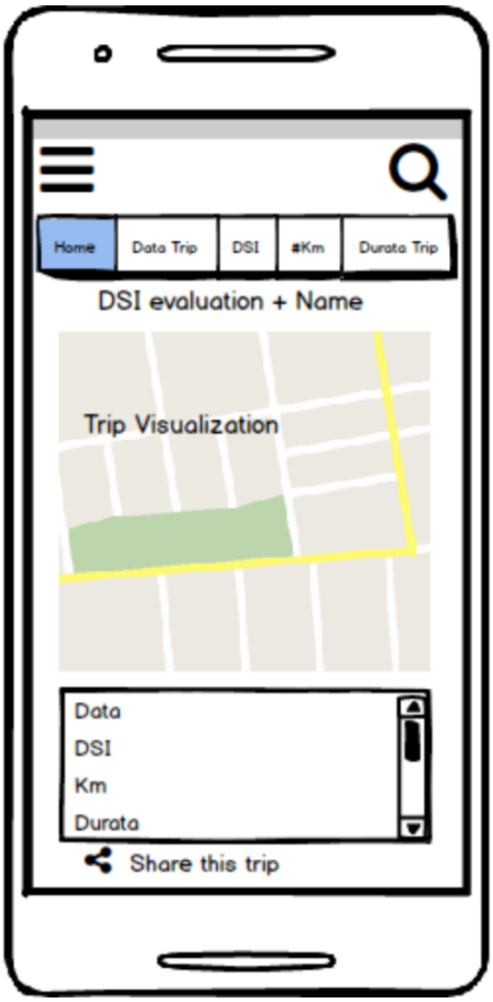


Figure 4.5: Report Page

4.5 Profile Page

These mockups show an idea of the *Profile Page* of the application. The *Profile Page* shows information about the user like his/her name, surname, email address, driver level and also some badges that he/she can unlock meeting specific conditions. Each badge is clickable and let the user know about its locked/unlocked status.

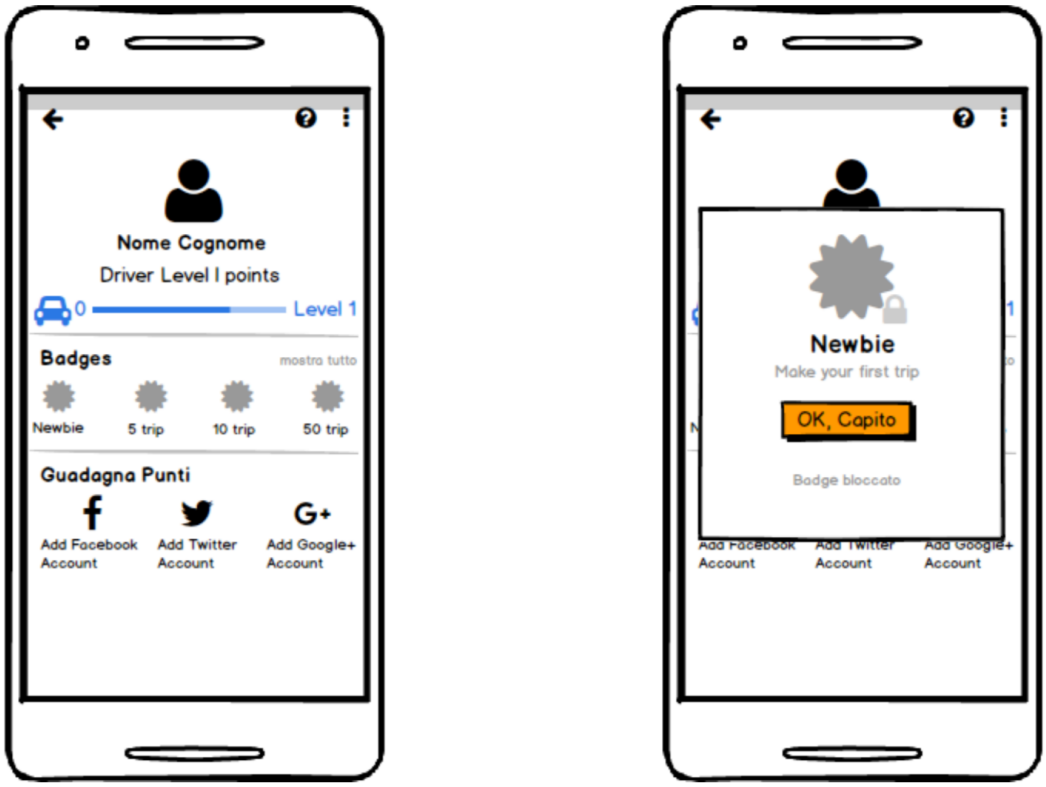


Figure 4.6: Profile Page

4.6 Settings Page

This mockup shows an idea of the *Settings Page* of the application. The *Settings Page* allows the user to manage some settings like push notifications and smart objects. This page also allows the user to see some info about the application and send feedback in order to improve it.

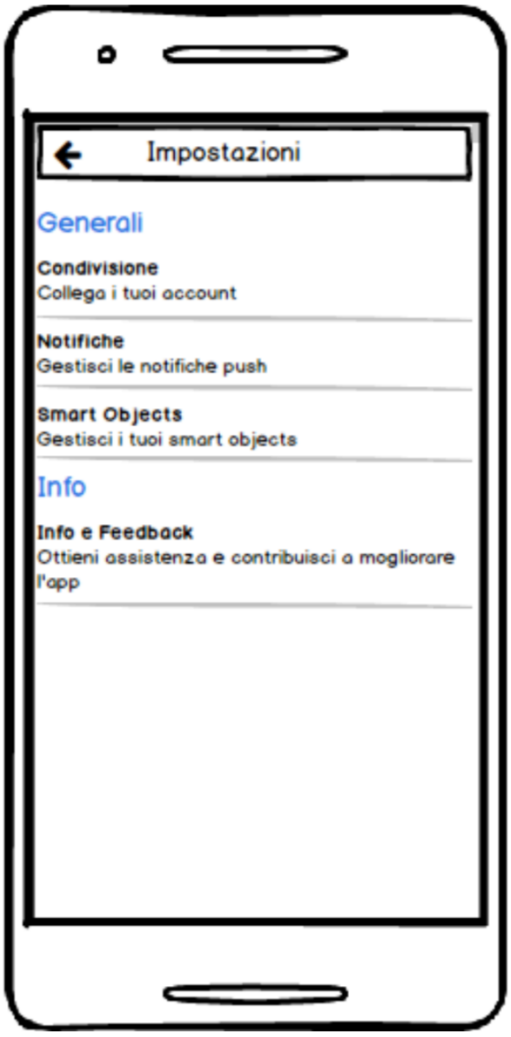


Figure 4.7: Settings Page

4.7 Smart Objects Page

These mockups give an idea of the *Smart Objects Pages* of the application. The *Smart Objects Pages* allow the user to manage smart objects necessary for the application. Using a bluetooth scanner the user can pair his/her device with a plug, used to retrieve relevant information about the current driving session and he/she can also manage all the paired plugs, inspecting or deleting them.

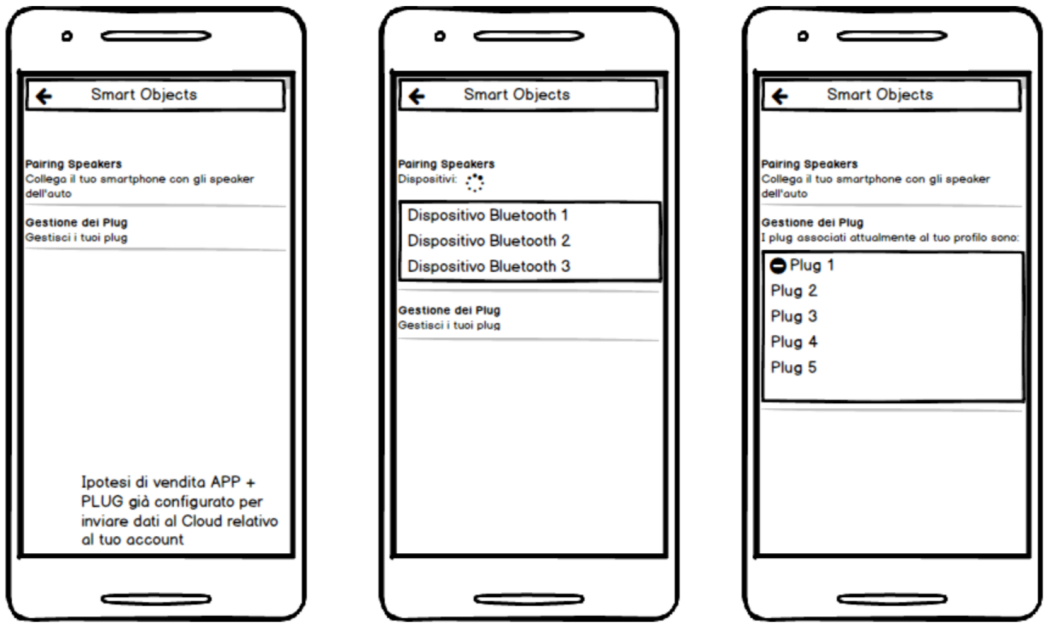


Figure 4.8: Smart Objects Pages

Chapter 5

Future developments

The application can be improved by integrating the use of a *text2speech* service that should have the role of reading the hints and send the relative audio to a speaker. This way the user will hear the hints during the drive.