POLITECNICO
MILANO 1863

Software Engineering 2: "PowerEnJoy"

Design Document

Version 1.0

Piccirillo Luca - 790380
Zampogna Gian Luca - 863097
Zini Edoardo - 875275

December 11th, 2016

Contents

1 Introduction

1.1 Purpose

The goal of this document is to provide an overall description of PowerEnJoy software architectural design together with a brief overview of service peculiar algorithms. Several standard description diagrams will help the reader to better understand taken design decisions. This document is intended for an audience of software developers and project managers.

1.2 Scope

PowerEnJoy is an application for a new electric car-sharing service.
User must be able to register to the service, with user-friendly mobile application, find a car nearby and book it. Once arrived near the car user must be able to unlock, enter and drive towards his destination.
Inside the car, the user can interact with the car onboard ride assistant for getting details about his ride or driving directions.
Once the ride is ended the car will be locked by the system and user is billed.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Acronyms, Abbreviations

- PEJ = PowerEnJoy
- SBL = Service Business Logic
- SPA = Special Parking Area

1.3.2 Definitions

- **Car:** every vehicle, which respects the requirements, that the system allows the users to use.
- **PowerUser:** user who is already registered to the service and is currently logged in.
- **Safe Parking Area:** pre-defined areas (i.e. streets) where a user is allowed to park.
- **Service Business Logic:** software logic of the service the user do not directly interact with.
- **Special Parking Area:** pre-defined areas (i.e. streets) where a user is allowed to park and where the batteries of the Cars can be plugged into the power grid.
- **Total Base Fare:** amount of money that the user pays for the ride duration only. It does not include any discount or penalty for any other specific condition.

- **Total Ride Fare:** amount of money that the user pays. It includes any discount or penalty for any other specific condition satisfied during the ride.

1.4 Internal Reference Documents

- Project's Assignment document: AA 2016-2017 Software Engineering 2 - Project goal, schedule, and rules.
- Software Engineering 2: "PowerEnJoy" Requirements Analysis and Specification Document.
- See section 7.1 for external references.

1.5 Document Structure

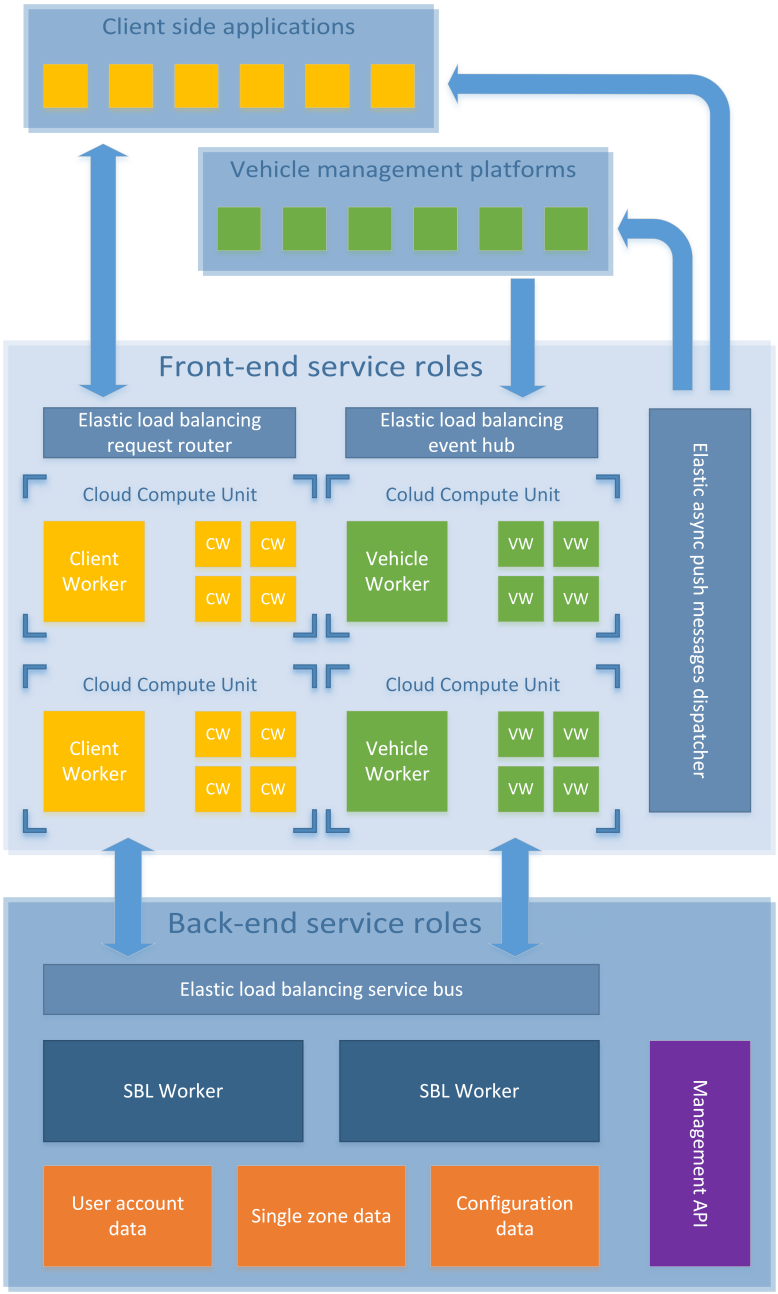
This document is split in 6 parts:

- **Architectural Design:** contains the different architectural views of the system.
- **Algorithm Design:** contains the pseudo-codes of the most relevant algorithmic parts of the system.
- **User Interface Design:** contains an overview of user interface and user experience.
- **Requirements Traceability:** shows the correlation among the requirements defined in the RASD and the design elements contained in this document.
- **Effort Spent:** contains the amount of time each member spent on the document, split between time spent working together and time spent on his own.
- **References:** contains all the external references required by this document.

2 Architectural Design

2.1 High level components and interactions

The following block diagram describes the overall solution design by giving an high-level perspective of all main application roles. In particular, it makes easy to understand the partition of software service components across the solution realm and their interaction mechanisms through internal subsystems interfaces.



- **Client side applications**
  Represent the pool of user interactive applications made of both mobile applications running on user owned devices and onboard ride assistants running on the onboard infotainment devices of PowerEnJoy Cars.

- **Vehicle management platforms**
  Represent the pool of embedded controllers devoted to remotely interfacing the vehicle hardware.

- **Front-end service roles**
  - **Client Worker**
    A single Client Worker node acts as a proxy between any remote interactive client and back-end components. In other words, it exposes all interfaces needed by any application the end user directly interacts with to allow decoupling from core service feature components, avoiding any client technology-dependent issue to be addressed by back-end design decisions. It is able to validate an existing authentication token avoiding redundant messaging to the back-end. It also provides access to static/cached resources and it's designed to be fully stateless, allowing automatic/on-demand scale-up of nodes instances to distribute workload across, dramatically increasing reliability and performance. A Client Worker node can return execution results to remote clients both in synchronous (request/response) or asynchronous mode (event/notification).
  - **Vehicle Worker**
    A single Vehicle Worker node acts as a proxy between any remote supported vehicle platform and back-end components. In other words, it exposes all interfaces needed by any supported vehicle to allow decoupling from core service feature components, avoiding any platform-dependent issue to be addressed by back-end design decisions. It's designed to manage and keep track of an huge number of status/telemetry updates. Inside a Vehicle Worker node, updates from multiple different vehicles are stored until expiration and timestamped. When an event is triggered by a new vehicle the node registers itself to the back-end as the handler of that vehicle. This way we are still allowing automatic/on-demand scale-up of nodes instances to distribute workload across, dramatically increasing reliability and performance. A Vehicle Worker node works only in asynchronous mode (event/notification) w.r.t. the remote vehicle.
  - **Elastic load balancing request router**
    Available from any common cloud provider. Provides automatic scale-up/-down of nodes instances based on their resources utilization and redirects request messages to nodes guaranteeing a fair workload distribution across them.

  - **Elastic load balancing event hub**
    Available from any common cloud provider. Provides automatic scale-up/-down of nodes instances based on their resources utilization and manages to forward an huge number of event messages to nodes guaranteeing a fair workload distribution across them.
  - **Elastic async push messages dispatcher**
    Available from any common cloud provider. Provides efficient queuing of large number of messages to dispatch to multiple groups of targets. It also takes care of creating and keeping up network notification channels.

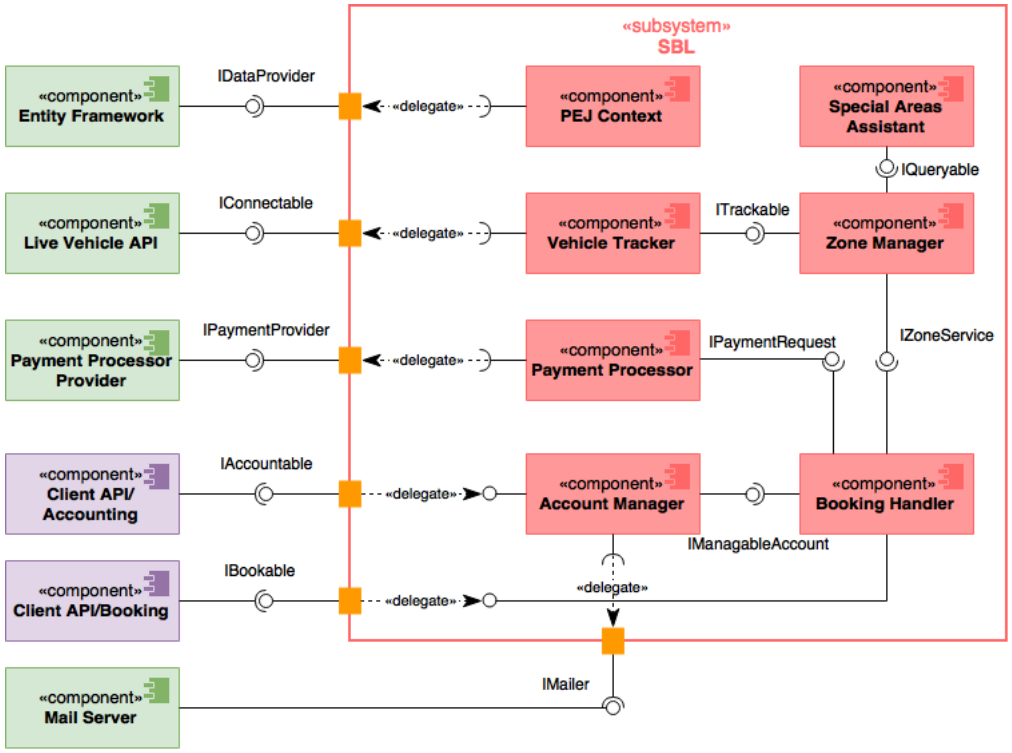
- **Back-end service roles**
  - **Service Business Logic Worker**
    Each SBL Worker node provides all PowerEnJoy core service features. They actually runs the whole service logic decoupled from any presentation layers and communication protocols to front-end devices. Each SBL Worker is fully stateless and supports simultaneous multiple worker's instances with obvious redundancy and performance advantages. For better performances a single node is restricted by design to serve only a specific service zone. An SBL Worker also provides some data access optimization (i.e. caching, mirroring) to inner components and can be instantiated both automatically by scaling managers or manually if needed. For example, if interface compatibility is preserved, it will be possible to manually instantiate an SBL Worker running a different software version, allowing seamless on-field testing, debugging and upgrade experience.
  - **Elastic load balancing service bus**
    Available from any major cloud provider. Allows efficient interconnection of components between back-end and front-end roles. It can be configured to act as an automatic scaling manager if the cloud provider offers that option.
  - **User Account Data**
    Stores all account related data such as user credentials, personal identity, driving license, credit card details and payments history. Accounts are unique and shared across all PowerEnJoy covered zones.
  - **Configuration Data**
    Stores all service global parameters that system administrators can modify according to their needing, like those related to discounts and penalties policies, service fares, booking timeouts and so on.
  - **Single Zone Data**
    Stores all data related to a single zone covered by the PowerEnJoy service such as vehicles, Safe and Special Parking Areas, active bookings and any parameter override that should only be applied to that zone.
  - **Management API**
    Provides administrative and monitoring low-level access to core components,

designed to be easily interfaced with any business management platform (ERP, CRM, enterprise portals, etc.). It's existence has been taken into account only to adapt current solution design for future software evolution. No further design or development effort will be spent on this component unless requirements renegotiation. As of this document revision, the described solution is fully functional and serviceable without this component by direct manipulation of data.

2.2 Component view

This section contains the component view and the descriptions of each component. Their interfaces are described in the section component interfaces (2.5).

2.2.1 Service Business Logic



The Service Business Logic subsystem is composed of:

- **PEJ Context**
  It is the component through which all the others can access data entities containing all the information and details about zone, vehicles, accounts, configurations, bookings, rides and payments.

- **Zone Manager**
  It is the component that connects Booking Handler to related Vehicle Tracker and Special Parking Areas of inherent zone applying any zone-specific configurations.

- **Special Areas Assistant**
  It is the component that provides the set of functions needed to get lists of Special Parking Areas satisfying specific requirements.

- **Vehicle Tracker**
  It is the component that asynchronously interacts with the Vehicle Management Platform collecting data required by other components.

- **Booking Handler**
  It is the component that manages the bookings and their related rides. It provides the functions for creating a new booking, having it progress to a ride and ending it creating a pending payment record of appropriate amount.

- **Payment Processor**
  It is the component in charge of managing pending payments issuing requests to Payment Processor Provider.

- **Account Manager**
  It is the component in charge of managing user data when he creates a new account, signs in, asks for his personal profile page or wants to modify personal details.

and it interacts with:

- **Entity Framework**
  It is the set of external sub-components that is in charge of binding data entity models to corresponding datasource schema.

- **Live Vehicle API**
  It is the component that connects to a vehicle management platforms in order to get live data including positions, battery level, engine ignited, and all the other data provided by the vehicle sensors.

- **Payment Processor Provider**
  It is the third party system PowerEnJoy relies on in order to bill the customer's credit cards.

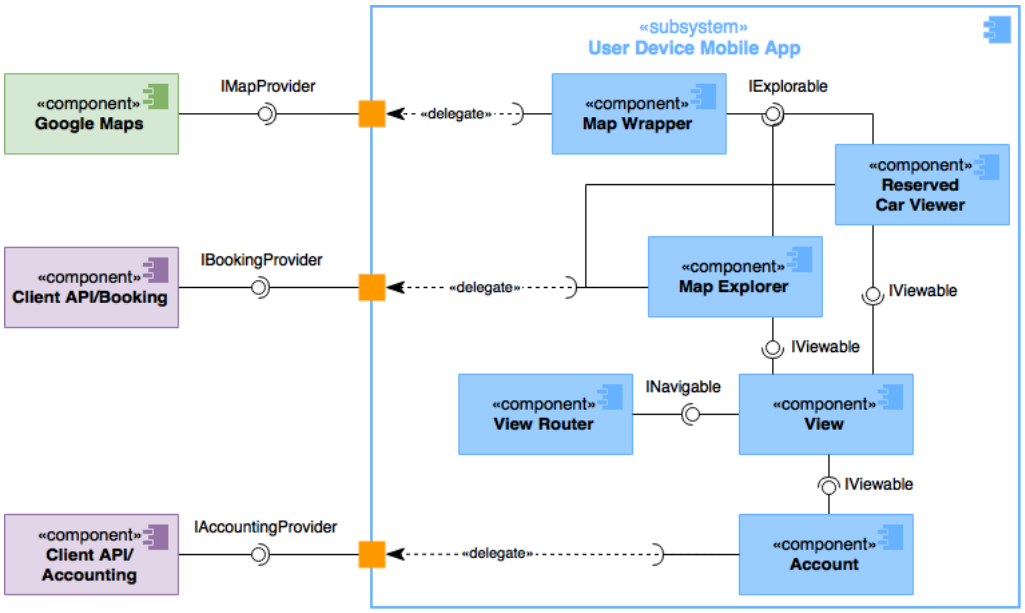
- **Client API/Accounting**
  It is the middleware between SBL and client-side apps the user interact with. It exposes only the set of interfaces required to register, sign in or modify the personal user data.

- **Client API/Booking**
  It is the middleware between SBL and client-side apps the user interact with. It exposes the set of interfaces required to create or cancel a reservation, and manage

the whole ride including access to the map of available vehicles, the position of the reserved Car or to get lists of Special Parking Areas.

- **Mail Server**
  It is the component in charge of sending to the specified user the mail containing payment result and details of the ride.

2.2.2 User Device Mobile App



The User Device Mobile App is composed of:

- **View**
  It is a set of sub-components representing something the user can see and interact with; each sub-component renders the view relying on the view models provided by other components and translates user inputs to well-formed and sanitized requests sent through other components. For this application this set includes sub-components related to map, sign-in and sign-up forms, account and reservation management.

- **View Router**
  It is the component that implements and keep tracks of navigation paths across various views, providing for a given view model the right view component to be navigated that is able to present it to the user. It also implements backward navigation feature.

- **Map Wrapper**
  It is the component that wrap the outsourced map images into an user control that

makes them scrollable and zoomable, with the ability to overlay custom objects in defined position. It allows the application not to be coupled to a specific third-party maps provider.

- **Map Explorer**
  It is the component that takes the map provided by the Map Wrapper and takes care of showing the position of all available Cars on that map. These positions are obtained via Client API/Booking call.

- **Reserved Car Viewer**
  It is the component that takes the map provided by the Map Wrapper and adds the position of the reserved Car. This position is stored within the app or it is requested via Client API/Booking call if necessary. It allows the user to request a booking cancellation or the unlock of its associated Car.

- **Account**
  It is the component in charge of all the account-related functions. After the sign in, it provides the view with the user avatar, so that it can be used as access to the user personal data page which is also managed by this component.

and it interacts with:

- **Google Maps**
  It is the third party map provider whose maps are used in the user device mobile app.

- **Client API/Accounting**
  See above.

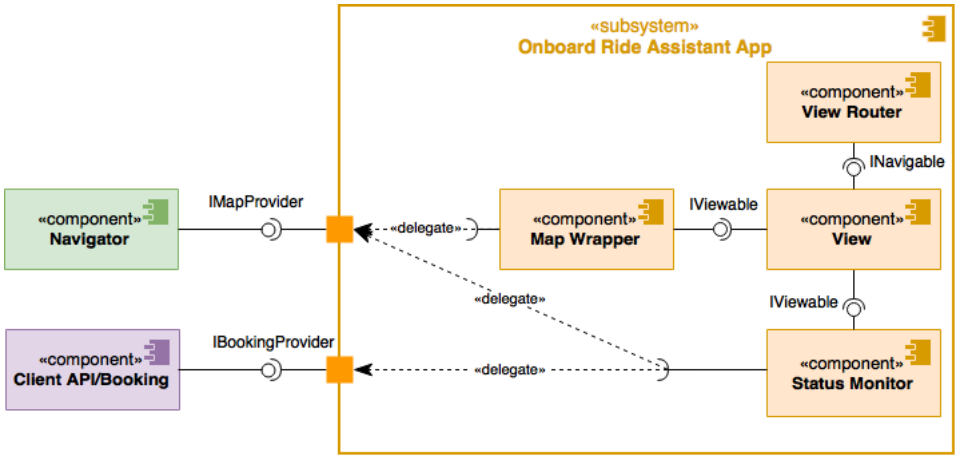
- **Client API/Booking**
  See above.

2.2.3 Onboard Ride Assistant App

The Onboard Ride Assistant App is composed of:

- **View**
  It is a set of sub-components representing something the user can see and interact with; each sub-component renders the view relying on the view models provided by other components and translates user inputs to well-formed and sanitized requests sent through other components. For this application this set includes sub-components related to navigator, search input field, options and ride details.

- **View Router**
  It is the component that implements and keep tracks of navigation paths across various views, providing for a given view model the right view component to be navigated that is able to present it to the user. It also implements backward navigation feature.



- **Map Wrapper**
  It is the component that wraps the maps provided by the Navigator and makes it scrollable and zoomable, with the ability to overlay custom objects in defined position. It allows the application not to be coupled to a specific third-party navigation product.

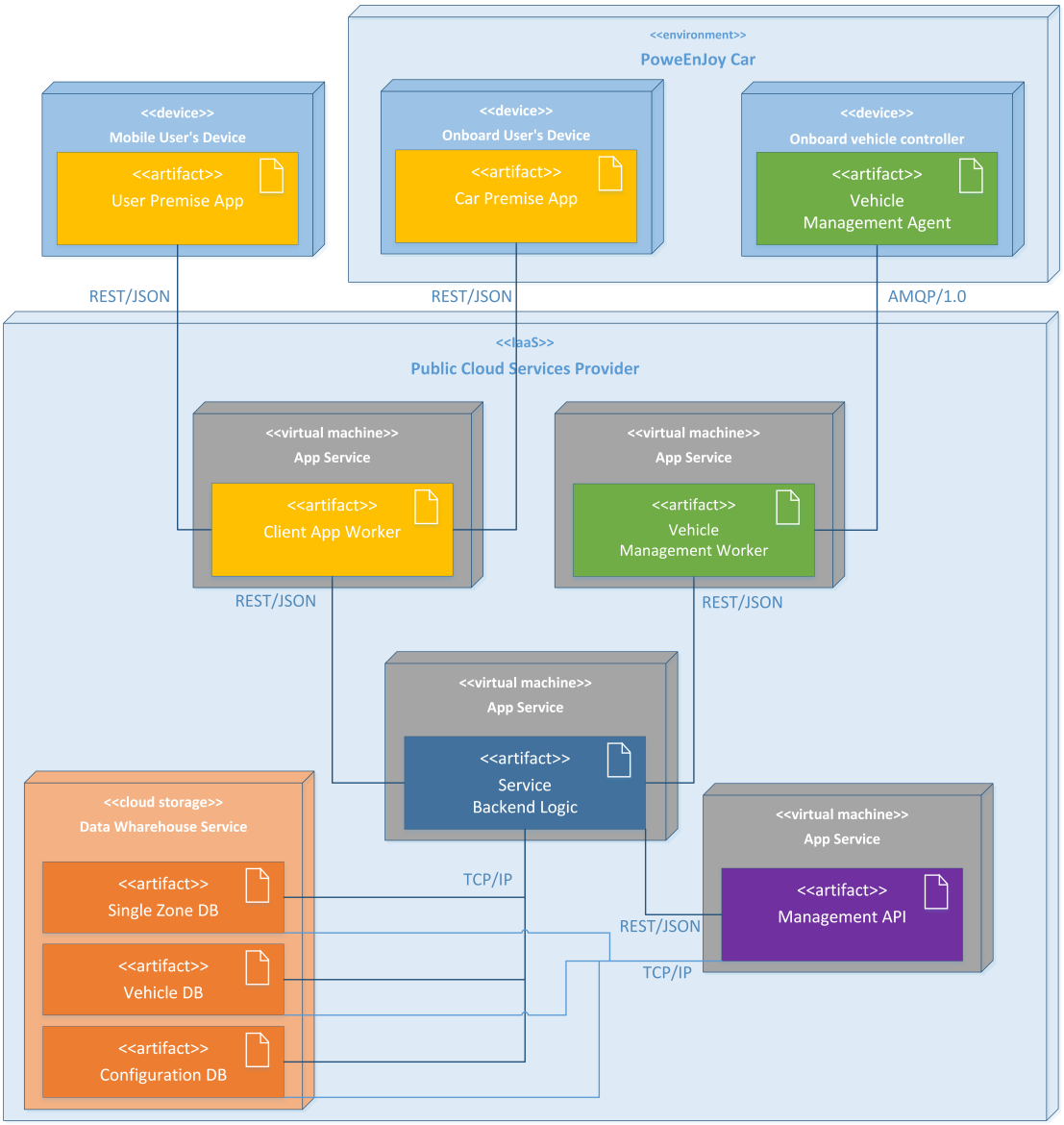
- **Status Monitor**
  It is the component that updates the ride details and the options available to the user: money saving alternative destination and Special Parking Areas listing.

and it interacts with:

- **Navigator**
  It is the third-party navigation software which provides navigation directions to the user, and translate the inserted addresses into precise coordinates. It also provides maps images to other components.

- **Client API/Booking**
  See above.

2.3 Deployment view



- **Mobile User's device**
  It's the mobile device owned by a generic user. He can download and install the PowerEnJoy application via platform specific app. Any major mobile systems are supported.

- **PowerEnJoy Car**
  - **Onboard User's Device**
    It's the device which every Car is equipped with, it executes a preloaded

interactive application, and includes the third-party navigator software along with offline map data. This device also shares GPS sensor data and Internet connectivity with any other onboard device.

  - **Onboard Vehicle Controller**
    It's an embedded device which every Car is equipped with, it's connected to Car hardware interface and sensor pool.

- **Public Cloud Service Provider**
  - **App Service**
    It's an IaaS virtual machine already configured to run web-based services. It can be replicated into new instances by scaling managers. The app service setup is stored in virtual storage and every app service instance has access to a virtual copy of that setup. The deployment is managed by continuous integration services or other specific tools by the cloud provider. Most software components of this solution runs on such kind of compute units.
  - **Data Wharehouse Service**
    It's a virtual relational data storage, running on a DBaaS environment. Every database for PowerEnJoy take place in this cluster but each database can also be configured to be phisically located in preferred country or region in case of regulatory constraints.

- **Communication protocols**
  For a comprehensive description of deployed communication protocols refer to section §2.6.3.

2.4 Runtime view

This section contains some Sequence Diagrams of the application.
In particular it shows what happens whenever:

- User send a request for unlock the Car;
- User set ride destination and drive towards it; the third diagram extends the second one and shows the Special Parking Area List and the Money Saving option features.

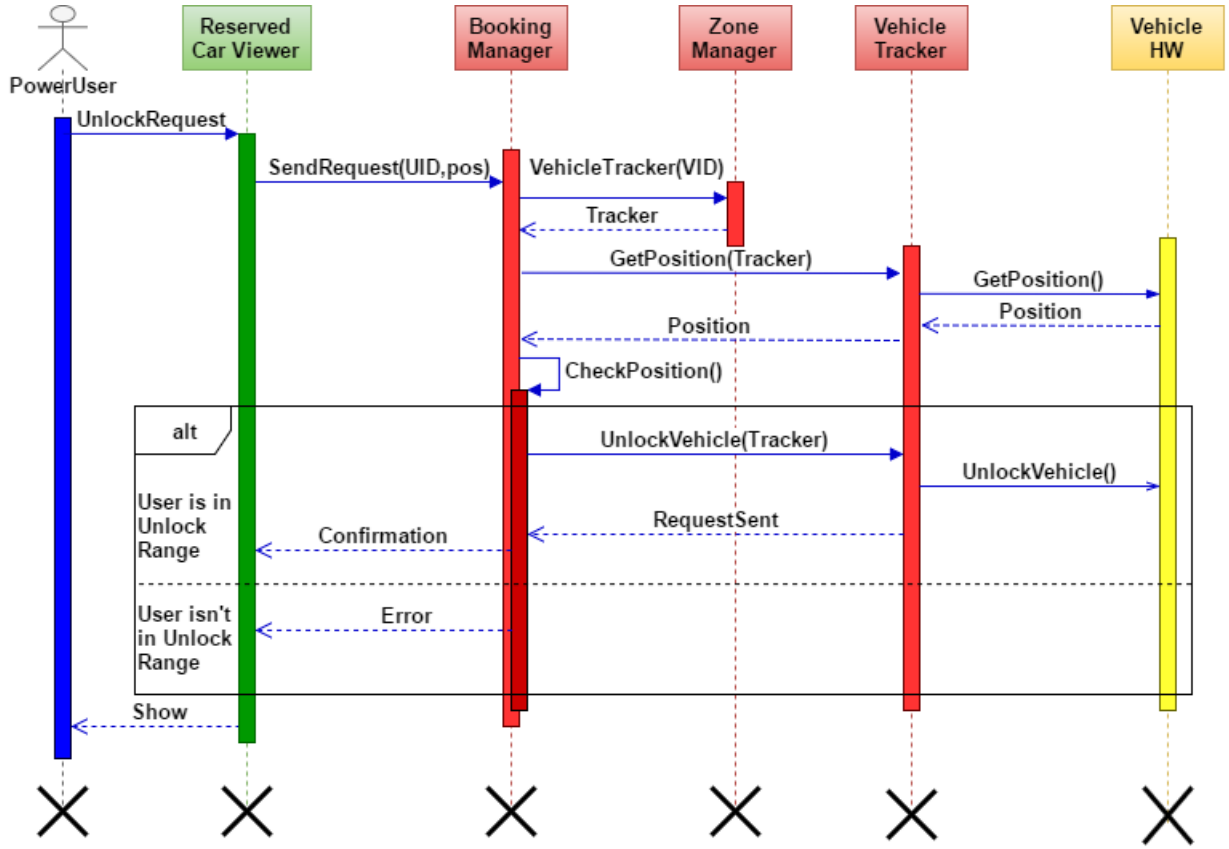


Figure 1: Car Unlocking

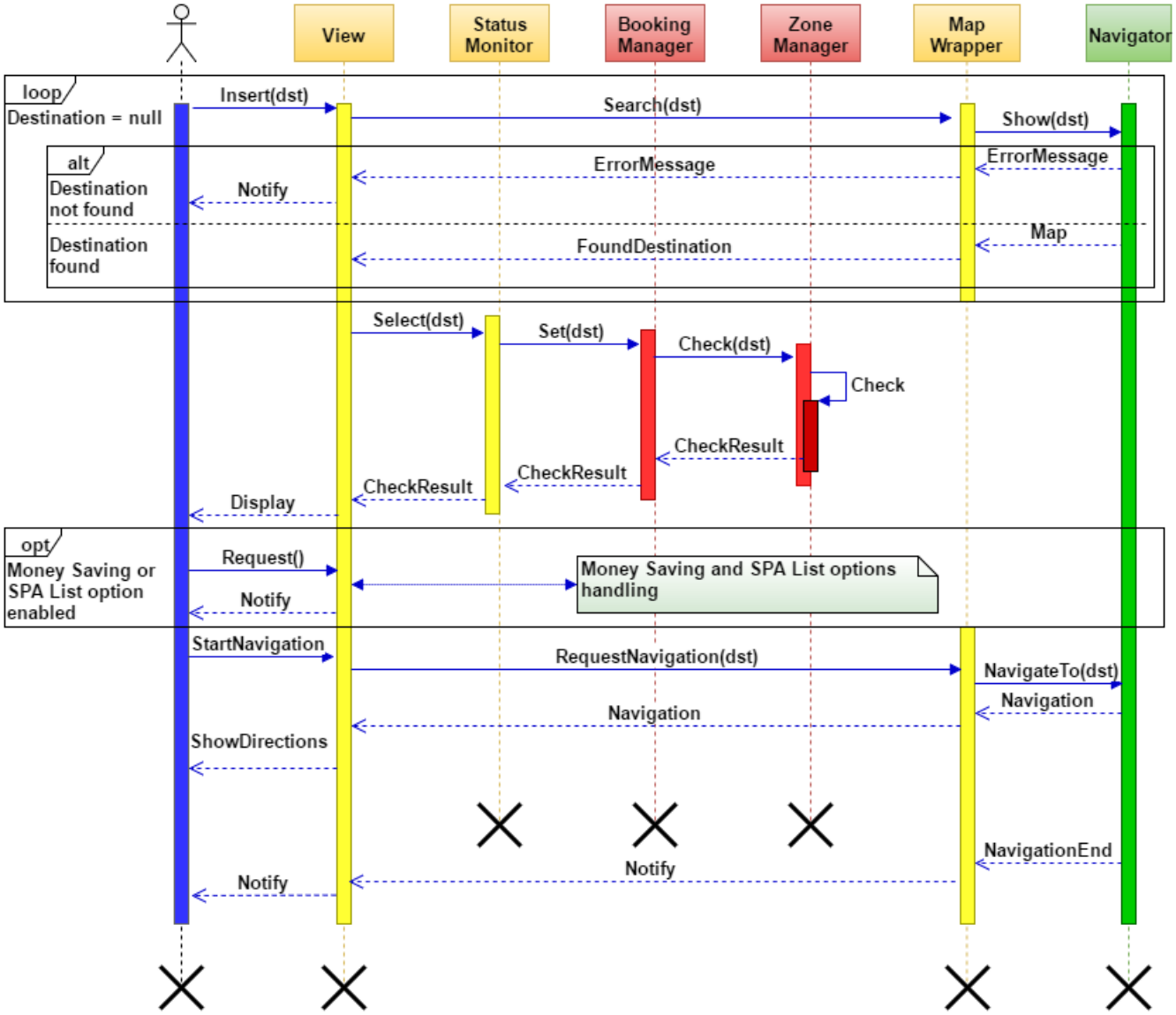


Figure 2: User's ride

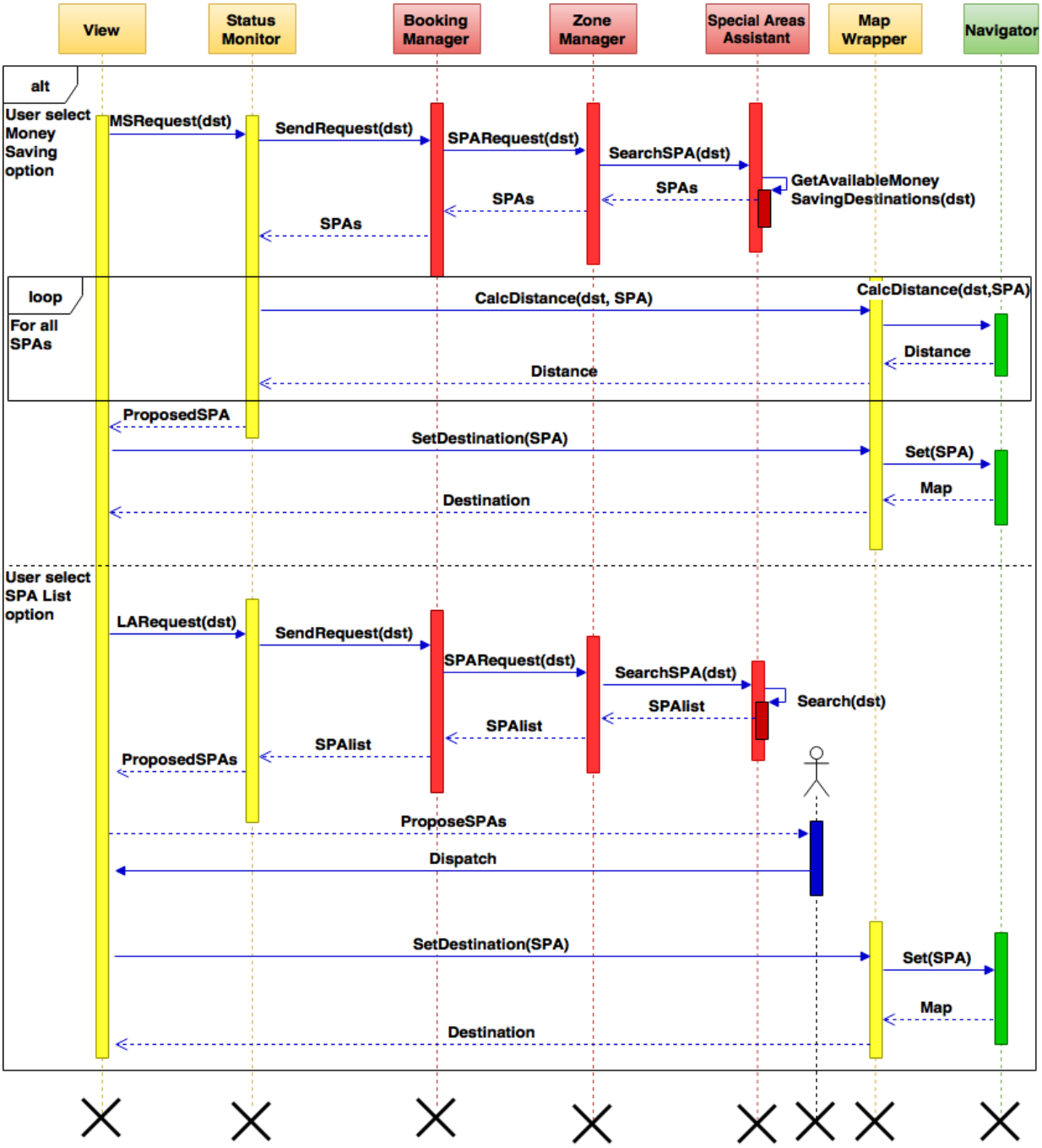


Figure 3: Money Saving and List Parking Area options

2.5 Component interfaces

In this section are defined the interfaces shown in the component view section. To make them easier to find they are listed in alphabetic order.

- **IAccountable**
  It is the interface to the Account Manager of the SBL through which it can handle requests about signing in, registration to the service and modification of account data.

- **IAccountingProvider**
  It is the interface the Client API/Accounting component shows to client-side apps. Via this interface the Account component of client-side apps can access the functions needed to signing in, register a new user, modifying user data.

- **IBookable**
  It is the interface to the SBL through which it can handle requests regarding positions of available Cars, their status, position of the reserved Car, unlocking of the reserved Car, positions of the Special Parking Areas close to the user provided destination, ride data, discounts and penalties applicability evaluation and any other active ride information.

- **IBookingProvider**
  It is the interface the Client API/Booking component shows to client-side apps. Via this interface the apps can submit their request about Cars positions, Cars status, Car unlocking, Special Parking Areas locations, ride details.

- **IConnectable**
  It is the interface that gives access to hardware sensor data of a vehicle. In particular about its location, the engine status, ignited or not, battery level and any other available sensor data.

- **IDataProvider**
  It is the interface provided by the entity framework the PEJ Context relies on to get bidirectional access to data in form of object entities whenever one of the SBL components requests or store a piece of data.

- **IExplorable**
  It is the interface in charge of providing the components which use it with the set of functions required to explore the map, such as zoom and scrolling.

- **IMailer**
  It is the interface used to send to Mail Server the recipient email address and content that must be sent.

- **IManagableAccount**
  It is the interface the Booking Handler relies on to get the data of the user who is making a reservation.

- **IMapProvider**
  It is the interface the Map Wrapper uses to get the data on the map, but it provides other functions too; in particular the possibility to look for a specific address, and by doing so checking whether that address exists or not, and to get the distance between two provided locations.

- **INavigable**
  It is the interface each View has to expose in order to support navigation and acceptance of input viewmodels.

- **IPaymentProvider**
  It is the interface exposed by Payment Processor accepting payment requests and returning transaction outcome on completion.

- **IPaymentRequest**
  It is the interface the Booking Handler uses to create a pending payment for a particular user. Via this interface the ride reference is passed to the Payment Processor in order to apply discounts or penalties before issuing the payment.

- **IQueryable**
  It is the interface the Special Areas Assistant exposes to be queried about the positions of the Special Parking Areas, in particular via this interface the Zone Manager can answer to the Booking Handler requests about the list of the closest SPA to a provided location.

- **ITrackable**
  It is the interface the Zone Manager uses to check the position and the other data of each vehicle. Furthermore it is via this interface that the Zone Manager can receive inform about engine ignition or shut down.

- **IViewable**
  It is the interface that components with visual contents expose to the View component in order to provide the information and the data to be rendered.

- **IZoneService**
  It is the interface the Zone Manager provides the Booking Handler with the set of functions required by the latter to accomplish its tasks, including getting vehicles location, asking for Car unlocking, asking for the closest set of SPA from a given location. Furthermore the Zone Manager uses this interface to notify the Booking Handler about vehicle status changes, such as location or engine ignition.

2.6 Selected architectural styles and patterns

The solution described in this document strongly embrace modern cloud-oriented philosophy. We believe the PowerEnJoy service has those key characteristics that made the choice of a cloud architecture to fit very nicely our customer needings. As most significant ones we would like to underline:

- The PowerEnJoy system will be developed from scratch, this allows the adoption of most convenient and future-proof architectural designs.
- PowerEnJoy is a start-up company, cloud Infrastructures as a Service are well-known to minimize initial deployment costs and risks of capital in case of business failure.
- Car sharing services, at the moment are experiencing a significant growth in terms of user adoption, therefore exists a concrete chance that the architecture would be heavy loaded even on service launch date which will probably require huge resources deployment that will take relatively long time to be completed with on-premise infrastructures.

2.6.1 Cloud Design Patterns

Designing cloud-ready software solutions has an immediate positive side-effect: each single taken design decision must always keep that elasticity and abstraction level the environment itself is characterized by. This practically turns out into helping solution architects to avoid many kinds of bad and harmful decisions that often come from technological concerns. The PowerEnJoy software platform this document describes follows most common recommended guidelines and best practices about roles subdivision across virtual compute units, message queuing and processing, data replication and caching. For more detailed description of those concepts you can refer to the MSDN library article you find in the External references section (§7.1) of this document.

2.6.2 Architectural Design Patterns

The whole PowerEnJoy software platform consists of a distributed, multi-tier architecture with a presentation layer split into a common front-end proxy and two different remote client-side apps, a middle tier that implements the PowerEnJoy business logic, a data tier which splits into multiple datasources and an IoT tier which includes both remote embedded devices and incoming telemetry data proxy. Each remote client-side app adopt an MVC style pattern, but its actual interpretation varies across different frameworks supported by the target OSs.

2.6.3 Communication Design Patterns

Given the distributed architectural layout, we had to keep particular attention designing each communication layer across various types of components. Where appropriate we have adopted a different approach always trying to leverage well-known design patterns in conjunction with open communication protocols that make much easier to integrate components based on heterogeneous technologies.

For interconnection of front-end and back-end roles a standard Enterprise Service Bus has been adopted. Being its queuing and dispatching mechanisms both REST-based and

event-driven, it accommodates our necessity of distributing message processing workloads across multiple different worker instances.

Client-side apps are connected to front-end roles by typical REST-based Client-Server patterns but are also able to receive asynchronous messages from the server via standard push notification service which basically resemble a Publisher-Subscriber channel. All messages sent through REST channels are encoded in JSON format. The choice of JSON/REST protocol suite over XML/SOAP has been adopted to minimize network loads caused by unnecessary XML verbosity, and also to maximize compatibility with existing messaging services from major cloud providers.

The connection between Vehicle Embedded Platforms is built over open AMQP/1.0 protocol, which turns out to be the most efficient one that integrates all security and performance features that the communication to IoT devices is expected to have. Those AMQP messages are processed by an Event Hub, a kind of Service Bus specifically designed to collect event streams at high throughput, which makes it perfect for IoT telemetry and management scenarios.

2.7 Other design decisions

Following paragraphs refer to relevant design decisions not explicitly explained anywhere else in this document.

The SBL supports configurable timeout values for any event regarding a ride. This means no software upgrade is needed to change those values in case of requirements changes.

The SBL supports configurable discounts policies. This means that each required feature regarding discount applicability can be modified or updated in future without any software upgrade. Each discount policy is defined by a set of conditions to be satisfied at a specific point of a booking (on Car unlocking, on engine turn-off, on Car locking) and parameters that allow to modify the charged fare (percentage discount ratio, absolute discount amount).

As said in previous chapters, each zone served by PowerEnJoy can override any global setting from the shared configuration datasource.

The client-side apps are expected to evolve much more frequently with respect to other service components. Therefore sounds reasonable not to define further the MVC pattern variant technicalities, taking advantage of any platform specific framework which best integrates with the target at the time of development. These components are designed such that the development of a brand-new version based on a completely different MVC framework should complete in short time and without generating issues to be addressed into other components of the solution, in the event of new target OSs release/update.

3 Algorithm Design

3.1 Getting money saving destination

3.1.1 Part 1

The first algorithm runs in the Special Areas Assistant component and it takes care of looking for the available Special Parking Areas located inside a certain range from the specified destination.

- The *isFull* method returns true if the Special Parking Area is full and no further vehicle can be parked there and plugged to the power grid.
- The *getRange* method returns the value of the range of the provided zone.
  This way the algorithm can be adapt to zones with different conformations just by changing the value of the range.
- The *calcAirDistance* method returns the air distance from the provided position to the the known position of the provided Special Parking Area. The air distance is used in order to keep low the level of the required computational power.

Algorithm 1

```
1: function getAvailableMoneySavingDestinations(DestPosition)
2:     var MSDestinations[] = empty
3:     var SpecialParkAreas[] = PEJContex.getSPAreas(DestPosition.getZone)
4:     for (int i=0; i<SpecialParkAreas[].count; i++) do
5:         if not(SpecialParkAreas[i].isFull) and
6:             (PEJContex.getRange(DestPosition.getZone)>
7:             calcAirDistance(DestPosition, SpecialParkAreas[i])) then
8:             MSDestinations[].append(SpecialParkAreas[i])
9:     return MSDestinations[]
10: end function
```

3.1.2 Part 2

The second algorithm runs in the Status Monitor component and it uses the list of feasible Special Parking Areas returned by the previous algorithm to establish which one is actually the closest to the destination.

- The *calcDistance* method returns the street distance from the provided position to the the known position of the provided Special Parking Area contained in *MSDestinations*. This real world distance is provided relying on the Navigator that can compute the actual distance between two provided positions.

Algorithm 2

```
1: function getMoneySavingDestination(MSDestinations[])
2:     var Distance = 99999
3:     if MSDestinations[]==empty then
4:         return errorMsgNoMoneySavingDestinationFound
5:     for int i=0; i<MSDestinations.count[]; i++ do
6:         if (Distance>calcDistance(DestPosition, MSDestinations[i])) then
7:             Distance = calcDistance(DestPosition, MSDestinations[i])
8:             MSDestination = MSDestinations[i]
9:     return MSDestination
10: end function
```

3.2 Total Ride Fare

This algorithm runs in the Payment Processor component and it takes care of applying any discount or penalty the user is eligible for at the end of the ride. In case the user has never started the ride, after the reservation expiration, this algorithm applies the penalty fee. Furthermore, it checks whether the user has any pending payment in order to add it to the total ride fare.

- The *getTotalBaseFare* method returns the Total Base Fare value which depends only on the amount of time of the Ride.
- The *hasStarted* method returns true if the user has ignited the engine of the Car at least once; otherwise returns false.
- The *getDiscounts* method is provided via PEJ Context, given a zone it returns the discounts and the penalties that can be applied in that particular zone.
  This way this algorithm can adapt to new discounts or penalties, or to removal of one of them, without needing of any modification.
- The *isApplicable* method returns true if that particular discount can be applied to that particular ride.
- The *applyDiscount* method compute the new value of the Total Ride Fare after the applicatin of the discount or penalty.
- The *getExpirationFee* method returns the amount of fee the user is charged with in case of expired reservation.
- The *hasPendingPayment* method return true if there is at least one payment for that particular user that has not completed successfully and still need to be collected.
- The *getPendingPaymentValue* method return the value of the money the user still owe the company from previous rides.

Algorithm 3

```
1: function caculateTotalRideFare(Ride)
2:     var TotalRideFare = Ride.getTotalBaseFare
3:     if Ride.hasStarted then
4:         for (Discount in PEJContex.getDiscounts(Ride.getZone)) do
5:             if Ride.isApplicable(Discount) then
6:                 TotalRideFare=TotalRideFare.applyDiscount(Discount)
7:     else
8:         TotalRideFare = PEJContex.getExpirationFee(Ride.getZone)
9:     if Ride.getUser.hasPendingPayment then
10:        TotalRideFare+=Ride.getUser.getPendingPaymentValue
11:    return TotalRideFare
12: end function
```

4 User Interface Design

Fully explained mockups of the user interface can be found in the RASD document.
In this section are presented the UX diagrams of the user experience of both client applications.

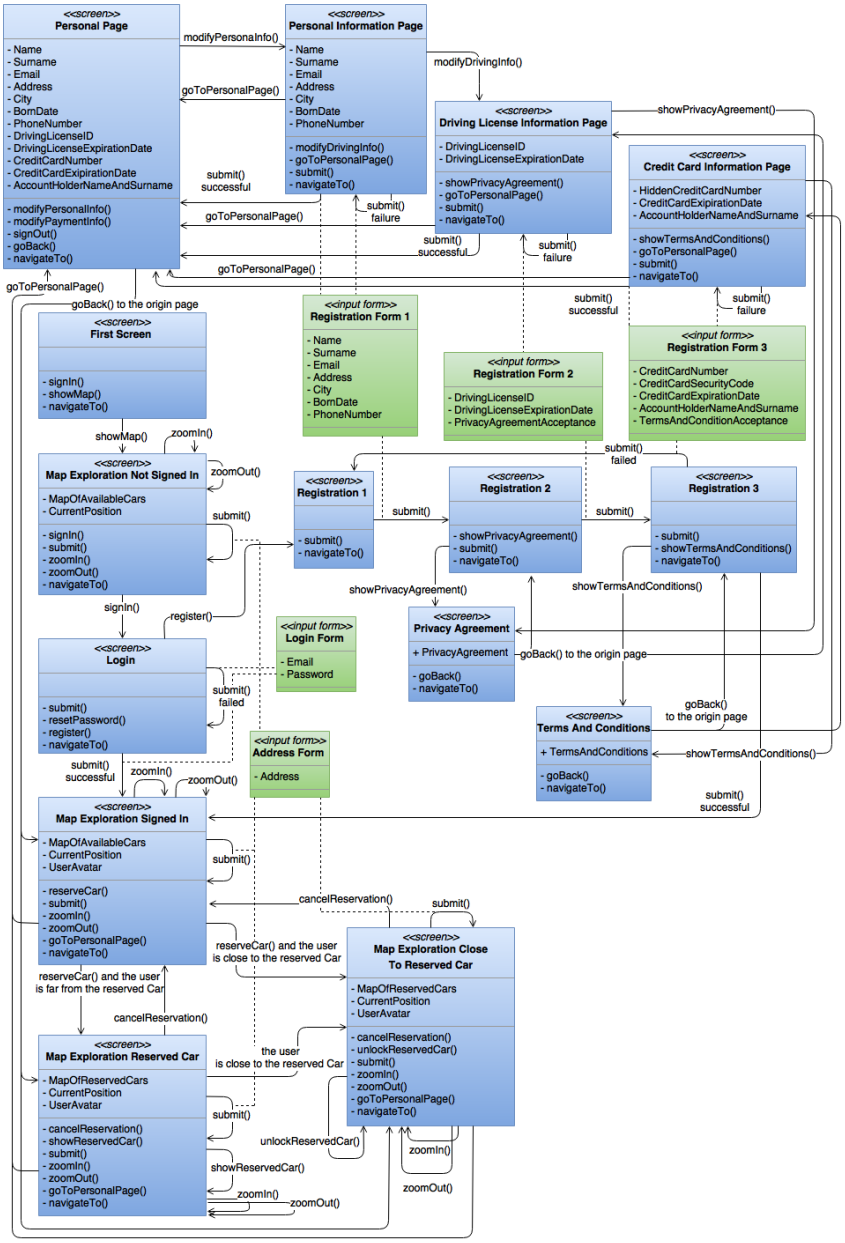


Figure 4: User Device Mobile App
For better readability, two larger images of the same diagram have been put on next pages.

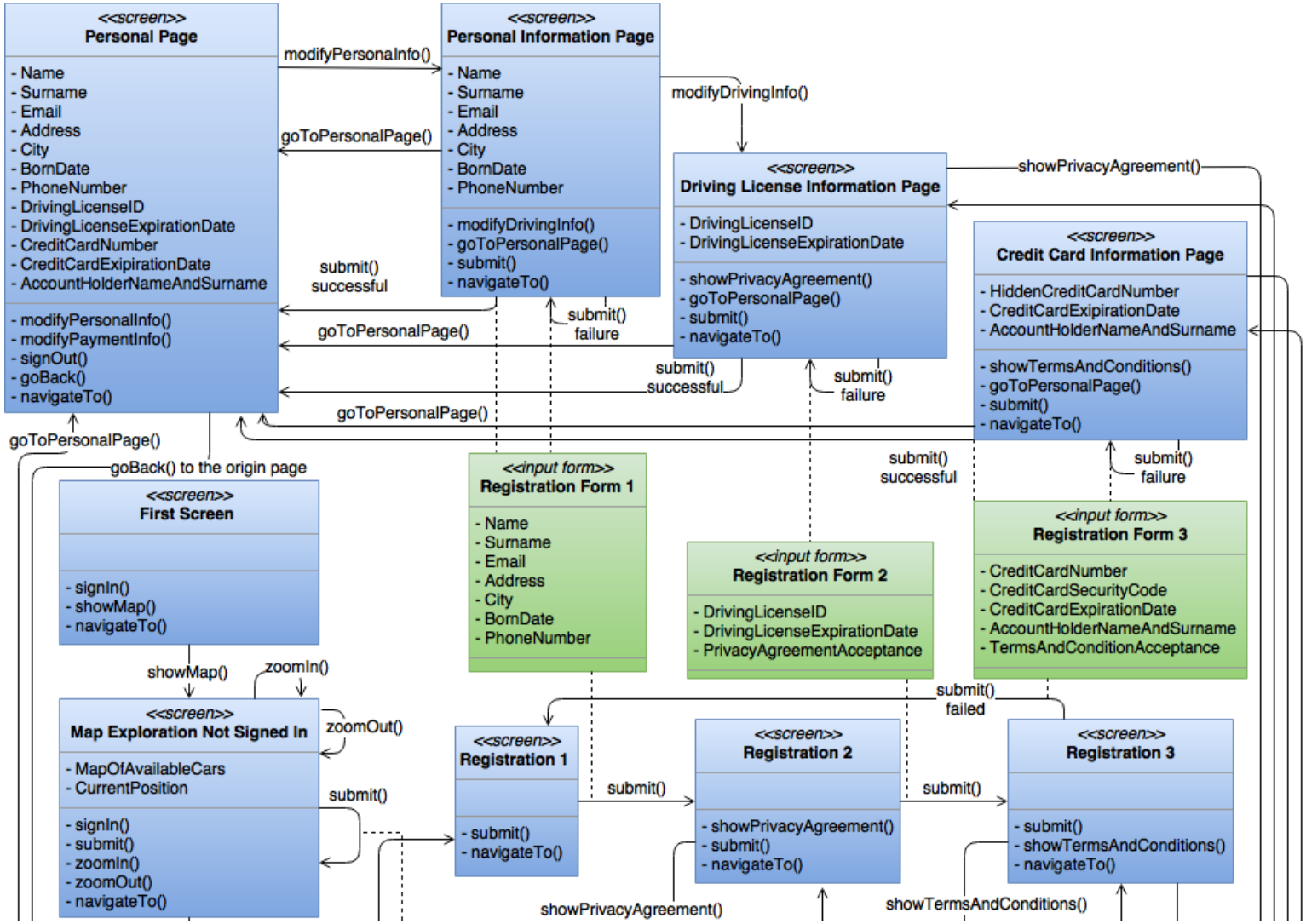


Figure 5: User Device Mobile App - Part 1

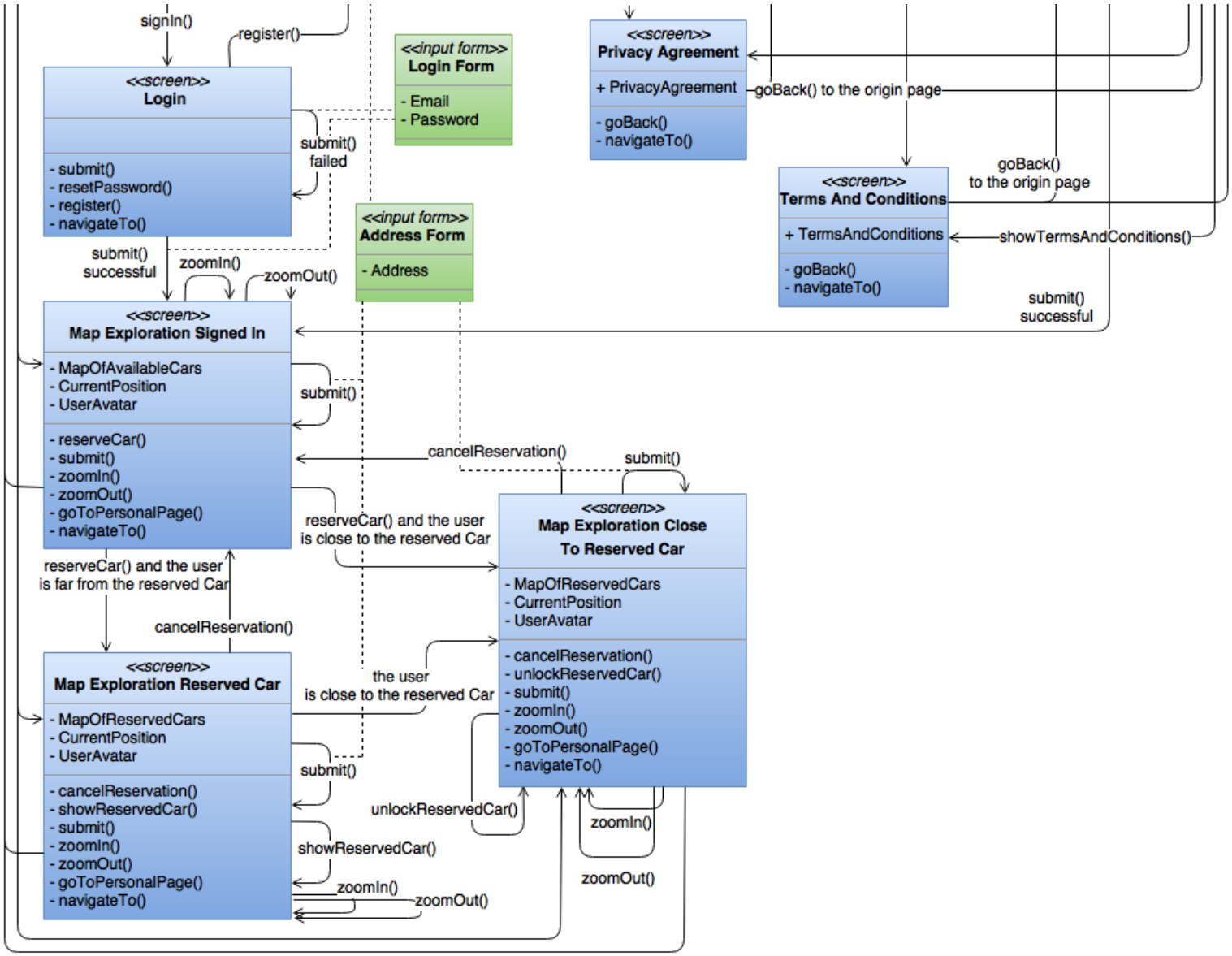


Figure 6: User Device Mobile App - Part 2

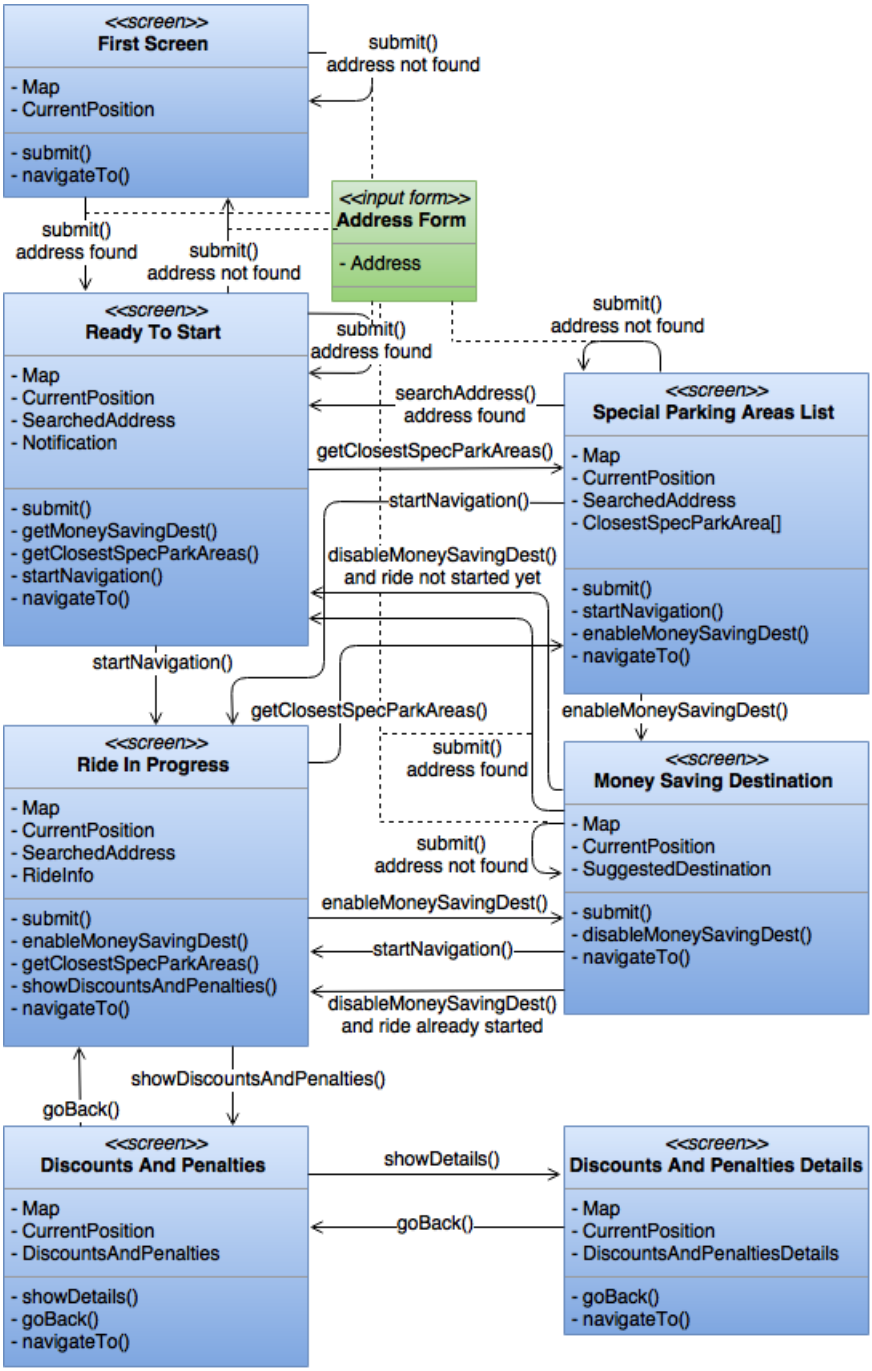


Figure 7: Onboard Ride Assistant App

5 Requirements Traceability

5.1 Goal 1

Allow any kind of user to view the map of the available nearby Cars.

- |R1| Users must have access to a map indicating the user current location.
  - 2.2.2 Map Explorer component.
  - 4 User Interface Design Figure 4.

- |R2| Users must be able to pan and scroll a map in any direction.
  - 2.2.2 Map Wrapper component.
  - 4 User Interface Design Figure 4.

- |R3| Every available Car must be shown on a map.
  - 2.2.2 Map Explorer component.
  - 4 User Interface Design Figure 4.

- |R4| Users shall have an input for inserting an address in which center the map.
  - 2.2.2 View component.
  - 4 User Interface Design Figure 4.

- |R5| Users shall have an input to center the map view on their position.
  - 2.2.2 View component.
  - 4 User Interface Design Figure 4.

5.2 Goal 2

Allow Visitor user to register to the service.

- |R6| Let a Visitor user start the registration wizard while he's still not logged in.
  - 2.2.2 View & Account components.
  - 4 User Interface Design Figure 4.

- |R7| The registration form must contain input fields for user identity, email, contact info, driving license number and expiration, privacy agreement confirmation, credit card number and expiration, billing identity.
  - 2.2.2 View & Account components.
  - 4 User Interface Design Figure 4.

- |R8| The chosen email address must not be already used by another PowerUser.
  - 2.2.1 Account component.
  - 4 User Interface Design Figure 4.

- |R9| The credit card must be verified not to be blocked or expired upon registration.
  - 2.2.1 Account component.
  - 4 User Interface Design Figure 4.

- |R10| The user must receive a system generated password to the registered email address.
  - 2.2.1 Account component.
  - 4 User Interface Design Figure 4.

5.3 Goal 3

Allow Visitor user to log-in and out as a PowerUser.

- |R11| A Visitor user must always see an input to access log-in form as long as he's still not logged in.
  - 2.2.2 View & Account components.
  - 4 User Interface Design Figure 4.

- |R12| An input to perform log-out must always be available to the PowerUser if he's currently logged in.
  - 2.2.2 View & Account components.
  - 4 User Interface Design Figure 4.

5.4 Goal 4

Allow PowerUser to check the status of the Car.

- |R13| For each available Car the PowerUser must be able to view its remaining battery charge.
  - 2.2.2 Map Explorer component.
  - 4 User Interface Design Figure 4.

- |R14| For each available Car the PowerUser must be able to view its current position.
  - 2.2.2 Map Explorer component.
  - 4 User Interface Design Figure 4.

5.5 Goal 5

Allow PowerUser to reserve a Car.

- |R15| The PowerUser must have the ability to start the reservation wizard for all and only available Cars.
  - 2.2.2 Map Explorer component.
  - 4 User Interface Design Figure 4.

- |R16| The PowerUser must see a reminder about unfulfilled reservation penalty before confirmation.
  - 2.2.2 Map Explorer component.
  - 4 User Interface Design Figure 4.

- |R17| Show an input to allow the PowerUser to confirm and finalize the reservation.
  - 2.2.2 Map Explorer component.
  - 4 User Interface Design Figure 4.

- |R18| The system shall prevent the PowerUser to reserve more than a Car from the same geographical region at a time.
  - 2.2.1 Booking Handler component.

5.6 Goal 6

Allow PowerUser to cancel a reservation.

- |R19| If a reservation exists for the PowerUser, show him an input to request cancellation.
  - 2.2.2 Reserved Car Viewer component.
  - 4 User Interface Design Figure 4.

- |R20| The system shall prompt the PowerUser for cancellation confirmation.
  - 2.2.1 Booking Handler component.
  - 4 User Interface Design Figure 4.

- |R21| A reservation must be automatically cancelled by the system after 1 hour from its creation.
  - 2.2.1 Booking Handler component.

- |R22| If a reservation is cancelled because of timeout, notify the PowerUser about that occurrence.
  - 2.2.1 Booking Handler component.

5.7 Goal 7

Allow PowerUser to check the position of the reserved car.

- |R23| As long as a reservation exists for the PowerUser he must always be able to get the position of the reserved Car on the map.
  - 2.2.2 Reserved Car Viewer component.
  - 4 User Interface Design Figure 4.

5.8 Goal 8

Allow PowerUser to unlock and enter the Car when inside the specific range.

- |R24| The system must be able to remotely unlock the Car.
  - 2.4 Runtime View Figure 1.
  - 2.2.1 Booking Handler component.

- |R25| The system must be able to compute the distance between the user location and his reserved Car.
  - 2.4 Runtime View Figure 1.
  - 2.2.1 Booking Handler component.

- |R26| The PowerUser must have an input allowing him to send an unlock request.
  - 2.4 Runtime View Figure 1.
  - 2.2.2 Reserved Car Viewer component.
  - 4 User Interface Design Figure 4.

- |R27| The system must accept the unlock request issued by the PowerUser if and only if the PowerUser is in the unlock allowance area.
  - 2.4 Runtime View Figure 1.
  - 2.2.1 Booking Handler component.

- |R28| If the unlock request is accepted, the PowerUser must be able to enter the Car.
  - 2.4 Runtime View Figure 1

5.9 Goal 9

Allow PowerUser to get driving directions to his destination.

- |R29| The user must be allowed to select a custom destination and start navigating to that location.
  - 2.4 Runtime View Figure 2.
  - 2.2.3 Map Wrapper component.

5.10 Goal 10

Bill the PowerUser for the amount of time spent riding a Car.

- |R30| Start counting the billing time from the first engine ignition.
  - 2.2.2 Booking Handler component.

- |R31| Stop the billing time counter exactly 1 second after the Car locking.
  - 2.2.2 Booking Handler component.

5.11 Goal 11

Allow PowerUser to see a list of the closest Special Parking Areas to his destination.

- |R32| The system must be capable of providing a list of Special Parking Areas sorted by distance from an input location.
  - 2.4 Runtime View Figure 2 & Figure 3.
  - 4 User Interface Design Figure 7.

- |R33| PowerUser must be allowed anytime during the navigation to input a custom location and be acknowledged about all nearest Special Parking Areas from the selected location.
  - 2.4 Runtime View Figure 2 & Figure 3.
  - 2.2.3 Map Wrapper & View components.
  - 4 User Interface Design Figure 7.

5.12 Goal 12

Allow PowerUser to keep track of the Current Fare.

- |R34| Show on the Car screen a live updated counter indicating the Current Fare amount that the user would actually pay if the ride ended in that same moment,as long as the ride is being charged.
  - 2.2.3 Status Monitor component.
  - 4 User Interface Design Figure 7.

5.13 Goal 13

Allow PowerUser to check whether he can be eligible for any discount or penalty.

- |R35| Provide through Car screen an input to access an overview of all discounts and penalties.
  - 2.2.3 Status Monitor component.
  - 4 User Interface Design Figure 7.

- |R36| For each shown discount or penalty allow the PowerUser to get a brief informal description of corresponding criteria.
  - 2.2.3 Status Monitor component.
  - 4 User Interface Design Figure 7.

- |R37| For each shown discount or penalty allow the PowerUser to know if the current ride satisfies all needed criteria at the moment.
  - 2.2.3 Status Monitor component.
  - 4 User Interface Design Figure 7.

5.14 Goal 14

Allow PowerUser to get a money saving alternative destination.

- |R38| Show the PowerUser the option to get a money saving destination alternative after entering desired destination address.
  - 2.4 Runtime View Figure 2 & Figure 3.
  - 3.1.1 Algorithm design Algorithm 1 & 2.
  - 4 User Interface Design Figure 7.

- |R39| Always show to PowerUser an input to get money saving proposal even if there already exists a selected destination.
  - 2.4 Runtime View Figure 2 & Figure 3.
  - 4 User Interface Design Figure 7.

- |R40| The destination proposal must correspond to the nearest (w.r.t. PowerUser selected destination) Special Parking Area where there are less than N# Cars attached to the power source.
  - 2.4 Runtime View Figure 2 & Figure 3.
  - 3.1.1 Algorithm design Algorithm 1 & 2.

- |R41| If the PowerUser accepts the money saving destination proposal, the current selected destination must be updated accordingly.
  - 2.4 Runtime View Figure 2 & Figure 3.
  - 4 User Interface Design Figure 7.

5.15 Goal 15

Allow the system to lock the Car in a Safe Parking Area at the end of the ride.

- |R42| The system must lock the Car if its position belong to the set of Safe Parking Areas, engine is stopped, all doors are closed and S# seconds passed from the last door closure.
  - 2.2.1 Booking Manager component.

5.16 Goal 16

Allow the system to apply penalty or discount according to the given criteria.

- |R43| The system shall apply a discount of 10% on the last ride Total Base Fare if the number of passengers at the end of the ride is greater or equal to the number of passengers at the start of the ride and the number of passengers at the start of the ride was at least 3, driver included.
  - 2.2.1 Booking Manager component.

- |R44| The system shall apply a discount of 20% on the last ride Total Base Fare if the remaining battery power at the end of the ride is greater or equal then 50%.
  - 2.2.1 Booking Manager component.

- |R45| The system shall apply a discount of 30% on the last ride Total Base Fare if the Car position at the end of the ride is within a Special Parking Areas and the power socket is detected as connected two minutes from the Car locking.
  - 2.2.1 Booking Manager component.

- |R46| The system shall apply a penalty of 30% on the last ride Total Base Fare if the position of the Car at the time of locking is more than 3km far from the nearest power grid station, or the remaining battery power is less than 20% and the Car is

not detected as attached to power grid within two minutes from locking.
- 2.2.1 Booking Manager component.

5.17 Goal 17

Let the system bill the PowerUser for the Total Ride Fare and issue a payment request for that amount at the end of the ride.

- |R47| The system must charge the PowerUser the Total Ride Fare after two minutes and an half from the Car locking.
  - 2.2.1 Booking Manager component.
- |R48| The system must issue the payment request to the Payment Processor Provider.
  - 2.2.1 Payment Processor component.
- |R49| The system must bill the PowerUser a penalty of 1e as soon as a reservation he made expires by timeout.
  - 2.2.1 Booking Manager component.
- |R50| The PowerUser shall be notified of any money charge by email.
  - 2.2.1 Account Manager component.
- |R51| The PowerUser shall be notified of the payment transaction result.
  - 2.2.1 Account Manager component.

6 Effort Spent

Teamwork	∼13h
Piccirillo Luca	∼28h
Zampogna Gian Luca	∼21h
Zini Edoardo	∼21h

7 References

7.1 External reference documents

- IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications.
- UX diagram PDF from the course "Sistemi Informativi" of Politecnico of Milan.
- Cloud Design Patterns: Prescriptive Architecture Guidance for Cloud Applications MSDN library, January 2014, visited on the 20th of November 2016 (`https://msdn.microsoft.com/en-us/library/dn568099.aspx`)
- See section 1.4 for internal references.

7.2 Document revisions history

Version	**Date**	**Changes**
1.0-RC1	11/12/2016	First deadline release.