PowerEnJoy Project

Design Document (DD)

Alfredo Maria Fomitchenko (mat. 874656)

Version: 1.1

Release date: 15 January 2017

1. Introduction

1.1 Purpose

This Design Document provides an in-depth view of the PowerEnJoy car sharing service system to be, addressing and illustrating the high-level architecture, the main components, the runtime behavior and the customer's user experience provided by the associated mobile application.

1.2 Scope

As in the RASD, this project aims to provide customers within the administrative division of Milan with a car sharing service and all the associated functionalities such as searching for an available car, reserving it for up to one hour, unlocking it and getting a discount thanks to virtuous behavior.

1.3 Definitions, acronyms, abbreviations

In extension to the RASD Definitions, acronyms, abbreviations paragraph, below are some acronyms used in this document:

- "API" = acronym for Application Programming Interface, communication channel by which a system allows intercommunication with other systems
- "URL" = acronym for Uniform Resource Locator
- "REST" = acronym for REpresentational State Transfer, it represents a way of providing interoperability between system on the Internet. Within the scope of this project, it is associated to HTTP requests between the mobile application and the application server logic layer
- "PHP" = recursive acronym for PHP Hypertext Preprocessor, scripting language used within the scope of this project as a communication means between the application server logic layer and the database layer
- "PDO" = acronym for PHP Data Objects, PHP extension that provides abstraction database layer interface functionalities
- "HTML" = acronym for HyperText Markup Language, markup language used in this project to represent the available cars markers onto a map inside the mobile application
- "JSON" = acronym for JavaScript Object Notation, lightweight format for data exchange
- "RDBMS" = acronym for Relational DataBase Management System, basis for SQL and MySQL database systems

1.4 Reference documents, used tools and resources

I used for this DD as reference for the project assignment, the general layout and structure

- Assignments AA 2016-2017.pdf,
- RASD v1.1 Alfredo Fomitchenko.pdf,
- Sample Design Deliverable Discussed on Nov. 2.pdf;

to create the UML diagrams

- draw.io website;

to mock up the user interface

- Adobe Photoshop CC 2015;

to write the actual document

- Microsoft Word 2016;

as scheduling and time effort management tracker

- Microsoft Excel 2016.

1.5 Document Structure

1. Introduction: this section presents the general purpose of the document, its structure and utility.

2. Architectural Design:

2.1 Overview: the division into tiers is presented in this section

2.2 High level components and their interaction: the general components and communication among them are presented in this section

2.2 Component view: the system components and the communication among them is presented in this section

2.3 Deployment view: the system hardware topology and components distribution is presented in this section

2.4 Runtime view: the interactions among components to fulfill customer's requests is presented in this section

2.5 Component interfaces: components interfaces are presented in this section

2.6 Selected architectural styles and patterns: overall architectural choices and general design patterns are presented in this section

2.7 Other design decisions: any other design concern is presented in this section

3. Algorithm Design: pseudocode giving an algorithmic view of the most critical customer's requests is presented in this section

4. User Interface Design: Android mobile application mockups are presented in this section

5. Requirements Traceability: a map among system components and RASD requirements is presented in this section

2. Architectural Design

2.1 Overview

Four distinct elements compose the system to be:

1) Customer: representation of the customer's smartphone onto which the mobile application is installed. The mobile application gives the customer access to the functionalities offered by the service by communicating with the Application server via the smartphone's Internet connection. Also, the customer is able to unlock the car via a Bluetooth interface and display the ride charge timer onto the car screen via a standard USB interface.
   This and the car represent the *first* tier.

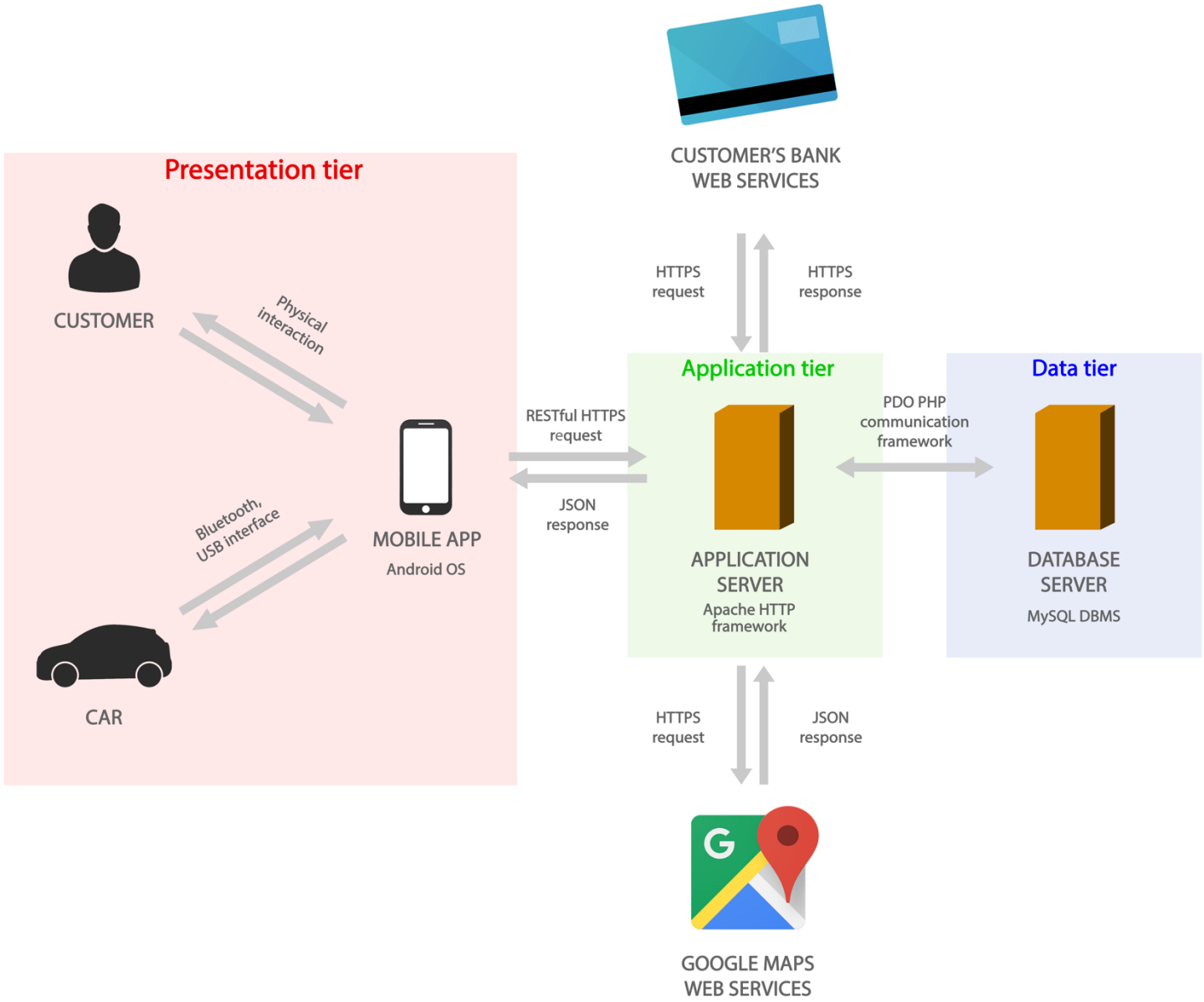
2) Car: representation of the cars Android Auto operating system onto which a pre-installed service necessary for the unlocking process is present.
   This and the customer represent the *first* tier.

3) Application server: focal provider of the service. It receives the customer's requests (in the diagram Customer Mobile Application) and gives back answers based on the world's information stored in the database (in the diagram Database server).
   This represents the *second* tier.

4) Database server: customers' and cars information management and storage. Data are not directly accessed, but cached inside Application server.
   This represents the *third* tier.

In addition, in the diagram there are also represented the external actors involved in the process of providing the service:

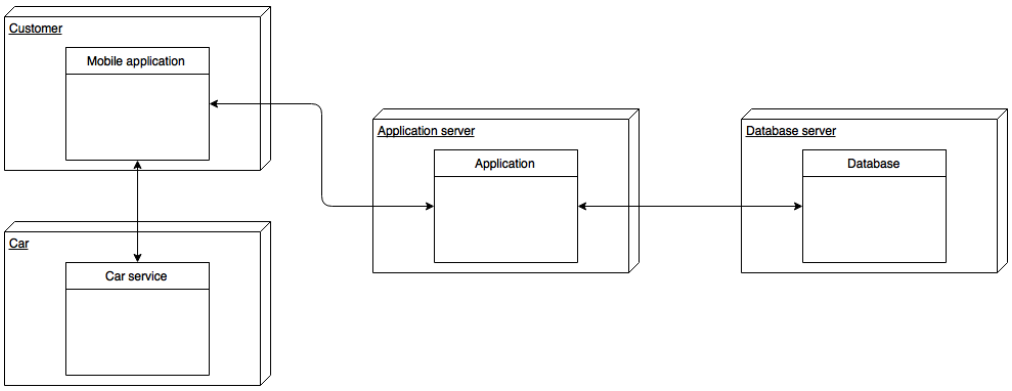
- Google Maps: representation of the Google Maps Web Services. In particular within the scope of this project I considered
    - GeoCoding API for the conversion of a string of text into a GPS position,
    - GeoLocation API for the GPS localization of devices,
    - Google Maps Javascript API for the representation of the cars positions onto a map.
- Email Service Provider: customer's email service provider by which the system sends the customer his automatically generated password. Not a single provider has been chosen and the abstraction is legit since nowadays the effort of integrating and exploiting the email protocol is practically negligible.
- Bank: customer's bank Internet interface. Not a single bank has been chosen and the abstraction is legit since nowadays any bank is provided with an Internet interface to allow authorized external entities to automatically ask the bank customers for charges.


Presentation tier
CUSTOMER
Physical interaction
Bluetooth, USB interface
MOBILE APP
Android OS
CAR
CUSTOMER'S BANK WEB SERVICES
HTTPS request
HTTPS response
Application tier
RESTful HTTPS request
JSON response
APPLICATION SERVER
Apache HTTP framework
PDO PHP communication framework
Data tier
DATABASE SERVER
MySQL DBMS
HTTPS request
JSON response
GOOGLE MAPS WEB SERVICES

2.2 High level components and their interaction

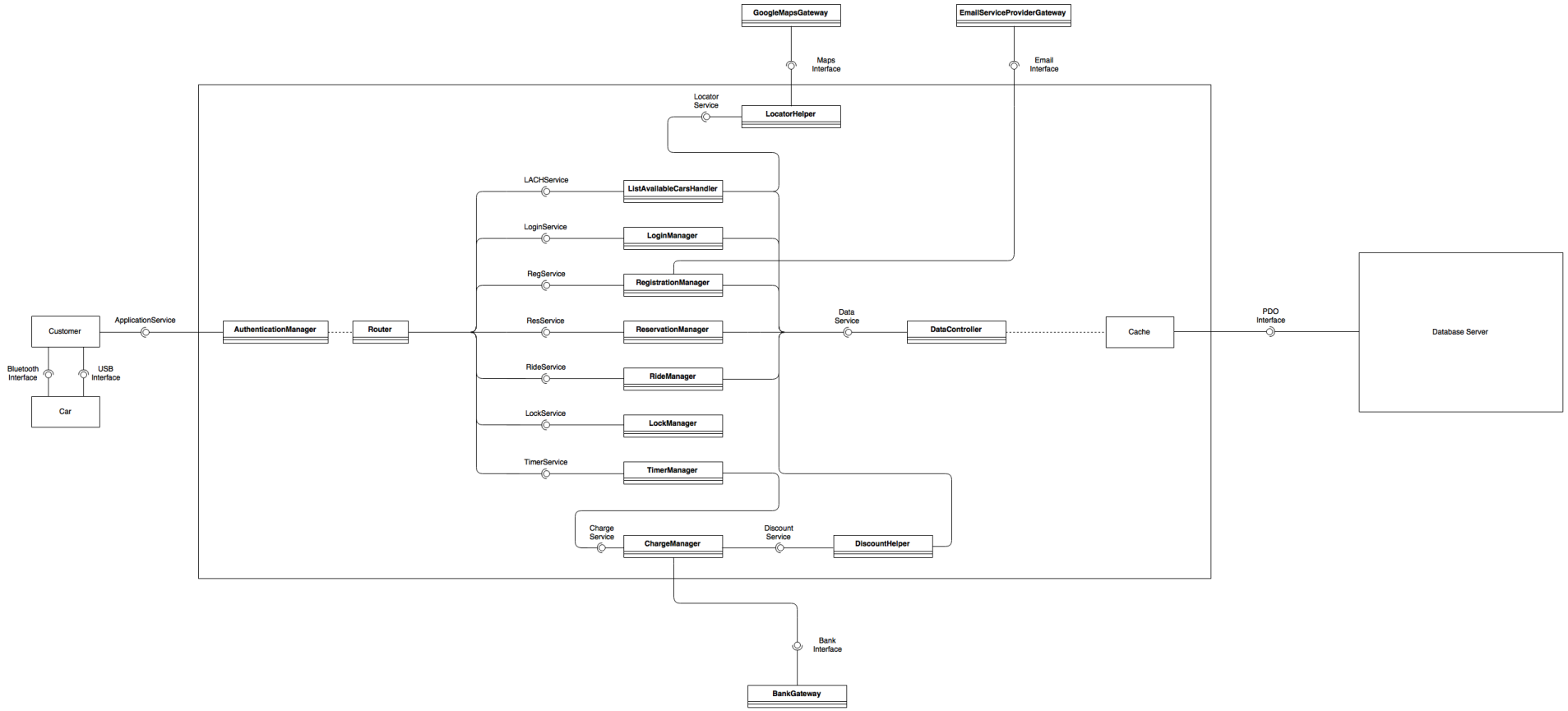
The previous overview naturally leads to the high level components view, in which the first tier is composed by the customer mobile application and the car service inside the Android Auto operating system of the car. They communicates via two distinct interfaces, a Bluetooth and a USB one. The mobile application interacts with the second tier via RESTful HTTP requests and the second communicates with the third via PDO interface.

See Component view paragraph for further details.



2.3 Component view

Please find the high-resolution diagram at the following link:
https://github.com/alfredo-f/electric-car-sharing/blob/master/releases/DD/Diagrams/PEJ-Composite.png


GoogleMapsGateway
EmailServiceProviderGateway
Maps Interface
Email Interface
Locator Service
LocatorHelper
LACHService
ListAvailableCarsHandler
LoginService
LoginManager
RegService
RegistrationManager
ResService
ReservationManager
RideService
RideManager
LockService
LockManager
TimerService
TimerManager
Charge Service
ChargeManager
Discount Service
DiscountHelper
Bank Interface
BankGateway
Customer
ApplicationService
AuthenticationManager
Router
Bluetooth Interface
USB Interface
Car
Data Service
DataController
Cache
PDO Interface
Database Server

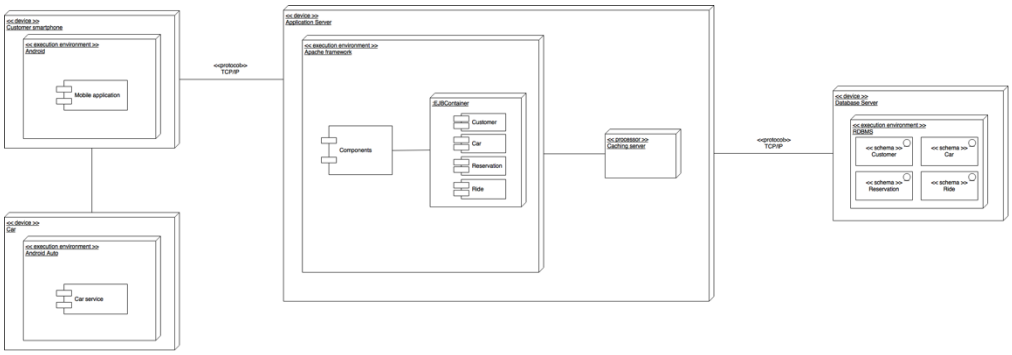
- Customer: the customer's smartphone application.
- Car: the pre-installed car service inside the Android Auto operating system responsible of the unlocking capabilities accessible via the Bluetooth interface and validated by the system via LockManager (see Runtime view paragraph).
- BluetoothInterface: the Bluetooth wireless interface enabling the customer to unlock the car by connecting the smartphone application to the car service.
- USBInterface: the standard USB physical interface by which the customer connects the smartphone to the car connectors and is able to display onto the car screen the current charge timer.
- ApplicationService: interface by which both the customer's smartphone application and the car service communicate to the system relying upon the customer's smartphone Internet connection.
- AuthenticationManager: security component that validates the requests coming into the system.
- Router: routes the requests validated by AuthenticationManager to the appropriate system component.

- RegistrationManager: manages the new customer's registration process, generating, storing and sending the password to the customer by means of EmailServiceProviderGateway.
- LocatorHelper: manages the requests of locating a customer or a car by means of GoogleMapsGateway.
- GoogleMapsGateway: Google Maps API providing the services which enable the system to locate devices and convert a string of text provided by the customer into a location.
- ListAvailableCarsHandler: focal component that manages the requests of retrieving the list of available cars starting from the customer's location, provided by the smartphone GPS location or as a string to be converted.
- DataController: checks on the cars availability and the customer's possibility to reserve a car; modifies the cars availability.
- ReservationManager: manages the car reservation requests.
- RideManager: manages the ride starting and stopping requests.

- LockManager: manages the car locking and unlocking requests.
- TimerManager: manages the timer starting and stopping requests.
- ChargeManager: manages the charge requests coming from TimerManager triggered by timer stopping requests. It is supported by DiscountHelper in the computation of the final charge applicable to the customer. It requests the final charge to BankGateway.
- DiscountHelper: supports ChargeManager in the computation of the customer's final charge according to the applicable discounts based on the policies described in the RASD.
- BankGateway: generic bank API which enables the system to charge customers according to ChargeManager requests.
- Model: representation of the world based on the database information.
- MapsInterface: Internet interface providing a standard HTTPS requests handler.
- EmailInterface: Internet interface providing a standard HTTPS requests handler.
- BankInterface: Internet interface providing a standard HTTPS requests handler.

2.4 Deployment view

Please find the high-resolution diagram at the following link:
https://github.com/alfredo-f/electric-car-sharing/blob/master/releases/DD/Diagrams/PEJ-Deployment.png



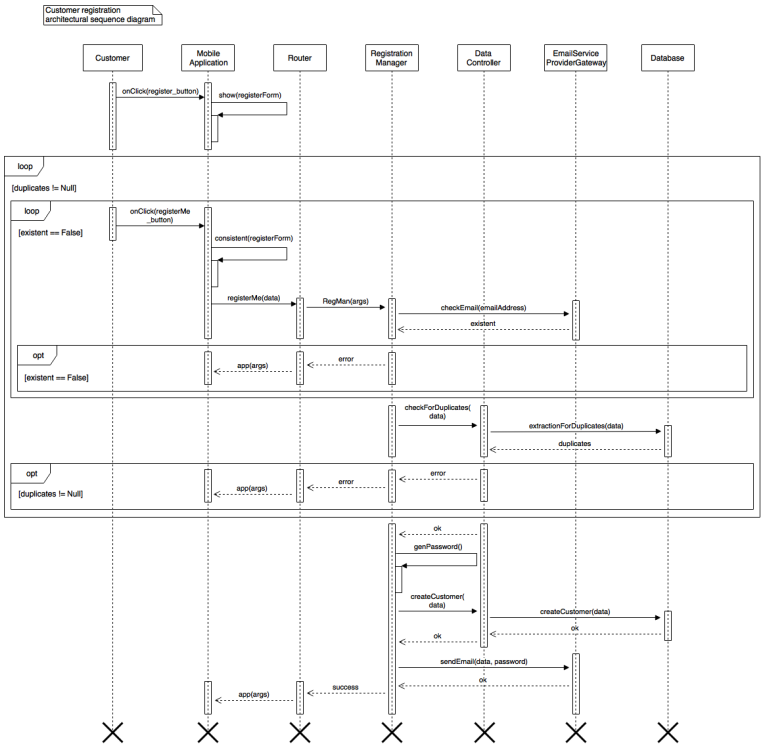
2.5 Runtime view

Interactions between the system components to fulfill customers' requests are illustrated in this section.

For the sake of legibility, the security component AuthenticationManager and its related input requests authentication processes and redirections to Router have been omitted, the Cache database synchronization requests likewise.

2.5.1 Customer registration architectural sequence diagram

Please find the high-resolution diagram at the following link:
https://github.com/alfredo-f/electric-car-sharing/blob/master/releases/DD/Diagrams/PEJ-Sequence-Register.png



This sequence diagram illustrates the interaction among the system components to fulfill the customer's request to register.

After showing the registration form and verifying the consistency of the data inserted by the customer (e.g. the email address shows the "@" character), Router forwards this form received from the mobile application to RegistrationManager.

This component needs to check whether:

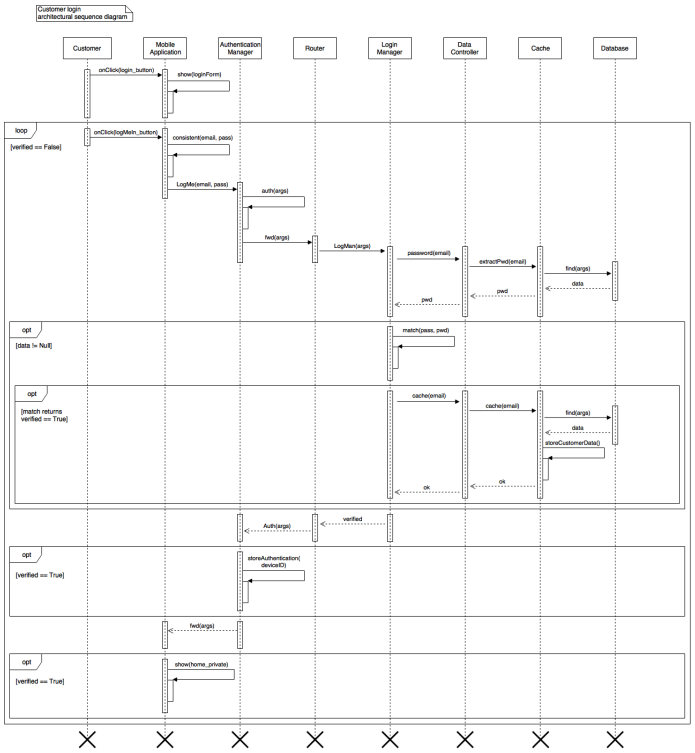
1) the email address exists and is valid, otherwise the customer cannot receive the password to access the system and this would lead to a waste of database resources. This is performed thanks to EmailServiceProviderGateway;
2) the customer has not already registered his account. DataController is in charge of this verification.

If both checks hold, RegistrationManager generates the customer's password, asks DataController to create the new customer profile inside the database and sends a confirmation email via EmailServiceProviderGateway.

Once done, the mobile application is instructed by RegistrationManager to display a confirmation message to the customer.

2.5.2 Customer login architectural sequence diagram

Please find the high-resolution diagram at the following link:
https://github.com/alfredo-f/electric-car-sharing/blob/master/releases/DD/Diagrams/PEJ-Sequence-Login.png



This sequence diagram illustrates the interaction among the system components to fulfill the customer's request to log into the system.

For this sequence diagram, the security component AuthenticationManager and its related input requests authentication processes and redirections to Router have not been omitted, the Cache database synchronization requests likewise, as for this particular process they do not perform trivial omissible operations.

After showing the login form and verifying the consistency of the data inserted by the customer (e.g. the email address shows the "@" character), AuthenticationManager checks on the integrity and security of the communication coming from the official mobile application, and passes the request onto Router, which forwards to LoginManager.

This component has to:

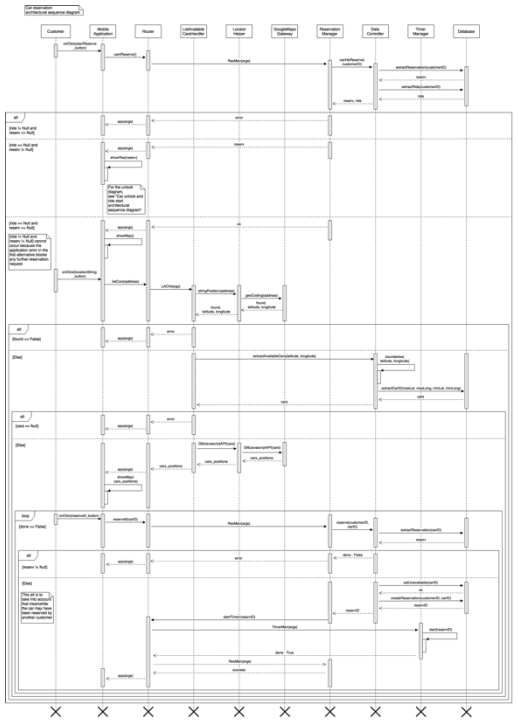
1) extract from the database the password associated to the email address provided by the customer in the login form. This request, passed onto DataController, is forwarded to Cache, which performs the call onto the database and returns the extracted data with trivial calls. If the data is Null, there are no records for the provided email address in the database and LoginManager returns to the mobile application an error;

2) match the password provided by the customer in the login form (*pass*) with the password extracted from the database (*pwd*).

When the match occurs, after receiving the call from Router, AuthenticationManager associates to the customer and stores all the necessary information about the device that received in the first incoming login request: in this way the customer is fully logged into the system and ready to gather personal information from the *home_private* screen shown by the mobile application.

2.5.3 Car reservation architectural sequence diagram

Please find the high-resolution diagram at the following link:
https://github.com/alfredo-f/electric-car-sharing/blob/master/releases/DD/Diagrams/PEJ-Sequence-Reserve.png



This sequence diagram illustrates the interaction among the system components to fulfill the customer's request to see and reserve an available car near his position.

First and foremost the mobile application needs to know if the customer tapped on the "Reservation" button either to display his current one and unlock the car (see Car unlock and ride start architectural sequence diagram) or to reserve one if he is able to (according to "*The system shall verify if a customer neither has previously reserved nor is currently using a car.*" requirement). This request forwarded by the Router to ReservationManager is processed and fulfilled by DataController, the component that works as an interface between the other ones and the information about reservations, rides and availabilities inside the database: it extracts reservation and ride associated to the customer via his customerID originally provided by the application itself, and returns them to ReservationManager.

If a ride is present, the component returns an error all the way back to the application, for which there is no need to show the map and it thus blocks any further reservation requests.

If a reservation is present, the component returns it to the mobile application, which then displays it to let the customer unlock a car if he is nearby.

If neither apply, the customer is able to reserve a car and the application shows an empty map (retrieved independently from the system itself through the smartphone built-in set of Google Maps API) where the customer is being asked to insert either an address or to provide his current position from which to look for available cars.

For this sequence diagram the former case has been considered: the customers inserts an address, pushes the locationString_button and the Router forwards the application request to ListAvailableCarsHandler. This component asks for:

1) the conversion of the string of text into a pair (latitude, longitude) fulfilled by GoogleMapsGateway via GeoCoding API. If the provided address is not valid (i.e. found is equal to False) ListAvailableCarsHandler requests the application to display an error message;
2) the extraction from the database of the available cars within a certain range. Latitude and longitude boundaries computation and the actual extraction from the database are performed by DataController, which returns a list of cars. If it is empty (i.e. cars is Null) ListAvailableCarsHandler requests the application to display an error message;
3) the representation of the cars positions onto the map fulfilled by GoogleMapsGateway via Google Maps Javascript API.

The communication with Google Maps API is considered available according to "*The system is able to communicate with Google Maps service and finalize requests such as managing GPS positions of customers and cars and converting a string of text into a location.*" assumption, so that within this sequence diagram GoogleMapsGateway is considered reachable.

Once done, ListAvailableCarsHandler returns the cars positions to the application, and the application shows them onto the map. The customer is then able to tap on a car to display and press the associated reserveIt_button, which makes a call through the Router to ReservationManager. This component has to:

1) verify if the selected car is still available, as the customer may have lost the opportunity because another one reserved it between the extraction of the available cars list and the reservation request. This is done by DataController: if the reservation extraction for that particular car does not yield any, ReservationManager requests the application to display an error message;

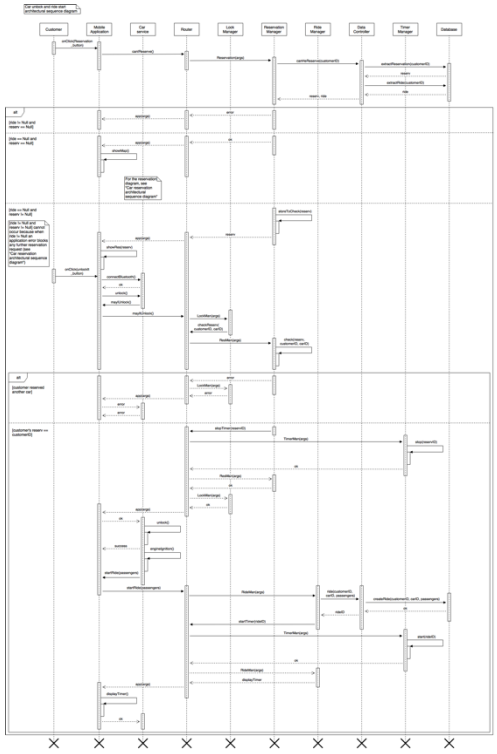
2) if available, ReservationManager

    2.1) sets it unavailable via DataController,
    2.2) creates into the database the reservation association between carID and customerID,
    2.3) makes TimerManager start the associated reservation timer.

The process is now complete: ReservationManager requests the application to display a success message.

2.5.4 Car unlock and ride start architectural sequence diagram

Please find the high-resolution diagram at the following link:
https://github.com/alfredo-f/electric-car-sharing/blob/master/releases/DD/Diagrams/PEJ-Sequence-Unlock.png



This sequence diagram illustrates the interaction among the system components to fulfill the customer's request to unlock a previously reserved car with which then start a ride.

As seen in Car reservation architectural sequence diagram, first the customer taps onto the Reservation button, but this time ReservationManager receives a not Null reservation from DataController. The component stores and sends the reservation back to the mobile application: this way it knows it has to show the reservation details and the associated button to unlock the vehicle instead of the map onto which the customer was meant to reserve a car.

After tapping the unlockIt_button, the application shows a message asking for the permissions to access Bluetooth capabilities and then to switch it on (according to "*Customers willing to start a ride make sure they have the necessary smartphone Internet and Bluetooth connection capabilities and sufficient battery charge to correctly terminate the ride.*" assumption). Once done if the customer's smartphone is sufficiently close the application is now able to connect to the car Bluetooth interface and demand for the automated unlock.

This request must be of course forwarded to and authorized by the system via a process securely encrypted and transparent to the customer. LockManager is the component that handles this call: it asks ReservationManager to verify that the car requesting the unlock verification matches the one in the previously extracted reservation associated to the customer; if not, ReservationManager and consequently LockManager return an error all the way back to the car, which does not perform the unlock.

If the condition holds, ReservationManager asks via Router TimerManager to stop the reservation timer, and once correctly stopped it confirms to LockManager that the car is authorized to unlock the doors, requests sent to the car service through BluetoothInterface.

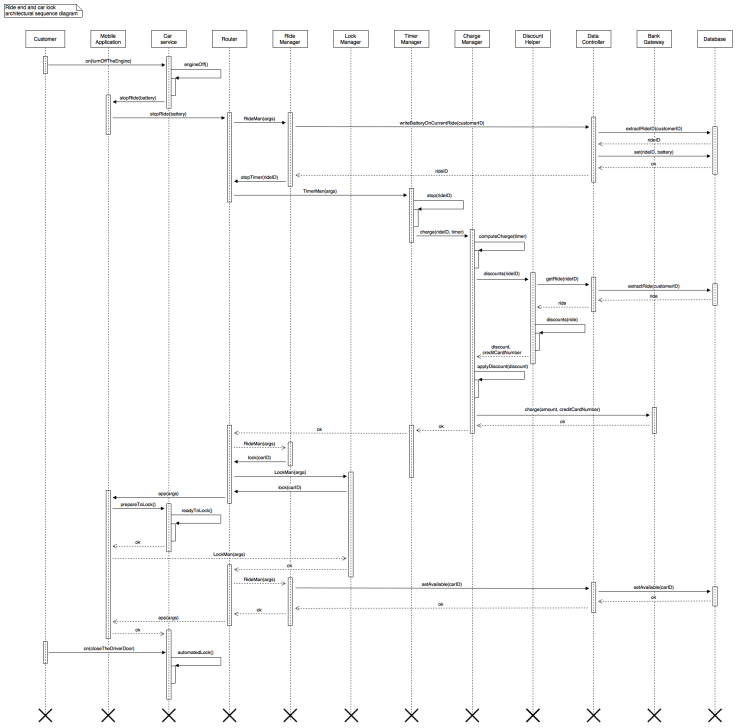
Now customer and passengers enter the car and the customer connects his smartphone to the car standard USB connectors, enabling the mobile application to be displayed onto the screen next to the dashboard and to further communicate with the system (according to "*Customers unlocking a car actually use it.*", "*Customers willing to start a ride enter and let any passengers enter the car after unlocking it and before igniting the engine.*" and "*After entering the car, customers plug their smartphones into the car connectors.*" assumptions).
At the engine ignition, the car gathers information about the seat sensors activated by the passengers and sends it via Router to RideManager (according to "*Customers' behavior to the passengers discount policy is fair as they do not attempt to fool the car seat sensors into detecting passengers. In this respect, customers carry any package in the trunk, so that an active seat sensor implies the presence of a passenger.*" and "*Customers' behavior to the passengers discount policy is fair, i.e. the number of passengers carried by the customer doesn't change throughout a ride.*" assumptions).
RideManager:

1) creates the ride association inside the database thanks to DataController;
2) makes TimerManager start the associated ride timer.

Received confirmation, RideManager instructs the application to display said timer, allowing the customer to keep track of the ride charges.

2.5.5 Ride end and car lock architectural sequence diagram

Please find the high-resolution diagram at the following link:
https://github.com/alfredo-f/electric-car-sharing/blob/master/releases/DD/Diagrams/PEJ-Sequence-Lock.png



This sequence diagram illustrates the interaction among the system components to fulfill the customer's request to terminate a ride and therefore lock the car.

The entry condition is that the customer has driven to and stopped the vehicle within allowed parking lines belonging to a safe area (according to "*Customers do not leave in a safe area their in usage cars unless they are willing to terminate the ride.*" and "*Customers willing to terminate a ride exit the car within one minute after parking it inside allowed parking lines belonging to a safe area.*" assumptions).

When the customer turns the engine off, the car service communicates to Router via USBInterface that the customer is willing to terminate the ride (according to "*Customers plug their smartphones out of the car connectors after turning the engine off, otherwise they are not able to correctly terminate the ride.*" assumption). Forwarded the request to RideManager, this component has to

1) store the car battery percentage into the database ride information and retrieve the rideID associated to the customer (task performed by DataController) and
2) stop the timer associated to the ride.

The latter request, forwarded by Router to TimerManager, triggers the call to ChargeManager. This component receives from TimerManager the rideID and the total ride time, by which it computes the temporary charge amount. It then needs to apply the applicable discounts, so rideID is forwarded to DiscountHelper, which

1) retrieves via DataController all the necessary information about the ride, and
2) computes the total applicable discount,

returned to ChargeManager. This component applies the discount to the initial amount of money, and thanks to BankGateway charges the customer.

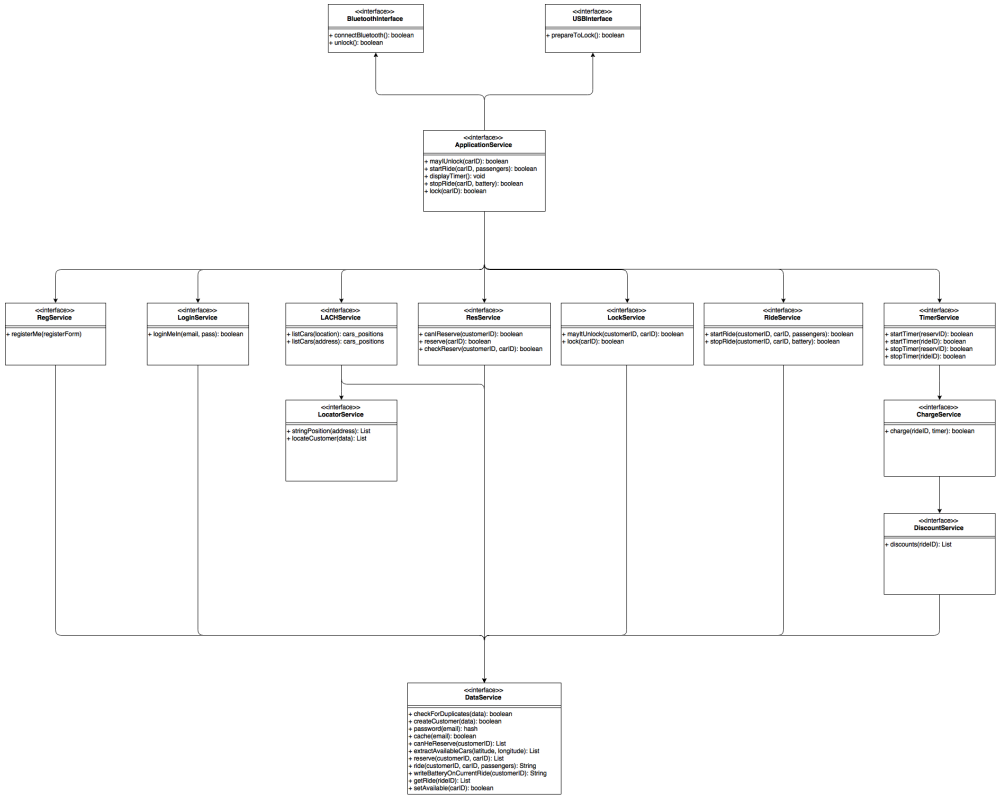
Being confirmed of its request success, TimerManager involves LockManager in the process, asking for the car automated lock. This final call is forwarded by the mobile application to the car service, which prepares the car doors for the automated lock, performed when the customer finally closes the driver door.

2.6 Component interfaces

ApplicationService, the mobile application interface by which the car service via BluetoothInterface and USBInterface and the customer himself interact with the car sharing service, represents the entry point for the communication to the various system component, validated by AuthenticationManager and redirected by Router.

The following diagram represents the component interfaces as previously highlighted and named in the Component view.

Please find the high-resolution diagram at the following link:
https://github.com/alfredo-f/electric-car-sharing/blob/master/releases/DD/Diagrams/PEJ-Interfaces.png



2.7 Selected architectural styles and patterns

The overall architecture is divided into three tiers:

1) Mobile application layer
2) Application server logic layer
3) Database layer

Communication between the mobile application layer and the application server logic layer (clear representation of a client-server pattern) is provided by the Internet through basic GET RESTful HTTP requests and responses encoded by the application itself and authorized by AuthenticationManager component (see Component view paragraph), and JSON responses for some requests such as retrieving the list of available cars near a location (see Car reservation architectural sequence diagram).

Communication between the application server logic layer and the database layer is managed by a PHP Data Objects (PDO) extension, which represents a standard interface for accessing MySQL databases as in this case with an improved support for object-oriented programming.

2.8 Other design decisions

As highlighted in the following section of this document, the application server logic layer will fully embrace HTML and Javascript support.

This is due to the fact that a fundamental functionality of the system, the possibility to retrieve and represent onto a map the available cars within an area, needs the support of Google Maps Javascript API to fully and easily exploit the capabilities that Google provides within the scope of its map service.

3. Algorithm Design

The system needs to fulfill different type of requests coming from the customer. Among them, the most suitable to show an algorithmic view of the system flow of calls is the one associated to a car reservation.

3.1 Car reservation algorithm

The following pseudocode shows main Java peculiarities with Javascript, HTML and PHP blends when indicated.

The first section is intended as a glimpse of the mobile application implementation.

```
// Activity for displaying the map
import com.google.android.maps.MapActivity;

public class HomeView extends Activity {
	public void onCreate(Bundle savedInstanceState){
		super.onCreate(savedInstanceState);

		Button startReserve_button = (Button)findViewById(R.id.reservation_button);
		startReserve_button.setOnClickListener(new View.OnClickListener() {
			public void onClick() {
				response = canIReserve();

				if (response == "ErrorRide") {
					AlertDialog.Builder alert = new AlertDialog.Builder(this);
					alert.setTitle("Error");
					alert.setMessage("Please correctly end the current ride before reserving another car");
					alert.create().show();
				}

				else if (response instanceof Reservation) {
					Intent intent = new Intent(this, DisplayReservationActivity.class);
					StartActivity(intent);
				}

				else if (response == "OK") {
					Intent intent = new Intent(this, ReservationMap.class);
					StartActivity(intent);
				}
			}
		}
	}
}
```

```
private class ReservationMap extends MapActivity {
    // End in itself Java to create the Activity (see HomeView introductory code)
    function initMap() {
        // Javascript pseudocode to call and manage Google Maps Javascript API
        // Omitted HTML code. It would include
        // <input id="input-box" class="controls" type="text" placeholder="Insert an address">
        // <div id="location_icon_div"></div>
        // <div id="map"></div>

        // Create the map

        var map = new google.maps.Map(document.getElementById('map'));
        var input = document.getElementById('input-box');

        var searchBox = new google.maps.places.SearchBox(input);
        searchBox.addListener('submit', function() {
            var cars_positions = listCars(searchBox.text);

            if (cars_positions == "ErrorAddress") {
                AlertDialog.Builder alert = new AlertDialog.Builder(this);
                alert.setTitle("Error");
                alert.setMessage("Address not found. Please double-check the provided information");
                alert.create().show();
            }
```

```
            else {

                // Add the markers to the map
                // Array.prototype.map() creates an array of markers
                // starting from locations based on the syntax shown below
                var markers = cars_positions.map(
                    function(location, i) {
                        marker = new google.maps.Marker({
                            position: location,
                            label: labels[i % labels.length]
                        });

                        var popup = '<button onclick="reserveIt(" + carID + ")">';
                        var infoWindow = new google.maps.InfoWindow({
                                content: popup;
                        });
                        marker.addListener('click', function() {
                            infoWindow.open();
                        });

                    });
            }
        }
    }
}
```

```
function canIReserve() {
	// Router calls for canIReserve_ReservationManager(customerID)
	return callRouter("App", "ReservationManager", "canIReserve()", customerID);
}
function listCars(address) {
	// Router calls for listCars_LACH(address)
	return callRouter("App", "LACH", "listCars()", address);
}
function reserveIt(carID) {
	// Router calls for reserveIt_ReservationManager(customerID, carID)
	return callRouter("App", "ReservationManager", "reserveIt()", customerID, carID);
}
```

```
function canIReserve_ReservationManager(customerID) {
	response = canHeReserve(customerID);
	reserv = (Reservation)response[0];
	ride = (Ride)response[1];

	if (ride != Null) {
		return "ErrorRide";
	}
	else if (reserv != Null) {
		return "OK";
	}
	else {
		return reserv;
	}
}

function canHeReserve(customerID) {
	response[0] = (Reservation)extractFromDatabase("ReservationManager", "Reservation", customerID);
	response[1] = (Ride)extractFromDatabase("ReservationManager", "Ride", customerID);
	return response;
}
```

```
function listCars_LACH(address) {
	response = stringPosition(address);
	found = response[0];
	latitude = response[1];
	longitude = response[2];
	if (found == false) {
		return "ErrorAddress";
	}
	else {
		Car[] cars = extractAvailableCars(latitude, longitude);
		if (cars == Null) {
			return "ErrorCars";
		} else {
			return GMJavascriptAPI(cars);
		}
	}
}

function extractAvailableCars(latitude, longitude) {
	boundaries_array = boundaries(latitude, longitude);
	maxLat = boundaries_array[0];
	maxLong = boundaries_array[1];
	minLat = boundaries_array[2];
	minLong = boundaries_array[3];
	Car[] cars = extractFromDatabase("LACH", "carID", maxLat, maxLong, minLat, minLong);
	return cars;
}
```

```
function reserveIt_ReservationManager(customerID, carID) {
	response = reserve_DataController(customerID, carID);
	if (response == false) {
		return "ErrorReservation";
	} else {
		done = callRouter("ReservationManager", "TimerManager", "reserveIt_ReservationManager()", reservID);
		if (done == true) {
			return "OK";
		} else {
			return "Error";
		}
	}
}

function reserve_DataController(customerID, carID) {
	response = extractFromDatabase("ReservationManager", "Reservation", carID);
	if (response instanceof Reservation) {
		return false;
	} else {
		availability = writeOnDatabase("ReservationManager", "reserve_DataController()", "unavailable", false);
		reservID = writeOnDatabase("ReservationManager", "reserve_DataController()", "Reservation");
		return reservID;
	}
}
```

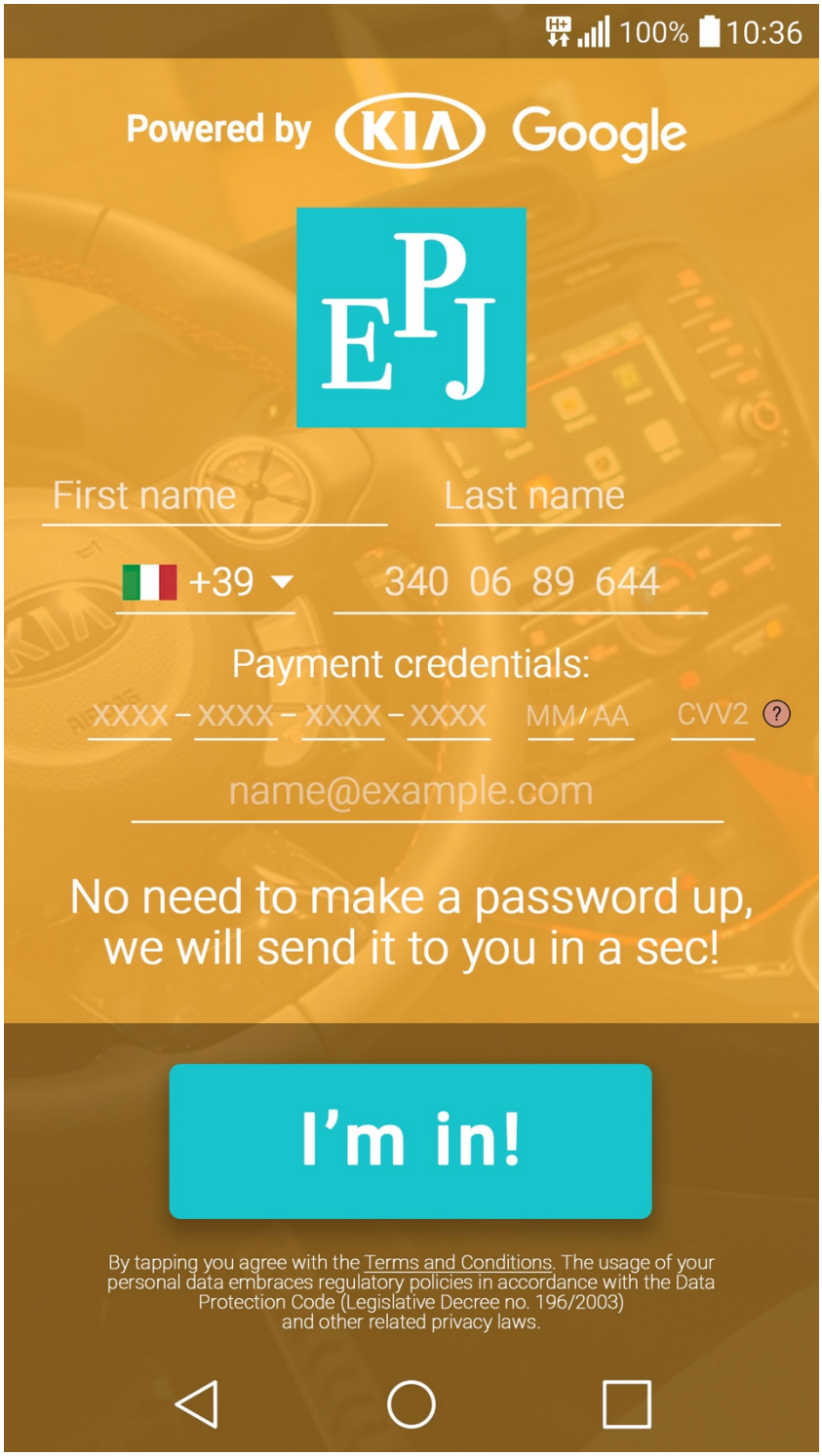
```
function extractFromDatabase(args[]) {
	// PHP pseudocode to extract from MySQL database
	$result = array();
	$con = mysqli_connect("host", "authentication_arguments");
		$sql_query = // based on args[] fetching
		$r = mysqli_query($con,$sql_query);
			while($row = mysqli_fetch_array($r)){
				array_push($result,array(
					// fetching the data retrieved from the database to build $result
				));
			}
	mysqli_close($con);
	return $result;
}

function writeOnDatabase(args[]) {
	// PHP pseudocode to extract from MySQL database
	$con = mysqli_connect("host", "authentication_arguments");
		$sql_query = // based on args[] fetching
		$result = $conn->query($sql_query);
	mysqli_close($con);
	return $result; // either true or false
}
```

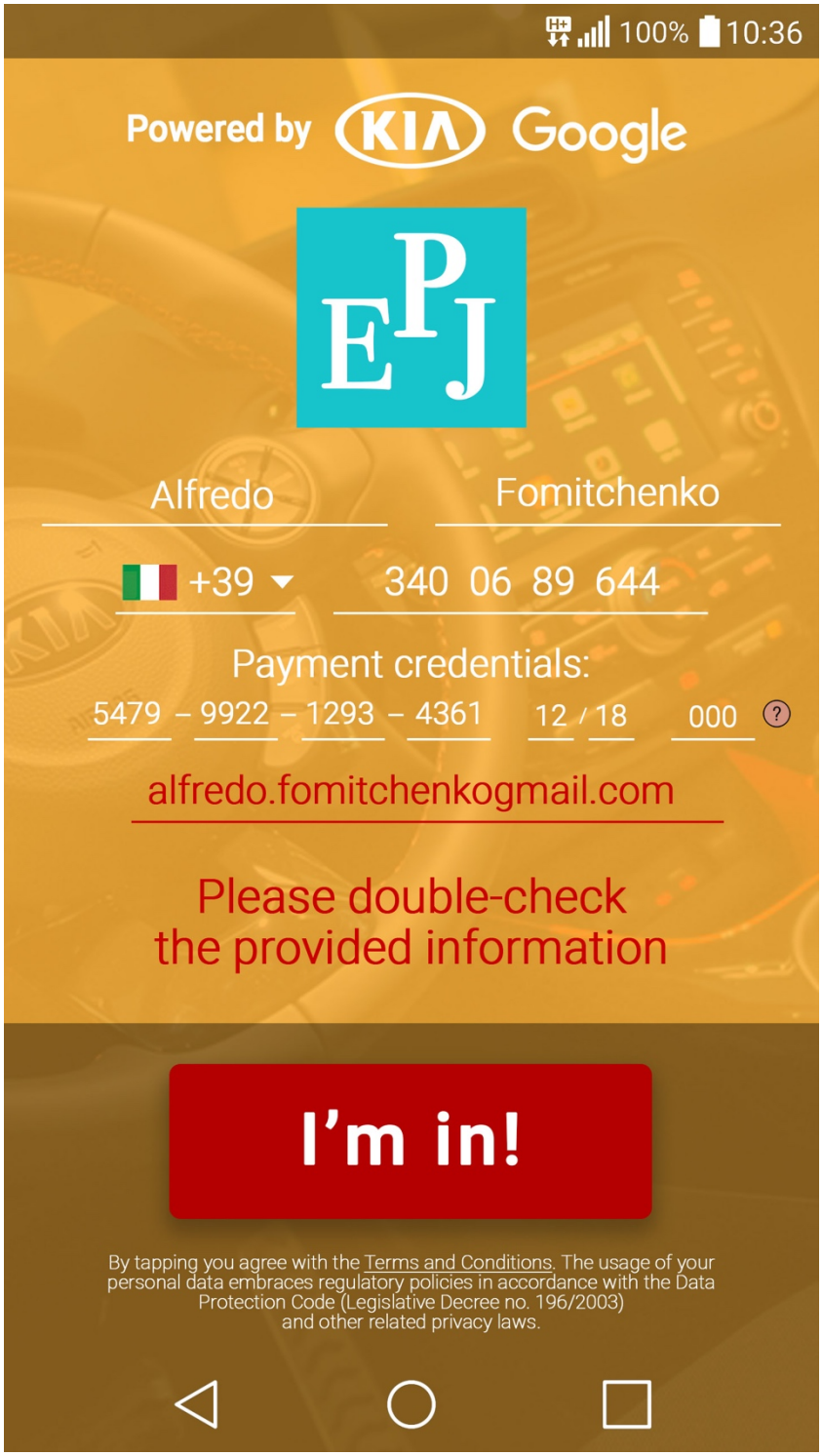
4. User Interface Design

The main frontend for the customers' requests for the system is the mobile application installed on their Android smartphones.

When connected to the USB car connectors, the application extends and changes its interface onto the car screen to show the charge timer.

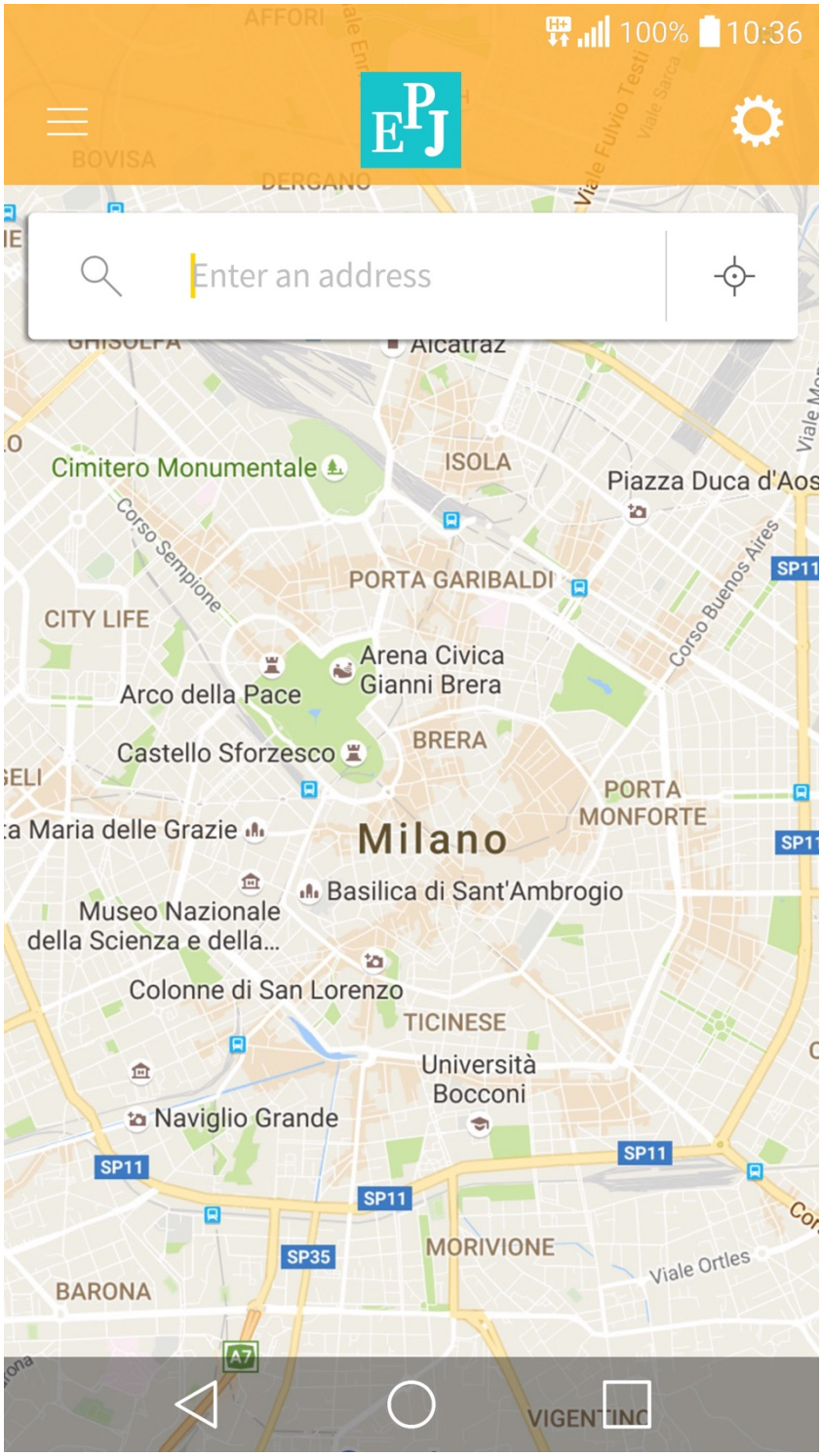

100% 10:36
Powered by
KIA
Google
EPJ
Login
New?

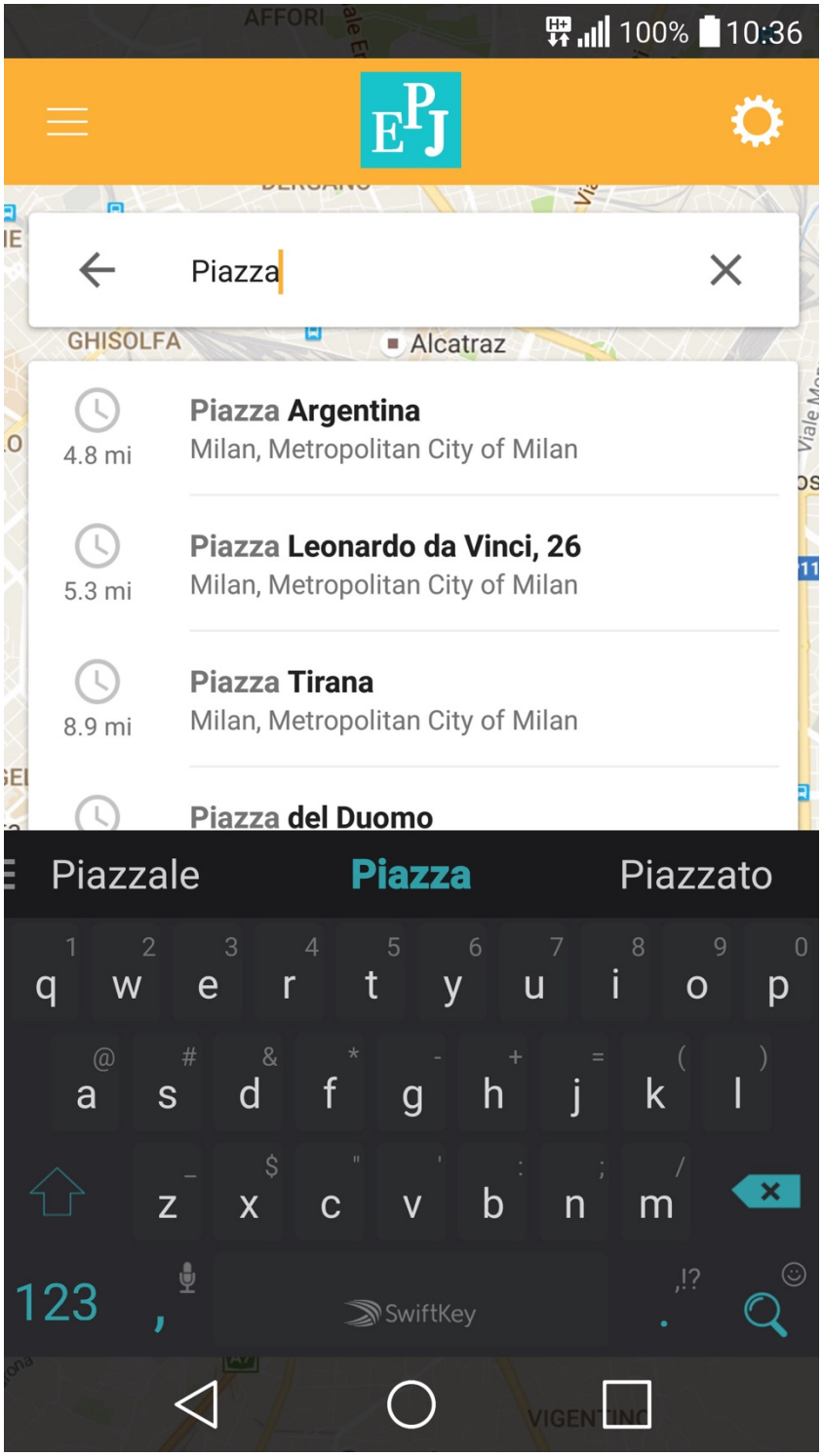

100% 10:36
Powered by KIA Google
EPJ
First name
Last name
+39
340 06 89 644
Payment credentials:
XXXX – XXXX – XXXX – XXXX
MM / AA
CVV2
name@example.com
No need to make a password up,
we will send it to you in a sec!
I'm in!
By tapping you agree with the Terms and Conditions. The usage of your personal data embraces regulatory policies in accordance with the Data Protection Code (Legislative Decree no. 196/2003) and other related privacy laws.

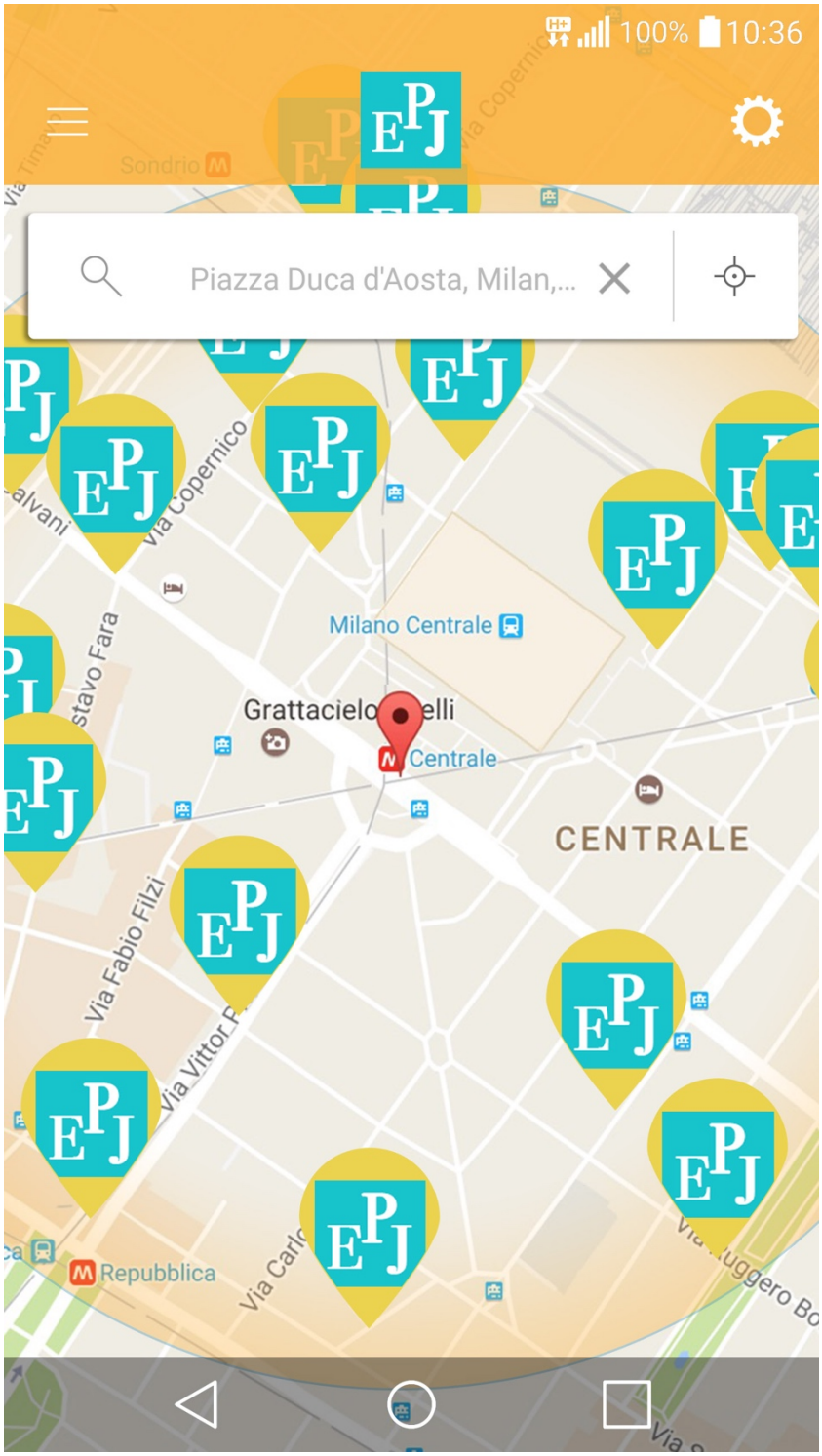

100% 10:36
Powered by KIA Google
EPJ
Alfredo
Fomitchenko
+39
340 06 89 644
Payment credentials:
5479 – 9922 – 1293 – 4361
12 / 18
000
alfredo.fomitchenkogmail.com
Please double-check
the provided information
I'm in!
By tapping you agree with the Terms and Conditions. The usage of your personal data embraces regulatory policies in accordance with the Data Protection Code (Legislative Decree no. 196/2003) and other related privacy laws.

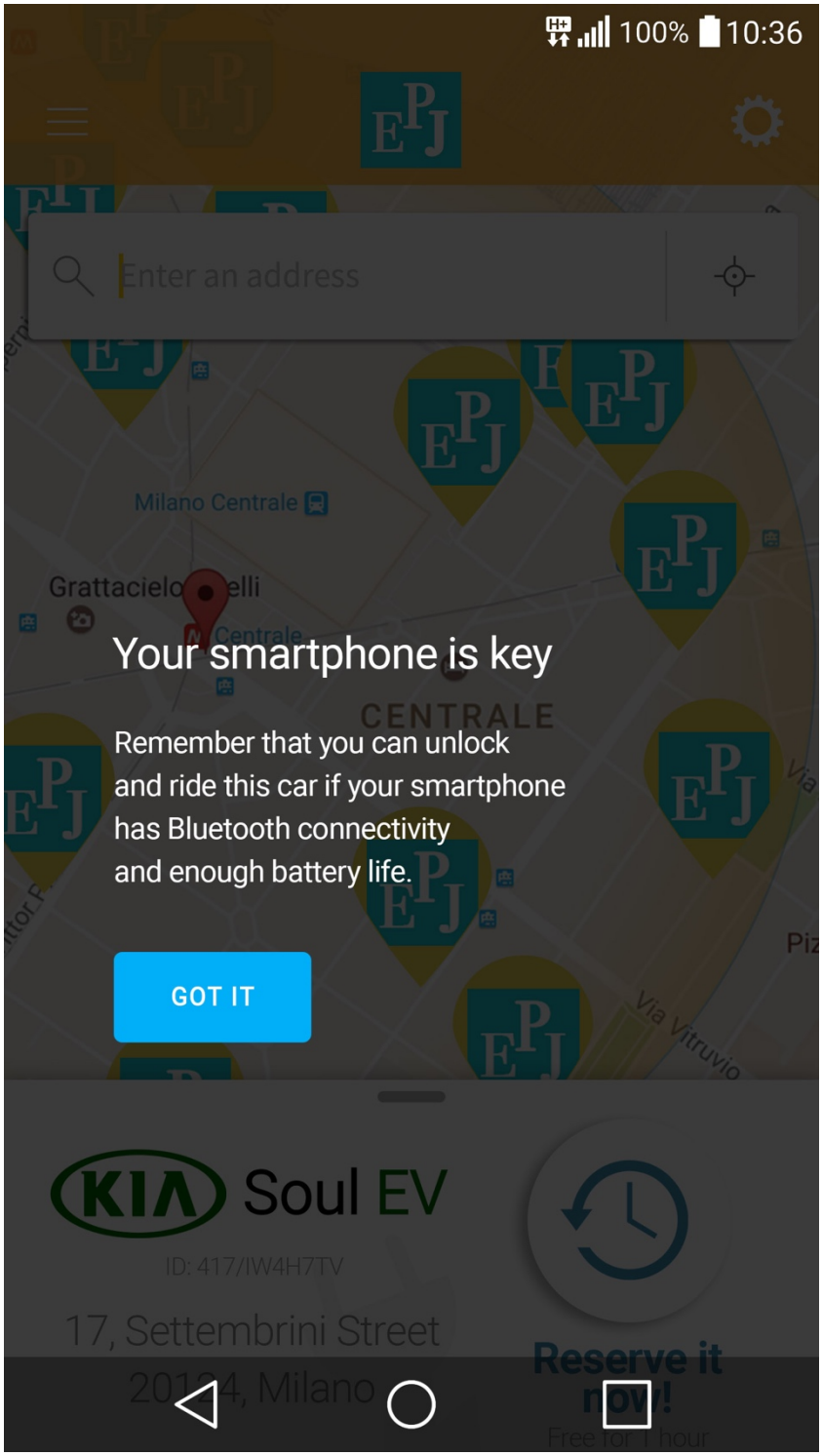

100% 10:36
Powered by
KIA
Google
EPJ
your email address
your password
Let's go!

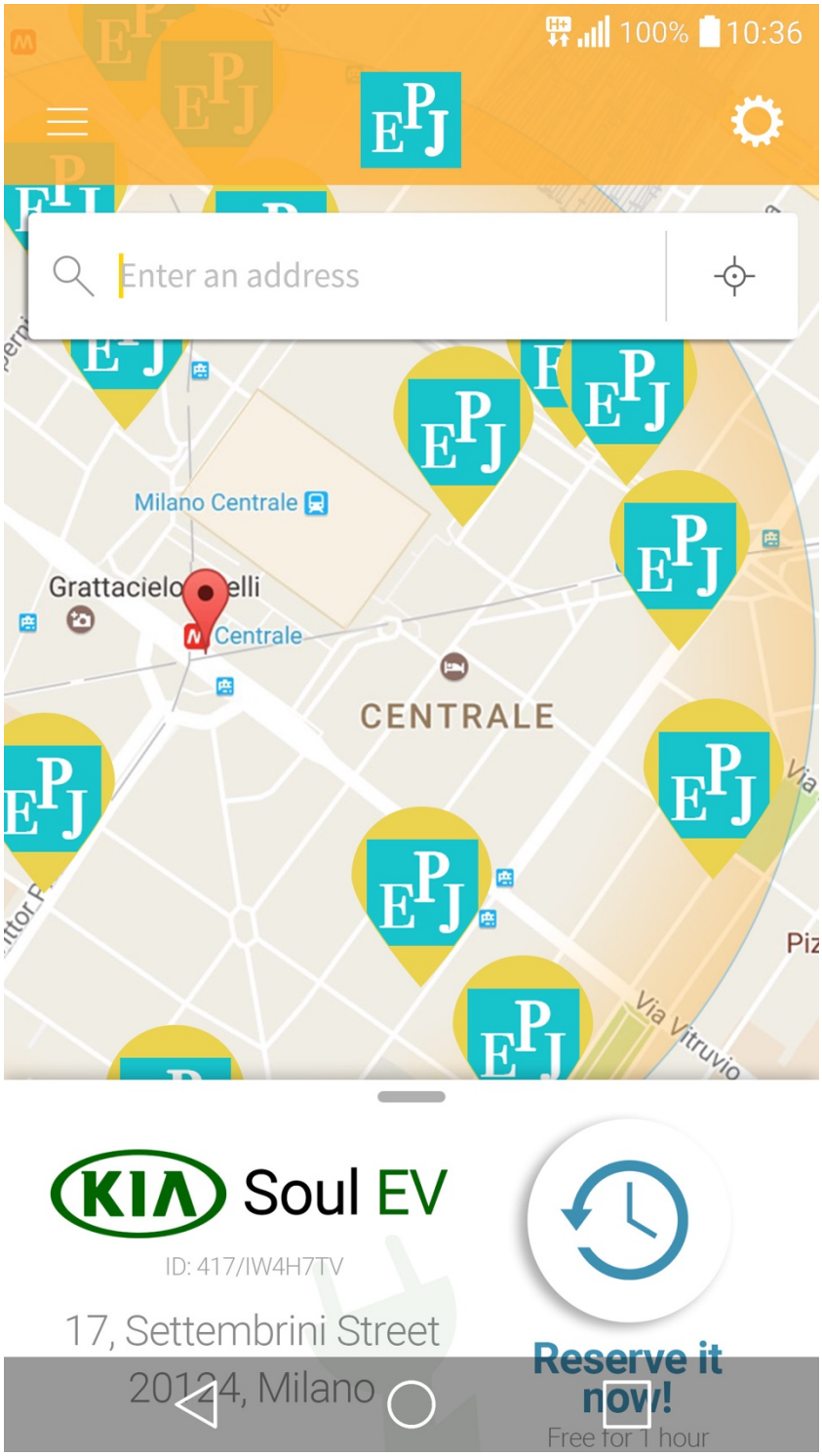

100% 10:36
EPJ
WELCOME
Tap on the KIA
and start saving the planet
We care about the environment. A new car sharing service like us could not have employed old gasoline cars. We only use electric vehicles. Supporting us you are supporting the bold move of saving the planet, so thank you.

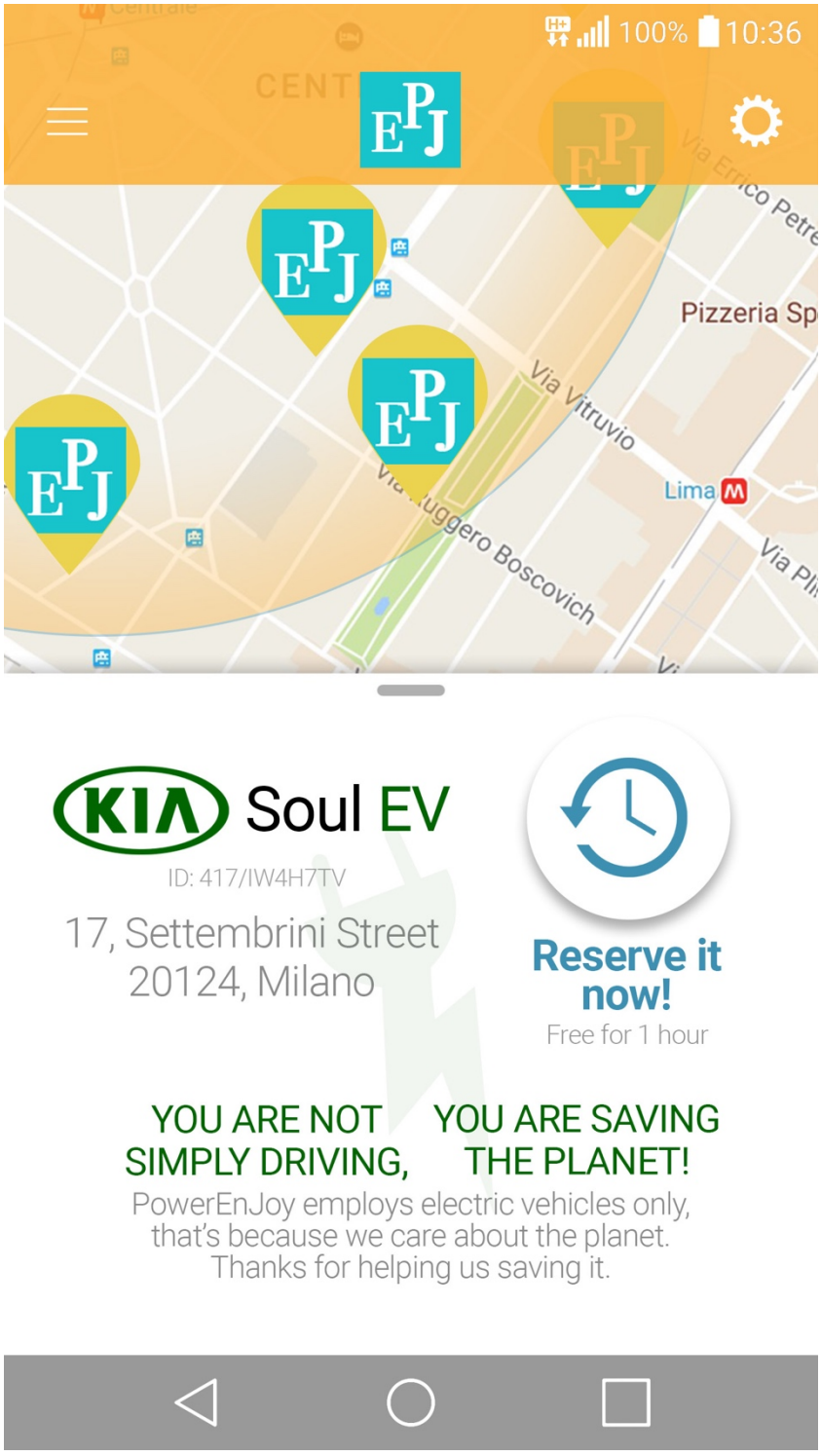

100%
10:36
EPJ
Enter an address
Cimitero Monumentale
ISOLA
Piazza Duca d'Aos
Corso Sempione
PORTA GARIBALDI
CITY LIFE
Corso Buenos Aires
SP11
Arena Civica
Gianni Brera
Arco della Pace
Castello Sforzesco
BRERA
PORTA
MONFORTE
Milano
Basilica di Sant'Ambrogio
Museo Nazionale
della Scienza e della...
Colonne di San Lorenzo
TICINESE
Università
Bocconi
Naviglio Grande
SP35
MORIVIONE
Viale Ortles
BARONA
A7

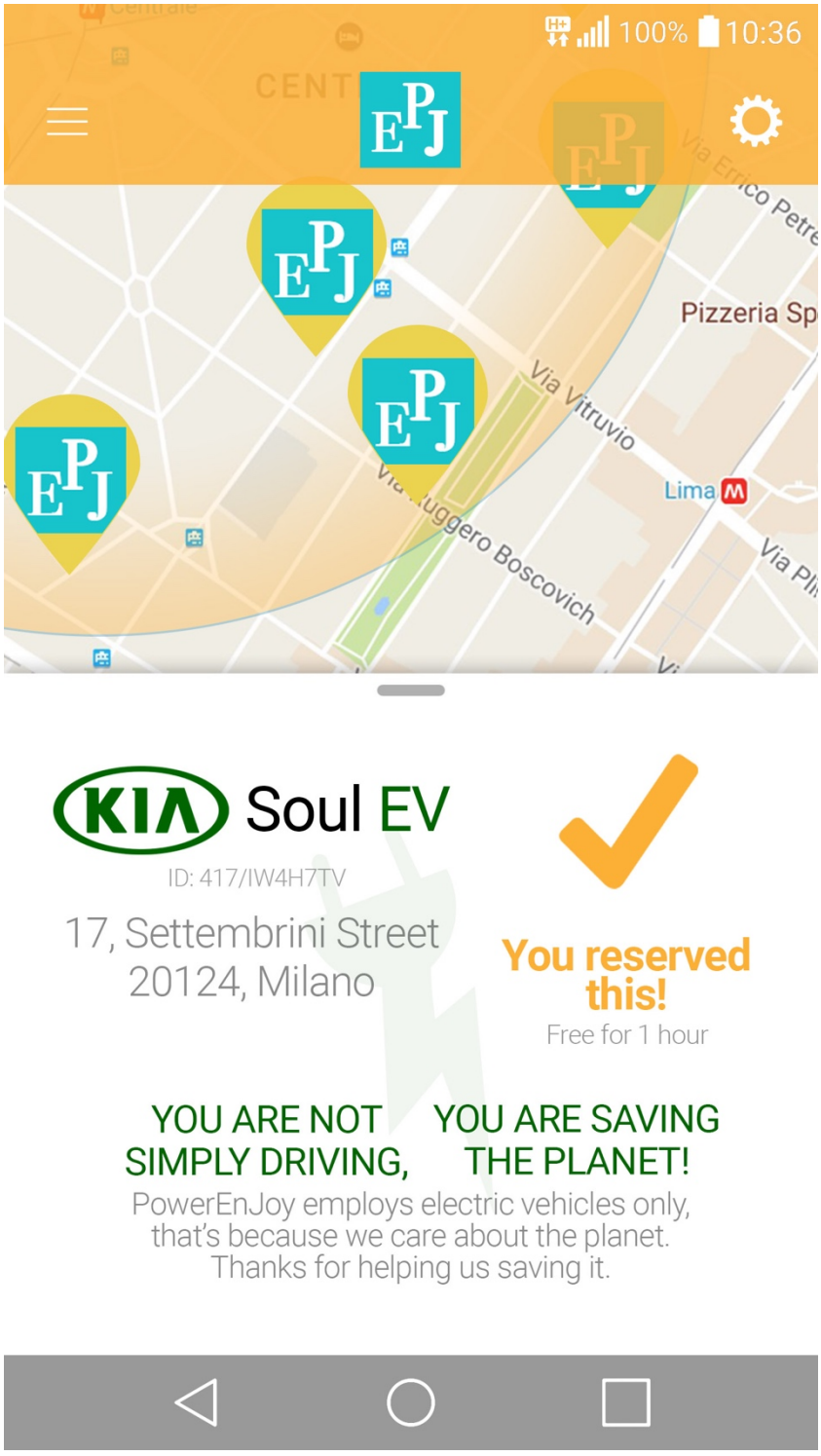

AFFORI
100%
10:36
EPJ
Piazza
GHISOLFA
Alcatraz
4.8 mi
Piazza Argentina
Milan, Metropolitan City of Milan
5.3 mi
Piazza Leonardo da Vinci, 26
Milan, Metropolitan City of Milan
8.9 mi
Piazza Tirana
Milan, Metropolitan City of Milan
Piazza del Duomo
Piazzale
Piazza
Piazzato
SwiftKey
123

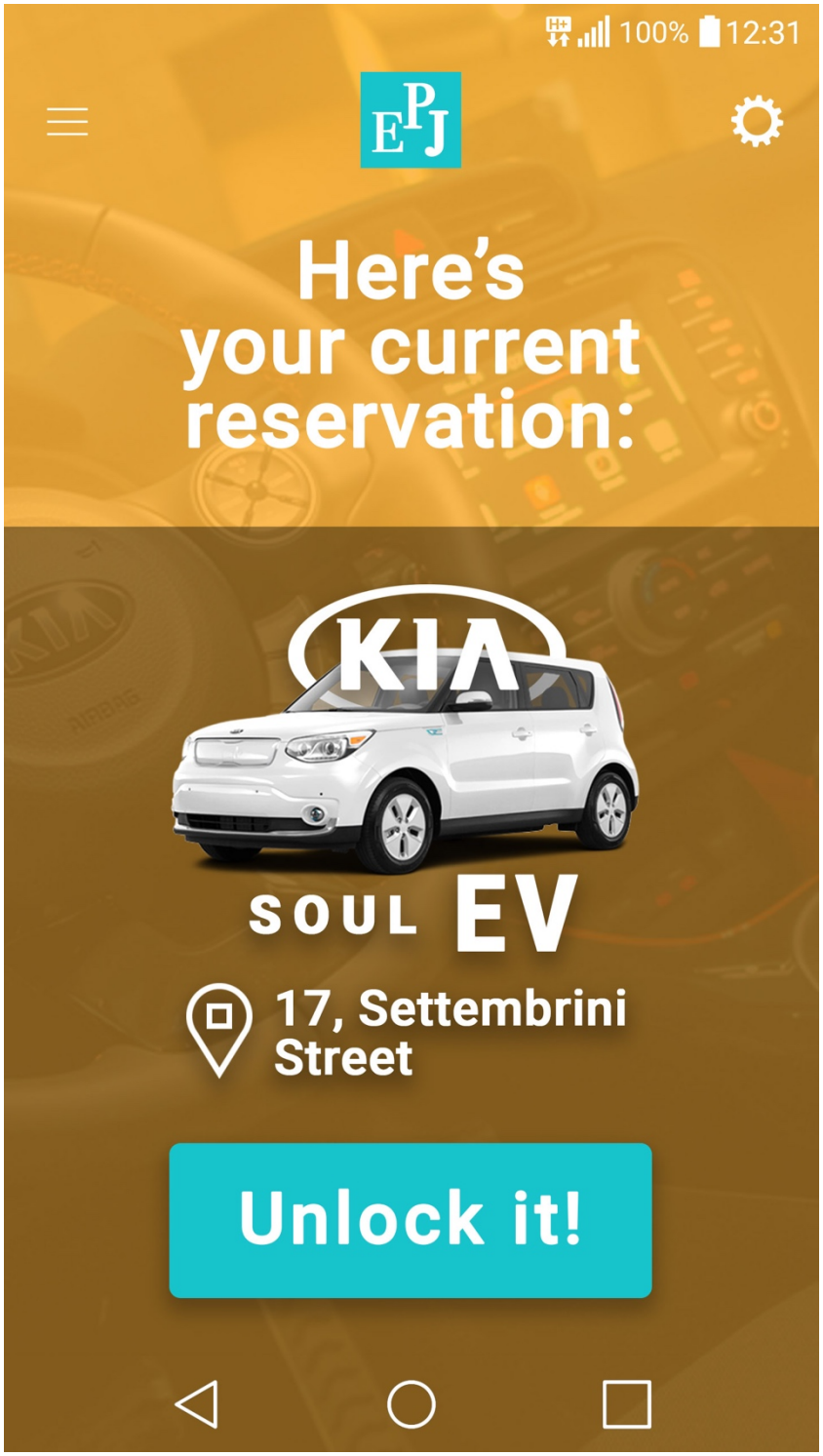

100%
10:36
Piazza Duca d'Aosta, Milan,...
Sondrio
Via Copernico
Milano Centrale
Centrale
CENTRALE
Via Fabio Filzi
Repubblica

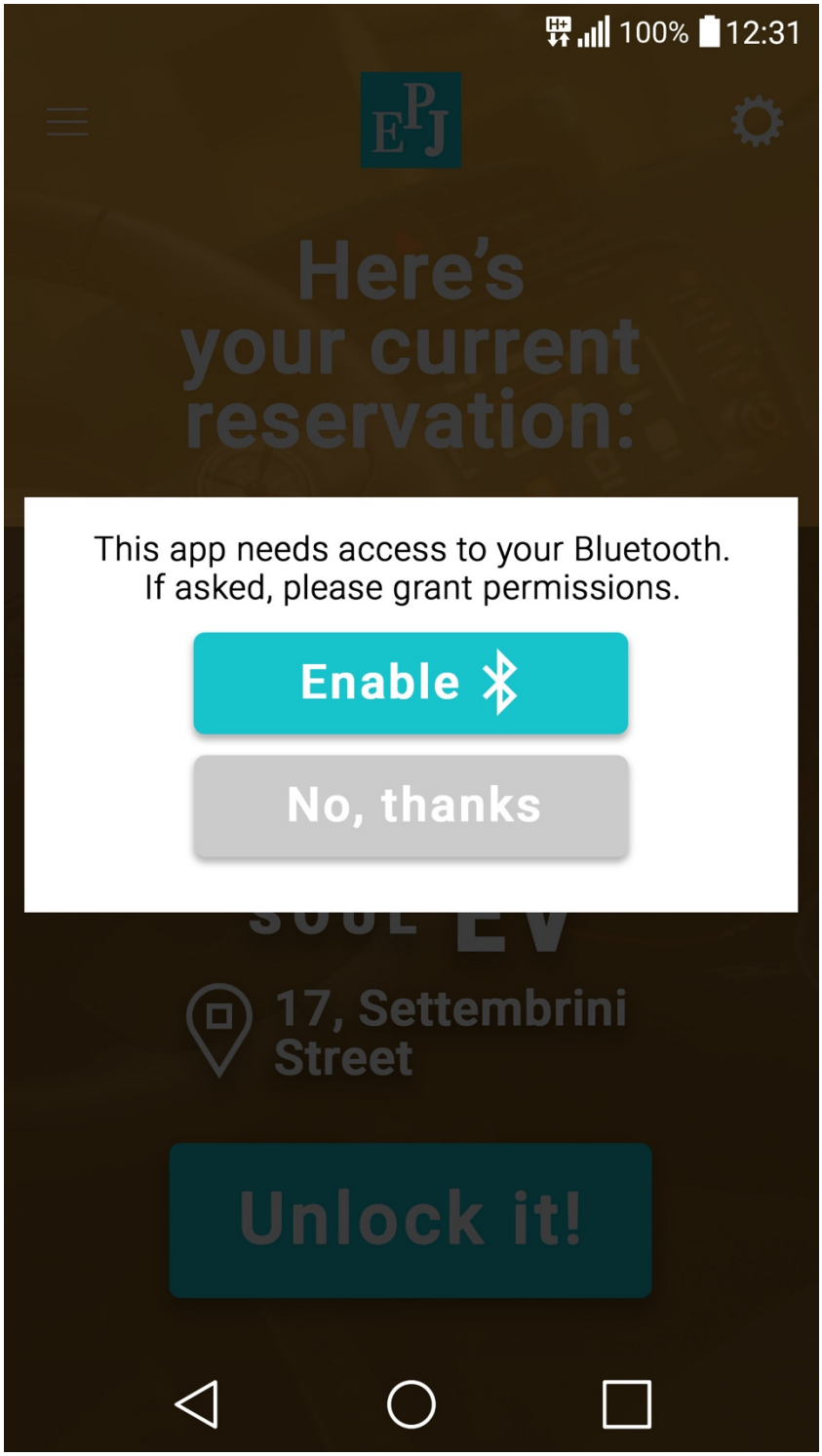

100% 10:36
Enter an address
Milano Centrale
Centrale
CENTRALE
Your smartphone is key
Remember that you can unlock and ride this car if your smartphone has Bluetooth connectivity and enough battery life.
GOT IT
Via Vitruvio
Soul EV
ID: 417/IW4H7TV
17, Settembrini Street
Reserve it now!

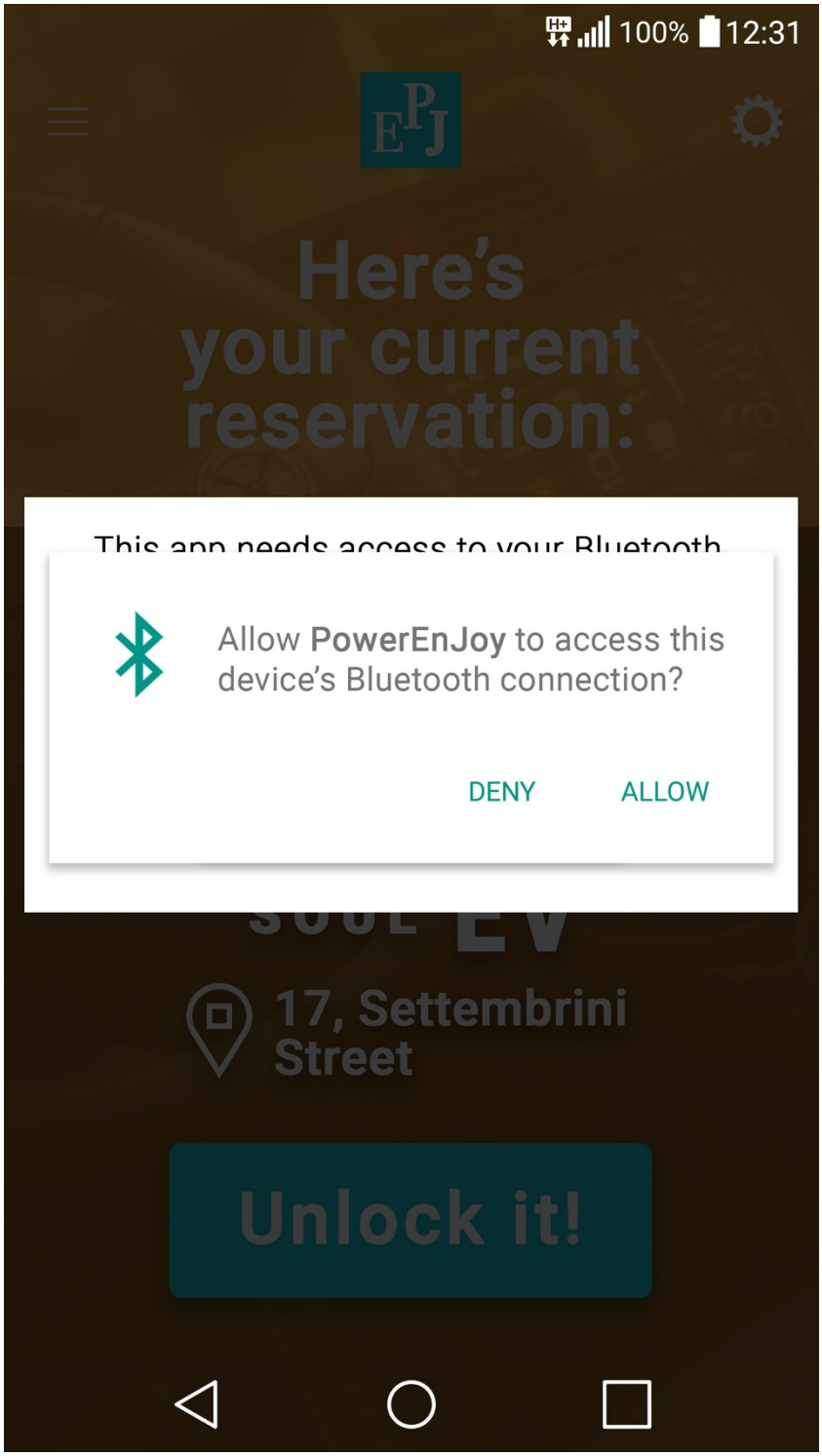

100% 10:36
Enter an address
Milano Centrale
Centrale
CENTRALE
Via Vitruvio
KIA Soul EV
ID: 417/IW4H7TV
17, Settembrini Street
20124, Milano
Reserve it now!
Free for 1 hour


100% 10:36
Pizzeria Sp
Via Vitruvio
Lima
Via Ruggero Boscovich
KIA Soul EV
ID: 417/IW4H7TV
17, Settembrini Street
20124, Milano
Reserve it now!
Free for 1 hour
YOU ARE NOT SIMPLY DRIVING,
YOU ARE SAVING THE PLANET!
PowerEnJoy employs electric vehicles only,
that's because we care about the planet.
Thanks for helping us saving it.

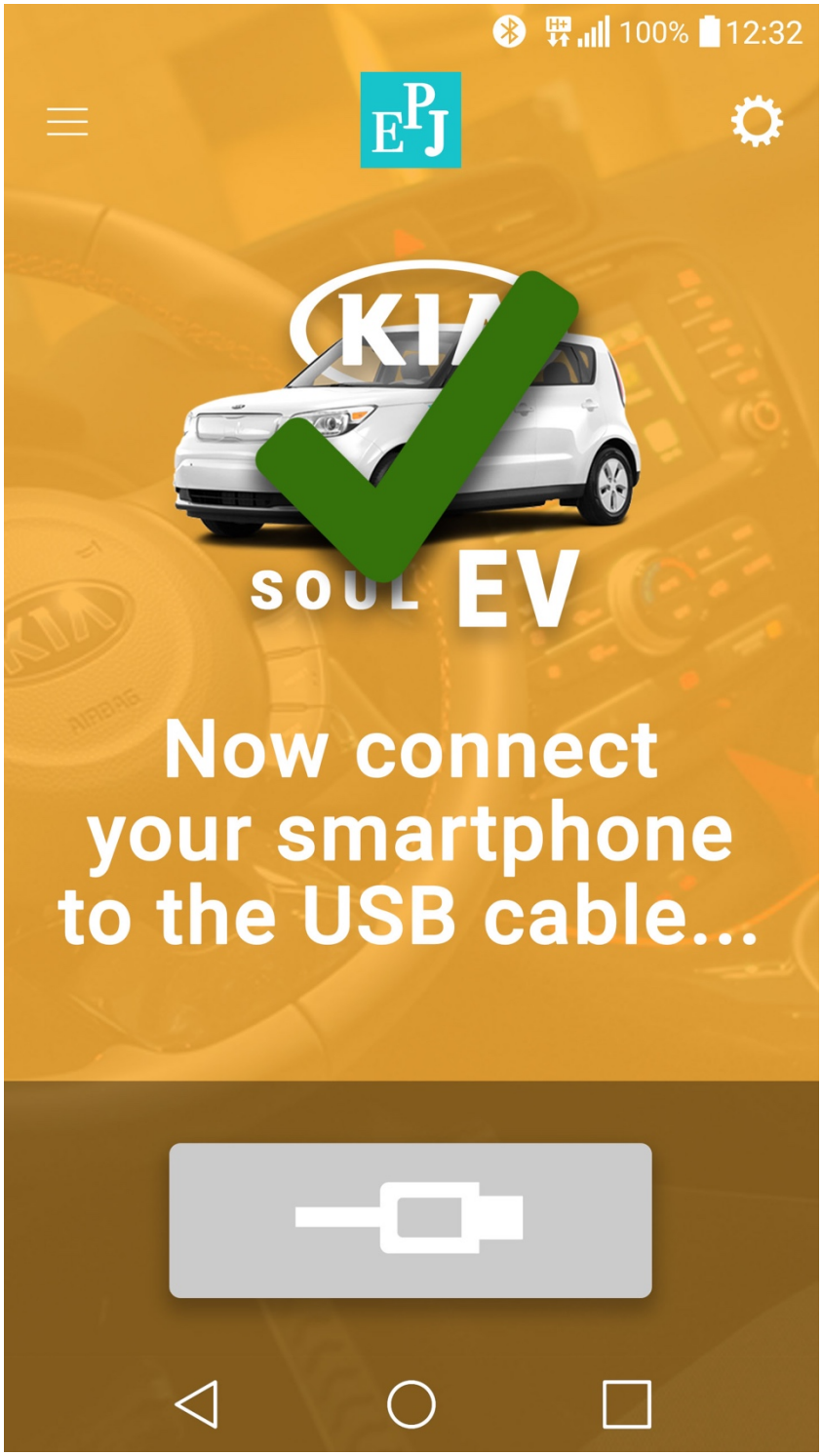

100% 10:36
Pizzeria Sp
Via Vitruvio
Lima
Via Ruggero Boscovich
KIA Soul EV
ID: 417/IW4H7TV
17, Settembrini Street
20124, Milano
You reserved this!
Free for 1 hour
YOU ARE NOT SIMPLY DRIVING,
YOU ARE SAVING THE PLANET!
PowerEnJoy employs electric vehicles only,
that's because we care about the planet.
Thanks for helping us saving it.

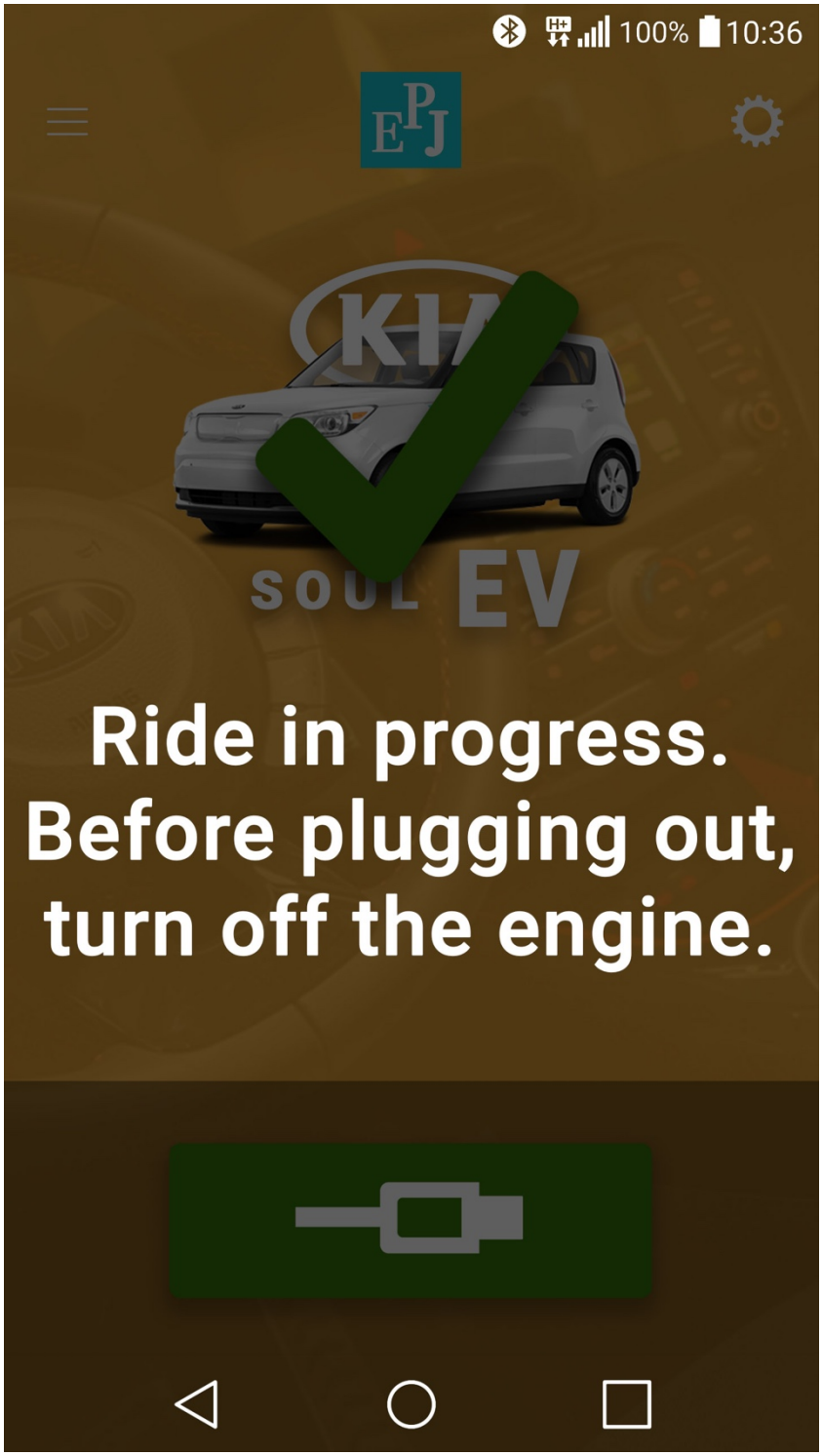

100% 12:31
EPJ
Here's your current reservation:
KIA
SOUL EV
17, Settembrini Street
Unlock it!


100% 12:31
Here's your current reservation:
This app needs access to your Bluetooth.
If asked, please grant permissions.
Enable
No, thanks
17, Settembrini Street
Unlock it!


100% 12:31
Here's your current reservation:
This app needs access to your Bluetooth
Allow PowerEnJoy to access this device's Bluetooth connection?
DENY
ALLOW
17, Settembrini Street
Unlock it!


100% 12:32
EPJ
You are unlocking
KIA
SOUL EV
17, Settembrini Street
Please wait while we connect to the car...
Please wait


100% 12:32
EPJ
KIA
SOUL EV
Now connect
your smartphone
to the USB cable...


100% 10:36
EPJ
SOUL EV
Ride in progress.
Before plugging out,
turn off the engine.


infinity
MAP
Radio
Bluetooth
RADIO
MEDIA
UVO
SEEK
TRACK
POWER PUSH
VOL
MAP VOICE
NAVI
PHONE
INFO
SETUP
TUNE
FILE
12:32
WELCOME TO
PowerEnJoy
To start your ride,
please ignite the engine.
Clean Air
FRONT
REAR
AUTO
OFF
A/C
MODE
CLIMATE
active ECO
PASSENGER AIR BAG
OFF


infinity
MAP
Radio
Bluetooth
RADIO
MEDIA
UVO
SEEK
TRACK
POWER PUSH
VOL
MAP VOICE
NAVI
PHONE
INFO
SETUP
TUNE
FILE
12:32
I parked.
Can I plug my
smartphone out?
Not yet...
5.42
€
EPJ
Clean Air
FRONT
REAR
AUTO
OFF
A/C
MODE
CLIMATE
active ECO
PASSENGER AIR BAG
OFF


infinity
SiriusXM
MAP
HD Radio
Bluetooth
RADIO
MEDIA
UVO
SEEK
TRACK
POWER PUSH
VOL
MAP VOICE
NAVI
PHONE
INFO
SETUP
TUNE
FILE
13:22
I parked.
Can I plug my
smartphone out?
Yes!
9.58
€
EPJ
Clean Air
FRONT
REAR
AUTO
OFF
A/C
MODE
CLIMATE
active ECO
PASSENGER AIR BAG
OFF

5. Requirements Traceability

The system components design has been driven toward the RASD requirements fulfillment.

Below is the list of the main components and interfaces in which each one of them is associated to the fulfilled requirements (for the sake of legibility AuthenticationManager, Router, ApplicationService, PDOInterface have not been included since they are involved in basically every single communication request).

5.1 RegistrationManager

[RE1] The system shall let a new customer register.

[RE2] The system shall generate, store and send to the new registered customer his associated password.

5.2 EmailInterface

[RE2] The system shall generate, store and send to the new registered customer his associated password.

5.3 LoginManager

[RE3] The system shall let a customer log in if he correctly provides the email address and the associated password.

5.4 ListAvailableCarsHandler

[RE4] The system shall request the appropriate application permissions to obtain and manage the customers' positions.

[RE5] The system shall locate a customer by the GPS position.

[RE7] The system shall retrieve and provide to a customer the list of available car from the customer's position within a fixed distance range.

5.5 LocationHelper

[RE5] The system shall locate a customer by the GPS position.

[RE6] The system shall convert a specified address provided by a customer into a specific location through Google Maps service (see [A8]).

[RE19] The system shall locate a car via its GPS position.

5.6 MapsInterface

[RE5] The system shall locate a customer by the GPS position.

[RE6] The system shall convert a specified address provided by a customer into a specific location through Google Maps service (see [A8]).

5.7 ReservationManager

[RE8] The system shall verify if a customer neither has previously reserved nor is currently using a car.

[RE9] The system shall process a customer's request of reservation associating the car to the customer, tagging it as unavailable and starting an associated reservation timer.

[RE15] When the customer unlocks the car (see [RE14]) the system shall stop the reservation timer (see [RE9]).

5.8 DataController

[RE9] The system shall process a customer's request of reservation associating the car to the customer, tagging it as unavailable and starting an associated reservation timer.

[RE12] After stopping the reservation timer as in [RE10], the system shall tag the car associated to the reservation as available.

5.9 BluetoothInterface

[RE13] The system shall request the appropriate application permissions to access Bluetooth features.

5.10 RideManager

[RE16] The system shall start a ride charge timer as soon as the car operating system communicates to the system via the application that the engine has been ignited by the customer.

[RE21] The system shall stop the ride charge timer when the car has been locked by the system (see [RE16], [RE20]).

[RE23] The system shall detect the car seat sensors states through the application at the beginning of the ride and determine the number of passengers of the ride (see [A21], [A22]).

[RE25] The system shall detect at the end of the ride the car battery charge through the application (see [A18]).

5.11 LockManager

[RE14] The system shall communicate with the car operating system via the Bluetooth interface (see [RE13]) to perform a car unlock when the customer's smartphone is close enough and the customer communicates to do so.

[RE20] The system shall perform an automated lock after the car engine has been turned off inside a safe area (see [A18], [A19], [A20]).

[RE21] The system shall stop the ride charge timer when the car has been locked by the system (see [RE16], [RE20]).

5.12 TimerManager

[RE9] The system shall process a customer's request of reservation associating the car to the customer, tagging it as unavailable and starting an associated reservation timer.

[RE10] The system shall stop the reservation timer associated to the reservation (see [RE9]) after one hour if the system has not stopped it yet (see [RE13]).

[RE15] When the customer unlocks the car (see [RE14]) the system shall stop the reservation timer (see [RE9]).

[RE16] The system shall start a ride charge timer as soon as the car operating system communicates to the system via the application that the engine has been ignited by the customer.

5.13 USBInterface

[RE18] The system shall display the current ride charge through the application onto the car screen.

[RE20] The system shall perform an automated lock after the car engine has been turned off inside a safe area (see [A18], [A19], [A20]).

5.14 ChargeManager

[RE11] After stopping the reservation timer as in [RE10], the system shall charge the customer 1 EUR.

[RE17] The system shall compute the current ride charge by means of the ride charge timer (see [RE16]) and the given charge rate per minute (see [A11]).

[RE22] After stopping the ride charge timer as in [RE21], the system shall request the charge as a financial transaction to the customer's credit card bank after verifying any applicable discount (see [G8], [G9]).

5.15 DiscountHelper

[RE11] After stopping the reservation timer as in [RE10], the system shall charge the customer 1 EUR.

[RE22] After stopping the ride charge timer as in [RE21], the system shall request the charge as a financial transaction to the customer's credit card bank after verifying any applicable discount (see [G8], [G9]).

[RE24] The system shall apply the mentioned discount to the ride charge before the charge has been requested to the credit card bank (see [RE22]).

[RE26] The system shall apply the mentioned discount to the ride charge before the charge has been requested to the credit card bank (see [RE22]).

6. Effort

-	14 November 2016:	**2 h**
-	16 November 2016:	**3 h**
-	17 November 2016:	**2 h**
-	19 November 2016:	**2,9 h**
-	20 November 2016:	**1,5 h**
-	21 November 2016:	**2 h**
-	23 November 2016:	**1 h**
-	25 November 2016:	**2,4 h**
-	26 November 2016:	**2 h**
-	27 November 2016:	**2 h**
-	29 November 2016:	**1,6 h**
-	30 November 2016:	**4,2 h**
-	1 December 2016:	**2,5 h**
-	2 December 2016:	**6 h**
-	3 December 2016:	**6 h**
-	4 December 2016:	**4,4 h**
-	5 December 2016:	**4,5 h**
-	6 December 2016:	**1 h**
-	7 December 2016:	**5 h**
-	8 December 2016:	**6,6 h**
-	9 December 2016:	**7,9 h**
-	10 December 2016:	**4,4 h**
-	11 December 2016:	**2 h**
-	20 December 2016:	**0,3 h**
-	23 December 2016:	**1,1 h**
		78,3 h

7. Changelog

- v1.0
- v1.1: added more interfaces between components inside the Component view to explicitly reference them in the ITPD. Removed DataService interface usage by LockManager and TimerManager. Removed model representation since the logic entirely and independently resides within DataController.
  Major changes to Component interfaces diagram.