Design Document

Simone Mosciatti & Sara Zanzottera

December 9, 2016

Contents

1 Introduction

1.1 Purpose

This Design Document aims to provide to everyone involved in the actual development of the application specific insights about the structure of PowerEnJoy, its acthitecture's details, the desing patterns we chosed to implement, but also some details about its high level components, their interactions and general behavior.

1.2 Scope

PowerEnJoy is a digital management system for car sharing that exclusively employs electric cars to provide its service. The system provides all the functionalities normally provided by a car sharing service: registering to the service, find the location of nearby available cars, reserve cars up to a short amount of time, unlock the chosen car once found, ride it and then park it in a safe area, when it will be automatically locked and the fee paid.

In addition, the system gives bonuses and penalities in term of discounts or over-prices depending on the behavior of the user, in order to promote virtuous behaviors.

PowerEnJoy is therefore a inherently distributed system, based on a central server interactions with many distributed nodes. In detail the system can be divided into four main parts:

- a public app, used by customers to access the service
- a centralized backend that provides the service
- the cars' onboard system, that communicates only with the centralized backend
- a reserved fronted, used exclusively by the staff members to better organize their job

All these four components will be examined in more detail in the subsequent sections of the document.

1.3 Definitions, Acronyms, Abbreviations

RASD Requirements and Specification Document.

DD Design Document.

User A customer of PowerEnJoy using the service.

Staff Operator An employee of PowerEnJoy which takes care of the cars.

Ride The action of getting onboard of a PowerEnJoy car, start its engine, drive to destination and park.

Running Time The time an user spends using the PowerEnJoy service.

Issue Any problem a car may incur in, or a user may face while using the service.

Nearby Cars Cars located within a maximum distance to a specific position.

Nearby Issues Issues that are affecting cars close to a specific position.

Booking (Reservation) The act to reserve a car for a limited amount of time for future use by a user.

Reservation's maximum time The maximun amount of time a car can be reserved.

Driver Whoever is driving a regularly booked PowerEnJoy car.

Passenger Whoever is in inside a PowerEnJoy car but is not the driver.

Driving License The state's issued driving license of the user.

Notification A form of comunication where the user is actively notified of some event.

Issue Report An incoming notification that states a car incurred in an issue.

Fine A fine issued by the local law enforcing officers to a user while driving a PowerEnJoy car.

Pending Bills Bills that an user still need to pay to PowerEnJoy .

Safe Area An parking area, predefined by the company, where is possible to safely park the cars of the PowerEnJoy fleet.

Battery Charge The amount of charge that is kept inside the car's battery.

Charging Station Dedicated areas where is possible to plug the PowerEnJoy cars to charge their batteries.

Car's Onboard System The controll system of the car that is able to exchange data with the central system and to relevate operation parameters.

Customer's App An implementation of the system frontend tailored to the need of the customers.

Operator's App An implementation of the system frontend tailored to the need of the staff.

Central System The central system for PowerEnJoy . All the command and all the data are streamed, analyzed and used here.

Credentials Pair {Username, Password} necessary to access the PowerEnJoy system.

GPS : Global Positioning System is a global navigation satellite system (GNSS) that provides location and time information in all weather conditions, anywhere on or near the Earth where there is an unobstructed line of sight to four or more GPS satellites.

System's Frontend The interface provided to the user of the PowerEnJoy system.

System's Backend The whole technical infrastructure necessary to PowerEnJoy .

1.4 Document Structure

1. **Introduction**

   This sections aims to explain the purpose and the scope of the document, introducing the reader to subsequent sections of the document itself.

2. **Architectural Design**

   This sections will explain the main architectural decision we made.

3. **Algorithm Design**

   In this section we focus on the most critical code section and we provide an in-depth analysis of how they should be structured, eventually providing pseudocode for them.

4. **User Interface Design**

   In this section we carry on the UX design with the help of UX and BCE diagrams, eventually completing them with updated and extended application mockups.

5. **Requirements Traceability**

   In this section we map the requirements stated in the RASD to the actual component or processes that fulfill these requirements.

6. **Conclusions**

   In this section we enumerate the tools we used to redact this document, the hours of work spent by each group member and the (eventual) revision history of the document itself.

1.5 Reference Documents

- *Assignments AA 2016-2017.pdf* (Assignments document given by the teacher)
- *Sample Design Deliverable Discussed on Nov. 2.pdf* (Sample document provided by the teacher)

2 Architectural Design

The overall design process has been carried in a bottom-up approach, starting from the analysis of goals and requirements moving upwards to the definition of the higher level components of the system. In the following sections we provide more details on the designed architecture.

2.1 Design Process Description

The overall design process starts from the physical structure of the system. Taking forward the considerations made in the RASD, we identify which physical nodes that are to be deployed in order to meet the requirements, and make the very first design choices onto them.

Then we analyse the interface between the world and the machine, according to our choices. Given the list of goals and requirements, we identify what interfaces each node should provide to the world and to other nodes in order to accomplish such goals.

Once the interfaces are identified, we proceed organizing those interfaces into higher-level components, caring at respecting the Single Responsability principle in order to provide highly decoupled and reusable components.

Once components have been defined, we organize them into an high level architecture and define in detail how they interact with the help of some Sequence Diagrams.

The rest of this section follows this flow, starting from the deploy analysis and finally providing the overall design.

2.2 From Functional Requirement to User Interfaces

In this part of the document we describe the iteraction between the user and the system.

Each iteration also describe what other functionality is required to actually works, those functionality are described using the convetion Sy/NameFunctionality, where "Sy" indicate an abstract system that we are going to model in the next parts.

UI/Registration

Description User can register to the system, the user need to provide personal information and payment information, the system check whenever the user is already registered and if the information provided are correct, then proceed with the registration and provide feedback to the user.

Requires

- Sy/Register ⇔ USER/Register
- Sy/Validate ⇔ USER/Validate

UI/Login

Description User can log into the system providing their username (email) and password.

Requires

- Sy/Login ⇔ USER/Login

UI/Lookup

Description User can lookup the position of cars near them or near a specific position.

Requires

- Sy/GeoLocationCars ⇔ GEOLOCATION/AvailableCars
- Sy/PositionUser ⇔ POSITION/User

UI/Book

Description User can book cars to use in the next our.

Requires

- Sy/Book ⇔ BOOKING/Book

UI/Unbook

Description User can remove a previously made prenotation.

Requires

- Sy/Unbook ⇔ BOOKING/Unbook

UI/Unlock

Description User can unlock a nearby car that was previously booked.

Requires

- Sy/PositionUser ⇔ POSITION/User
- Sy/PositionCar ⇔ POSITION/Car
- Sy/Unlock ⇔ CAR/Unlock

UI/Ride

Description User can ride a car.

Requires

- Sy/StartRide ⇔ RIDE/Start
- Sy/EndRide ⇔ RIDE/End
- Sy/ValidateLicense ⇔ CAR/ValidateLicense
- Sy/GeoLocationAreas ⇔ GEOLOCATION/Areas
- Sy/Lock ⇔ CAR/Lock

UI/SafeAreas

Description User can locate safe parking areas.

Requires

- Sy/GeoLocationAreas ⇔ GEOLOCATION/Areas

UI/UnsafeParking

Description The system react to unsafe parking.

Requires

- Sy/EngineOff ⇔ CAR/TurnOff
- Sy/Lock ⇔ CAR/Lock
- Sy/UnsafeParkNotification ⇔ NOTIFY/UnsafePark

UI/PowerStation

Description User can locate and use charging station.

Requires

- Sy/GeoLocationAreas ⇔ GEOLOCATION/Areas
- Sy/Pluged ⇔ CAR/Telemetry
- Sy/CarPluggedNotification ⇔ NOTIFY/CarPlugged

UI/Charge

Description User are charged a fee at the end of the ride or if a prenotation expires.

Requires

- Sy/SendFee ⇔ NOTIFY/SendFee
- Sy/BookExpire ⇔ BOOKING/Expire
- Sy/CalculateFee ⇔ BILL/Calculate

UI/Payment

Description User can pay the bills through the app and set the default payment method.

Requires

- Sy/MakePayment ⇔ BILL/Pay
- Sy/SetPaymentMethod ⇔ USER/SetPaymentMethod

UI/ReportIssues

Description User can report issues to the system.

Requires

- Sy/ReportIssue ⇔ ISSUE/New

UI/FindIssues

Description The staff can locate cars that need their intervention.

Requires

- Sy/GeoLocationIssues ⇔ GEOLOCATION/Issues

UI/Support

Description The staff can identify and solve car′s issues.

Requires

- Sy/Lock ⇔ CAR/Lock
- Sy/Unlock ⇔ CAR/Unlock
- Sy/TakeChargeIssue ⇔ ISSUE/TakeCharge
- Sy/SolveIssue ⇔ ISSUE/Solve
- Sy/GiveUpIssue ⇔ ISSUE/GiveUp

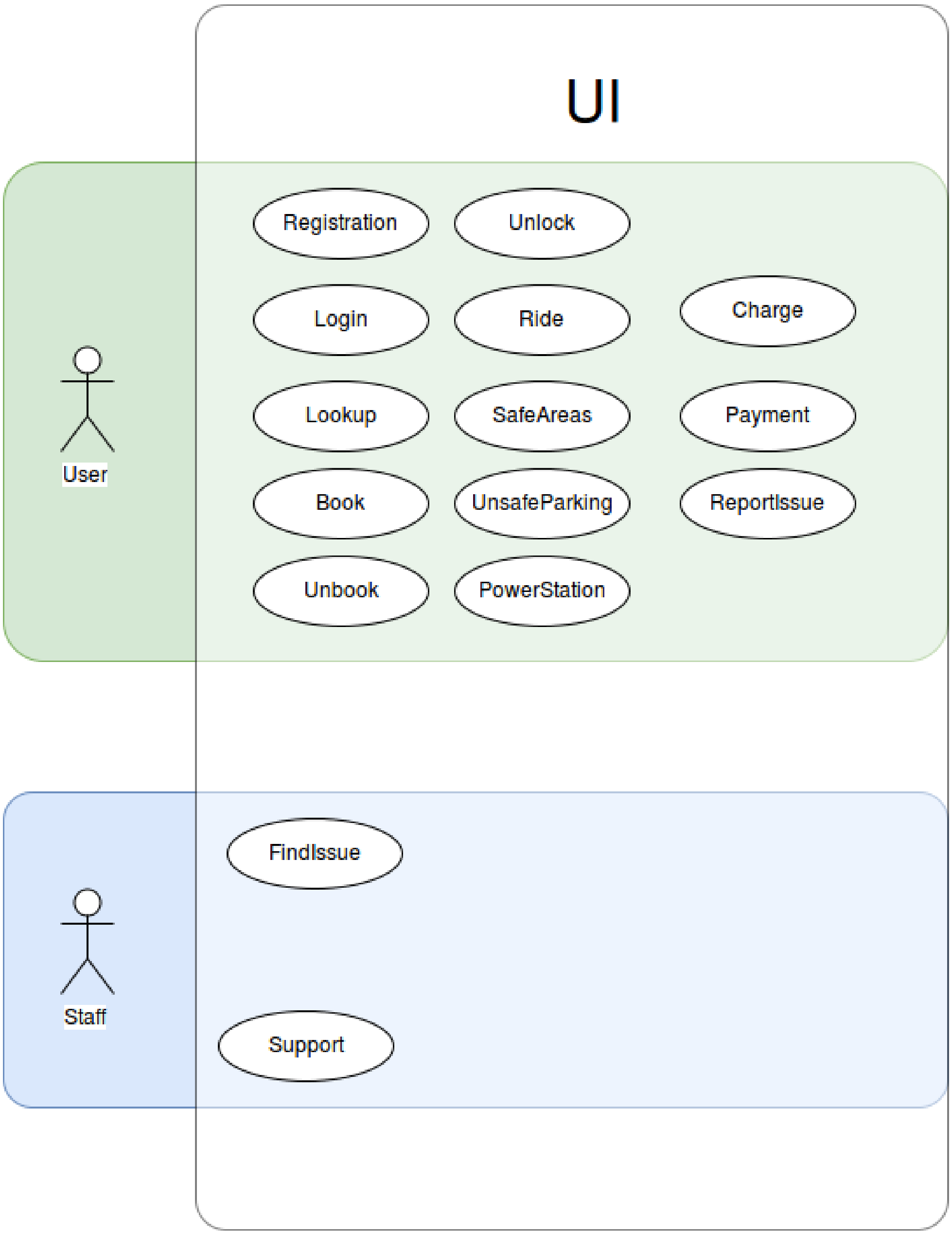

Figure 1: Graphical visulaization of the UI divided by tipology of users.

2.3 From Functionality to Componets

In this part of the document we are going to gather together all the functionality that we describe earlier and we are going to organize them in component.

Moreover we are also mapping, one by one each abstract functionality of the previous section with the functionality described in the components.

USER_MANAGER

Responsability Manages the users.

USER/Register ⇔ Sy/Register

Responsability Registers a new user into the system.

Input Information from the user such as:

- First name
- Last name
- Password
- Email
- License ID
- Credit card informations: credit card number, control code, expiry date, owner, etc.

Output The ID of the newly created user.

USER/Validate ⇔ Sy/Validate

Responsability Validate the information provided by the user, it makes sure that the user is not already registered into the system, that the License is valid as well the credit card information.

Input Information from the user such as:

- First name
- Last name
- Password
- Email
- License ID
- Credit card informations: credit card number, control code, expiry date, owner, etc.

Output Confirm correctness and validity of the informations.

USER/Login ⇔ Sy/Login

Responsability Allows users to log into the system.

Input Email (considered a unique user ID) and password.

Output A session key, meaning that the user is logged into the system.

USER/SetPaymentMethod ⇔ Sy/SetPaymentMethod

Responsability Update user's information about the preferred payment method.

Input The ID of the user and new payment informations.

Output The payment method data related to this user is updated.

GEOLOCATION

Responsability Locates elements, points and areas of interest around a specific coordinate. "Search" service for elements of interest.

GEOLOCATION/AvailableCars ⇔ Sy/GeoLocationCars

Responsability Retrives the position of available cars.

Input Search parameters such as:

- Geographical coordinates of the center of the search range (latitude and longitude as provided by GPS sensors)
- Maximum walking distance from the specified position
- Other search settings, like minimum battery level, etc.

Output A set of available cars matching the search parameters.

GEOLOCATION/Areas ⇔ Sy/GeoLocationAreas

Responsability Retrives the position of areas of interest, such as power stations and safe parking areas.

Input Geographical coordinates of the center of the search range (latitude and longitude as provided by GPS sensors) and a search radius.

Output A set of areas of interest inside the circle of radius provided centered on the coordinates provided.

GEOLOCATION/Issues ⇔ Sy/GeoLocationIssues

Responsability Retrives the position of cars with some issues.

Input Search parameters such as:

- Geographical coordinates of the center of the search range (latitude and longitude as provided by GPS sensors)
- Radius of the search
- Issue type, Exeption status, and other similar search settings.

Output A set of cars with issues matching the search parameters inside the circle of radius provided centered on the coordinates provided.

POSITION

Responsability Locates elements of interest given their ID. "Lookup" service for elements of interest.

POSITION/Car ⇔ Sy/PositionCar

Responsability Retrieves the position of a specific car.

Input The ID of the car.

Output The coordinates of the car.

POSITION/User ⇔ Sy/PositionUser

Responsability Retrieves the position of an user.

Input The ID of the user.

Output The coordinates of the user.

POSITION/Areas

Responsability Retrieves the position of an area of interest.

Input The ID of the area.

Output The coordinates of the area as a set of boundary points.

BOOKING_MANAGER

Responsability Manages reservations.

BOOKING/Book ⇔ Sy/Book

Responsability Books one available car.
Input The ID of the car and the ID of the user.
Output The car is booked and the ID of the reservation is provided.

BOOKING/Unbook ⇔ Sy/Unbook

Responsability Removes a reservation.
Input The ID of the user and the ID of the reservation.
Output The reservation is cancelled.

BOOKING/Expire ⇔ Sy/BookExpire

Responsability Removes an expired reservation and fines the related user.
Input ID of the reservation.
Output The reservation is cancelled and the user is fined.

CAR_MANAGER

Responsability Manages the iteractions between users and cars.

CAR/Unlock ⇔ Sy/Unlock

Responsability Unlocks the car.
Input The ID of the car and the ID of the user asking to unlock.
Output The car is unlocked.

CAR/ValidateLicense ⇔ Sy/ValidateLicense

Responsability Confirms the scanned license is related to the user who booked the car.
Input The scanned image of the driving license and the ID of the booking
Output The car is unlocked.

CAR/Lock ⇔ Sy/Lock

Responsability Locks the car.
Input The ID of the car.
Output The car is locked.

CAR/TurnOff ⇔ Sy/EngineOff

Responsability Turns off the engine of a car.
Input The ID of the car.
Output The car is turned off.

CAR/Telemetry

Responsability Retrieves real-time, updated informations about a car.
Input The ID of the car.
Output All the latest informations available about the car.

CAR/SetStatus

Responsability Sets the Exception status of a car to a new value.
Input The ID of the car and the new status.
Output The new status is set.

CAR/Rides

Responsability Retrieve the list of rides done with a specific car in a defined time range.
Input The ID of the car and a time range.
Output The list of rides performed with that car in that time range.

RIDE

Responsability Manage the rides.

RIDE/Start ⇔ Sy/StartRide **Responsability** Start counting the time of the ride.
Input The ID of the user and the ID of the booking.
Output The ID of the ride.

RIDE/End ⇔ Sy/EndRide **Responsability** Stop counting the time of the ride.
Input The ID of the ride.
Output Stop the count of time for the ride.

BILLING_SYSTEM

Responsability Manages all the fees.

BILL/Calculate ⇔ Sy/CalculatFee

Responsability Calculates the amount of a riding fee.
Input The ID of the ride.
Output The final fee, including eventual discounts or overprices.

BILL/Pay ⇔ Sy/MakePayment

Responsability Requires user to pay a specific bill.
Input The ID of the user and the ID of the ride the bill refers to.
Output The fee is paid.

ISSUE_MANAGER

Responsability Manages car's issues.

ISSUE/New ⇔ Sy/ReportIssue

Responsability Rise a new issue.
Input ID of the car, ID of the user raising the issue, a title and a description of the issue.
Output The ID of the reported issue.

ISSUE/TakeCharge ⇔ Sy/TakeChargeIssue

Responsability Allows operators taking in charge of a particular issue.
Input The ID of the issue, the ID of the operator.

Output The operator is now responsable for the issue.

ISSUE/Solve ⇔ Sy/SolveIssue

Responsability Allows operators to close an issue marking it as Solved. The system is resposible of allowing this operation only if it can confirm the issue has been solved.

Input The ID of the issue, the ID of the operator.

Output The issue is set as Solved, or set as Closed and another new issue is opened.

ISSUE/GiveUp ⇔ Sy/GiveUpIssue

Responsability Allows operators to give up over an issue.

Input The ID of the issue, the ID of the operator.

Output The issue is no more related to the specified operator and available for others to be took in charge.

NOTIFIER

Responsability This component takes care to notify user of events.

NOTIFY/SendFee ⇔ Sy/SendFee

Responsability Ask for payment to the user before to actually execute the transaction.

Input The ID of the user and the total to pay.

Output The user is ask to confirm the payment.

NOTIFY/UnsafePark ⇔ Sy/UnsafeParkNotification

Responsability Communicate at the user that he parked in an usafe area.

Input The ID of the user.

Output The user is prompt to acknoledge that he is aware to have parked in an unsafe area.

NOTIFY/CarPlugged ⇔ Sy/CarPluggedNotification

Responsability Communicate at the user that he succesfully plugged the car into a power station.

Input The ID of the user.

Output The user is prompt to acknoledge that he plugged the car into a power station.

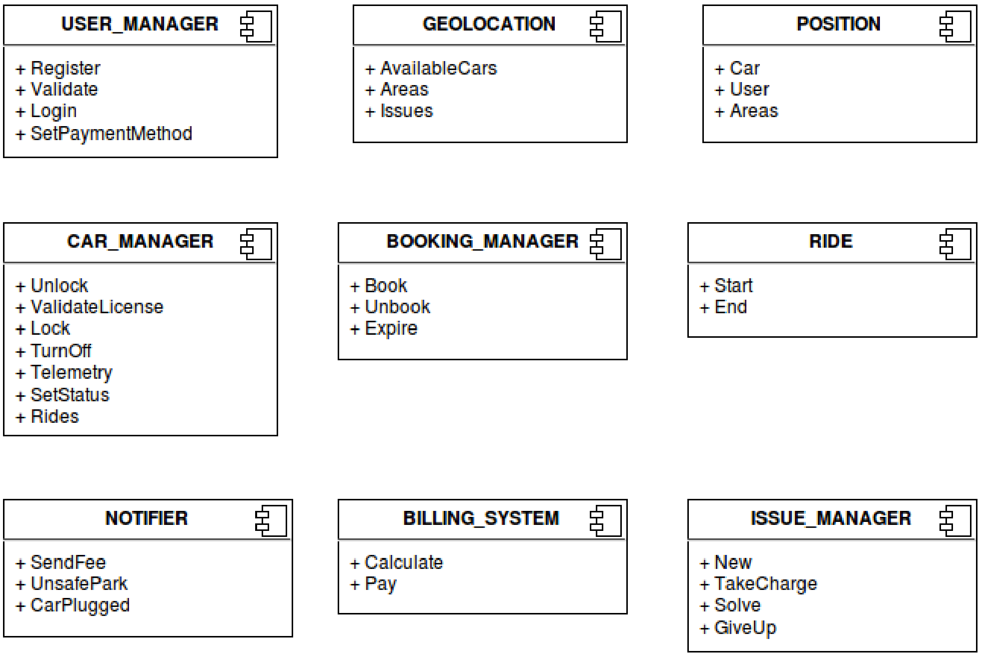

Figure 2: Graphical visualization of the abstract components needed.

2.4 From Abstract Components to Physical Nodes

Now that we have identify the abstract components that, together, provide the fucntionality of our project we are going to makes those components real.

In order to actually implements the components we - first - need to know where those components will be deployed and the form of comunication between and intra them.

In the next sub section we are going to explain what physical nodes will compose our system.

Physiscal structure

Overall there are 3 main identifiable nodes in our system:

- Main Server
- Car Onboard system
- User interface (App, web site, etc...)

We have refer to the User Interface as App throught the whole document and we will keep such terminology for sake of clarity, however, must be clear that a whole spectrum of technologies could be used to actually implement the user Interface, not only mobile application.

Also, for the scope of this analysis we often consider the customer's app and the staff's app as a single entity, called simply "app".

2.5 Communication between nodes

Now that our nodes are defined we need to understand the comunication between them.

It would have been possible to make each node comunicate with each other node, however this raise some concern about the security of the system, especially for the communication between the users and the cars. Of course a well done, and a well tested system should not raise these concerns, however real word engineering is constrained, and the authors believe that a more defensive and conservative approach is more suitable.

Following these consideration we decided that would be more maneagable to have only two communication channel, one between the main server and the user, and another between the main server and the cars.

Actual channel implementation

The communication between the main server and the user will employ a simple Client-Server approach which seems to naturally fit the domain space, it is a well know industry standard and is widely used.

On the other hand, communications between the server and the fleet is clearly more suitable for a **Publisher-Subscriber** pattern. The car has to communicate very often a lot of valuable information to the main servet, and a PubSub pattern achieves high troughput and low latency. Even if the Pub-Sub pattern may not be so widely used and know, our experience suggested that the trade-off is well worth.

2.6 Actual deploy of components into nodes

Now that we have identify the components that we need to deploy, the nodes where those components can be deployed and the communication mechanism between the nodes we are ready to actually decide where to put each component.

Most of the functionality will be invoked by the user, however, logically, the componentthey will reside on the main server.

Some component will be shared between the main server and the user or the car or both.

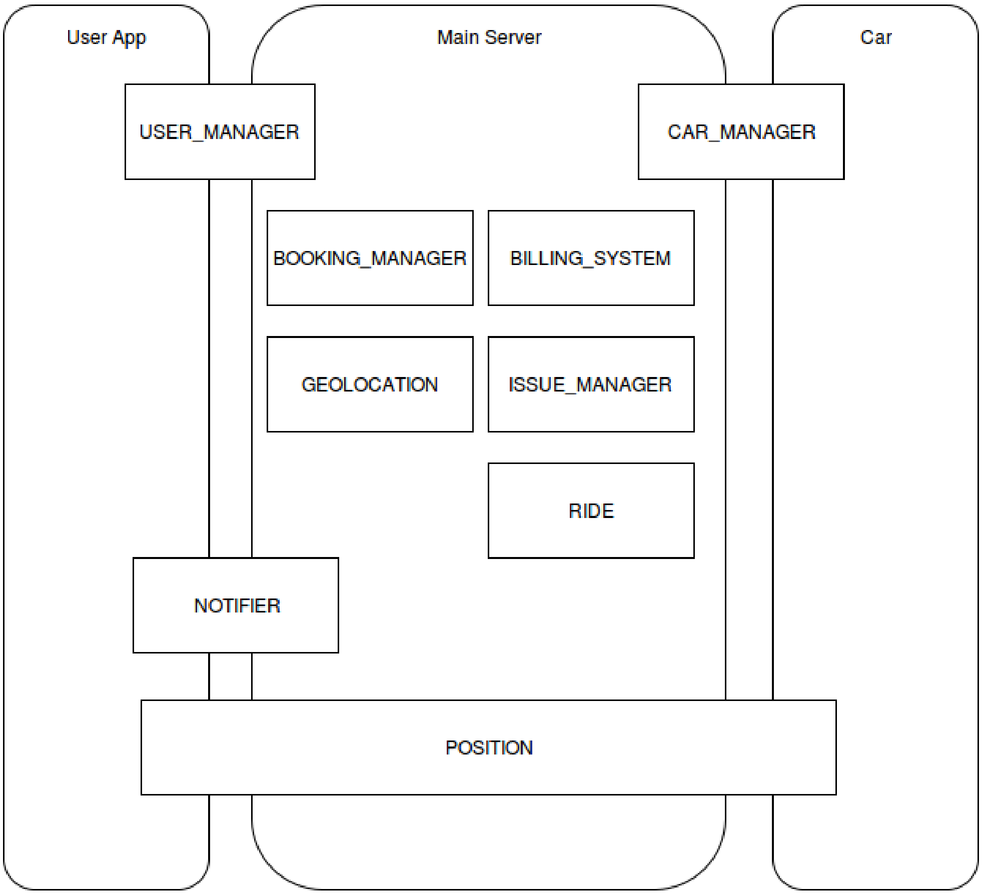

Figure 3: Graphical visualization of the abstract components needed.

REEXPLAIN WHY SOME COMPONENTES ARE SHARED

2.7 System's Structure

Asfor the RASD (section 2: Overall Description), the system is be divided into four parts:

- the **customer's app**, used by customers to access the service.
- the **main server**, a centralized backend that provides the service.
- the **cars' onboard system**, that communicates only with the centralized backend.
- the **staff's app**, used exclusively by the staff members to better organize their job.

For the scope of this analysis we often consider the customer's app and the staff's app as a single entity, called simply "app".

At a first glance, the above elements can be organized in a two-tier Client-Server architecture as follows:

- Tier 1, the main server, which handles Application Logic and Data Management.
- Tier 2, comprising mobile apps and cars, hosts the User Interface.

From now on, we design the system basing on these fundamentals elements, defining their roles and interactions.

2.8 Main Design Choices

Our choices are, first of all, focused on enforcing the architecture identified above.

Car to app communication

We decide to completely avoid all kinds of communication between the cars and the apps.

The main server is always an intermediary for app-to-car and car-to-app communications, in order for the system to have always full control on these otherwhise completely hidden interactions.

Interactions with 3rd-parties

We decided to prevent any access to third-part services to every component but the main server, mainly for security reasons.

The main server will expose the necessary API to apps and cars to allow them retrieving all the informations they need without having them communicating indipendently with any external service.

Communication protocols

More specific choices has been made on the communication protocols too.

A pure **Client-Server** communication protocol is implemented only for server-to-app and app-to-server communications, as they seem to naturally fit this model.

On the other hand, communications between the server and the fleet is clearly more suitable for a **Publisher-Subscriber** pattern. The car has to communicate very often a lot of valuable information to the main servet, and a PubSub pattern achieves high troughput and low latency.

To describe the communication approach in terms of the protocol itself, the car publishes messages about it own status and the server subscribes to those messages. The server is meant to host brokers too.

Moreover, we decided to model the most part of external services as Web services, so the server will communicate with them using a **Service-Oriented** approach.

Here we provide a high-level diagram of the system and some proposed technologies that we will discuss in the last sections of the document.

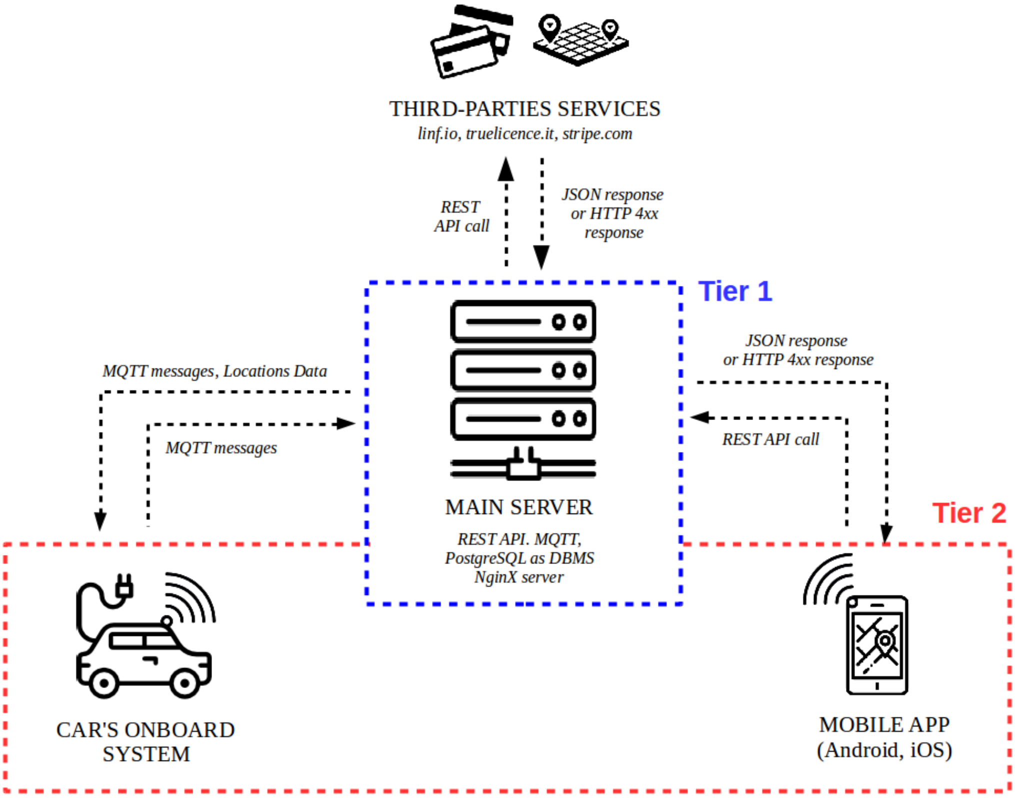
THIRD-PARTIES SERVICES
linf.io, truelicence.it, stripe.com
REST API call
JSON response or HTTP 4xx response
Tier 1
JSON response or HTTP 4xx response
REST API call
MQTT messages, Locations Data
MQTT messages
MAIN SERVER
REST API. MQTT, PostgreSQL as DBMS NginX server
Tier 2
CAR'S ONBOARD SYSTEM
MOBILE APP (Android, iOS)

Figure 4: High-level organization of the system's required elements

2.9 Component Interfaces

We now proceed defining the API that the nodes expose to each other, mapping them to the requirements they fulfill. Note that we are defining only internal interfaces: interface that components are exposing to the user are specified in the subsequent section “User Interfaces“.

In interfaces definitions, we reported only requirement's codes. For requirements definitions, see RASD section 3.2: Functional Requirements.

Server to All Apps API

List of interfaces exposed by the server to both customer's and staff's apps.

LOGIN Provided valid credentials, users are logged into the system.
Fulfills **LOG1, LOG2, LOG3** and **LOG4**

UNLOCK Logged users can unlock the car they booked.
Fulfills **UNLK1, UNLK2, UNLK3, UNLK5, SUP5, SUP6**

LOCK Logged users lock their car.
Fulfills **RIDE5, SAFE3, SAFE4**

REPORT_ISSUE Users can report issues about a car.
Fulfills **REP1**

Server to Customer's App API

List of interfaces exposed by the server only to customer's apps.

REGISTER Users provide their personal informations, including license number and billing information, and obtain an account.
Fulfills **REG1**.

VALIDATE The system validates the informations provided at registration time.
Fulfills **REG2** and **REG3**.

LOOKUP Logged users can retrieve a list of available cars according to their search settings.
Fulfills **LOOK1, LOOK2** and **LOOK3**

BOOK Logged users can reserve a car.
Fulfills **BOOK2, BOOK3, BOOK4, BOOK5**

UNBOOK Logged users can cancel a reservation they made.
Fulfills **UNBOOK2**

CALCULATE_FEE The system calculates the total fee that the user must pay when the car gets locked.
Fulfills **FEE1, FEE2, FEE3, FEE4, FEE5, FEE6**

SET_PAYMENT_METHOD Users can set their preferred paying method.
Fulfills **PAY1**

Server to Staff's App API

List of interfaces exposed by the server only to staff's apps.

RIDE The system cal tell which user is driving which car at the present time and at any moment in the past.
Fulfills **RIDE1**

FIND_ISSUE Staff operators can locate issued cars that needs their intervantion.
Fulfills **ISS1, ISS3**

TAKE_CHARGE The system allows operators to take charge of certain issues.
Fulfills **SUP1, SUP6**

SOLVE The system allows the operator to mark an issue as solved.
Fulfills **SUP3**

GIVE_UP The system allows the operator to give up over an issue.
Fulfills **SUP7**

SET_STATUS Operators can change the Exception status of issued cars.
Fulfills **SUP4**

Server to Car API

List of interfaces exposed by the server only to the cars' onboard system.

SHOW_INFORMATIONS Once the ride started, the car receives basic informations such as nearby safe parking areas and nearby charging stations.
Fulfills **RIDE4, SAFE1, SAFE2, PWRS1, PWRS2**

VALIDATE_LICENSE The system uses input information about the user's driving license to decide whether to grant the user permission to start the car's engine.
Fulfills **RIDE2, RIDE3, SUP6**

Customer's App to Server API

List of interfaces exposed by the customer's apps to the main server.

EXPIRE A booked car not unlocked after a system-defined period of time has passed is automatically unbooked and the user who booked it is fined.
Fulfills **BOOK6**

PAY The app receive payment requests.
Fulfills **PAY4**

Staff's App to Server API

We identified no interfaces exposed by the staff's apps to the main server at this stage.

Car to Server API

List of interfaces exposed by the car's onboard system to the main server.

Note that the communication protocol between the server and the car is meant to be message-based: these API thus represent messages exchanged between the server and the car.

ENGINE_STATUS The car reports the engine status (True if running, False otherwise).
Fulfills **UNSF2**

NUM_PASSENGERS The car reports how many passengers are onboard (excluding the driver).

DRIVER_PRESENT The car reports if the driver is inside the car (True) or not (False).
Fulfills **UNSF2**

PLUGGED The car reports if it is connected to a charging station (True) or not (False).
Fulfills **PWRS3**

LOCKED The car reports if it is locked (True) or unlocked (False).
Fulfills **UNSF2**

BATTERY_LEVEL The car reports its battery level (100 fully charged, 0 fully empty)

GPS_DATA The car reports its GPS coordinates as retrieved from the sensor.
Fulfills **UNSF2**

MOVING The car reports if it is moving (True) or if it stopped (False), regardless of the engine status.
Fulfills **UNSF2**

ISSUES The car reports its Exception Status (could be "No Issue").

LOCK_CAR The system forces the car to lock when left in an unsafe area.
Fulfills **UNSF3**

ENGINE_OFF The system forces the cas to switch its engine off when the car is left in an unsafe area.
Fulfills **UNSF4**

SET_EXCEPTION_STATUS The system sets the car's Exception Status to "Unsafely Parked" if it understands it is parked in an unsafe area.
Fulfills **UNSF2**

VALIDATE_SOLVE The car performs a self-check and confirms to the server its Exception status can be set back to "No Issue"

REPORTED_ISSUE The car sets its Avaliability status as "Not Available" and its Exception status as "Other issue" as the system receives a related issue report.
Fulfills **REP2**

External Services to Server API

List of interfaces exposed by external services to the main server. TODO!

Qui il diagramma delle interfacce a pagina 18 dell'esempio. Non capisco come faccia il loro a essere cosi magro...

2.10 User Interfaces

TODO!

Now we define in details which interfaces are being exposed to the final user. These are going to be better defined later, but we report them here in order to show exactly which requirements are meant to satisfy.

2.11 Component View

Given the interface we identified in the previous section, we organize such interfaces into higher level components as follows.

Components are described as **COMPONENT_NAME/Functionality_Name**, then highlighting responsibility, input and output data, and related interfaces for each function.

All the components are responsible of returning meaningful error messages in case of error that we are not going to specify in detail.

Qui un Components View simile a quello a pagina 8 dell'esempio... possibilmente con un font un po' piu' grande che nell'esempio.
Non so bene se qui o in fondo alla sezione successiva.

2.12 High-Level Architecture

Up to this point we defined the logical components of the system in terms of implemented interfaces. We now proceed deploying them on the nodes identified in the first section of the analysis in order to end up with a complete, high-level logical architecture for our system.

USER_MANAGER

Considering that all its three functions are exposed as API by the server to the apps, the component is deployed on the server entirely.

LOCATION

Considering that all its three functions are exposed as API by the server, the component is deployed on the server entirely.

POSITION

Considering that all its three functions are exposed as API by the server, the component is deployed on the server entirely. Define this better

BOOKING_MANAGER

Considering that all its three functions are exposed as API by the server, the component is deployed on the server entirely. What about EXPIRE?

CAR_MANAGER

This component requires a deeper analysis, as its interfaces are exposed by different elements. In details:

- CAR/Unlock is exposed by Server to Apps
- CAR/Rides is exposed by Server to Apps
- CAR/ValidateLicense is exposed by Server to Cars
- CAR/Lock is exposed by Cars to Server
- CAR/TurnOff is exposed by Cars to Server
- CAR/Telemetry is exposed by Cars to Server
- CAR/SetStatus is exposed by Cars to Server

Thus we create two CAR_MANAGER subcomponents:

REMOTE_CAR_MANAGER

Comprises all the functions exposed by the server:

- CAR/Unlock
- CAR/Rides
- CAR/ValidateLicense

LOCAL_CAR_MANAGER

Comprises all the functions exposed by the cars:

- CAR/Lock
- CAR/TurnOff
- CAR/Telemetry
- CAR/SetStatus

BILLING_SYSTEM

Considering that all its three functions are exposed as API by the server to the customer's apps, the component is deployed on the server entirely. Recheck PAY : is it on server or on the app?

ISSUE_MANAGER

Considering that all its three functions are exposed as API by the server to the customer's apps, the component is deployed on the server entirely.
Indeed, ISSUE/Solve involves an interface that is exposed by the car: thus we define another subcomponent, **VALIDATE_SOLVE**, to be deployed on the car to fulfill this function.

Il graficino High-level Arch a pagina 8 del DD di esempio, per quanto inutile possa essere in questo caso.

2.13 Deployement View

Il graficino del deploy a pagina 10 del DD di esempio, se capisci che cosa diavolo sia.

2.14 Runtime View

Tanti bei Sequence Diagrams :P

2.15 Selected Technologies

Having defined the interface between the several functionality provided by the components, all possible communication technologies may have been choosen. Obviously some choices are more apt than others with the respect of latency, throughput, elegance of the design and other factors; however the whole design will remain intact whichever technology we choose.

Here we simply give some reasonable suggestions for protocols and technologies that seems us a better fit for the system we modeled.

RESTful API

The main server will expose its API in the most conventional way, using the classical HTTP/TCP/IP stack. In particular we provide RESTful interfaces. Model everything as an entity provides enough capabilities to actual implement the whole system while remaining constrainted to only the basic REST verb. This helped a lot in keeping the whole API scheme simple to understand and to implement.

Apps consume the REST interface provide by the server, however to implement the communication between the server and the app we will use the long polling strategy.

MQTT

As for now, most of the communication between the server and the car will happens using the PubSub protocol, while some special messages (like, for example, the ValidateSolve function) will stay on a more basic Client-Server approach.

Between all the implementation of the PubSub protocol we chose to use MQTT for its low overhead, its QoS and because it is widely used in the industry.

Distributing the main server

what about this?

Database

what about this?

3 OLD STUFF

3.1 Interfaces

ns provided at registration time.
Fulfills **REG2** and **REG3**.

LOGIN Users can login to PowerEnJoy . double-check LOG4

- **LOGIN** Provided valid credentials, users are logged into the system and from now on they have the possibility to book cars, unlock cars, etc.
  Fulfills **LOG1, LOG2, LOG3** and **LOG4**

LOOKUP Users can find cars nearby a given position, according to their search settings. missing LOOK4, LOOK5, LOOK6

- **LOOKUP** Logged users can retrieve a list of available cars according to their search settings.
  Fulfills **LOOK1, LOOK2** and **LOOK3**

BOOK Users can book a car for a short amount of time. double-check. Missing BOOK1. EXPIRE added

- **BOOK** Logged users can reserve a car.
  Fulfills **BOOK2?, BOOK3?, BOOK4?, BOOK5?**
- **EXPIRE** A booked car not unlocked after a system-defined period of time has passed is automatically unbooked and the user who booked it is fined.
  Fulfills **BOOK6**

UNBOOK Users can decide to cancel a booking made before the expiration. Missing UNBOOK1

- **UNBOOK** Logged users can cancel a reservation they made.
  Fulfills **UNBOOK2**

UNLOCK Users can unlock the car they booked. Missing UNLK4

- **UNLOCK** Logged users can unlock the car they booked.
  Fulfills **UNLK1, UNLK2, UNLK5**
- **POSITION** The system must be able to locate the user.
- **UNLOCK_CAR** The system can unlock a car.
  Fulfills **UNLK3**

RIDE Users can drive to their destination. change RIDE def. Add PARK

- **RIDE** The system knows which user is driving which car at the present time and at any moment in the past.
  Fulfills **RIDE1**

- **READ_LICENSE** The system acquires information about the user's driving license and decides whether to let the user start the engine or not.
  Fulfills **RIDE2, RIDE3**
- **SHOW_INFORMATIONS** Once the ride started, the system shows to users basic informations such as nearby safe parking areas and nearby charging stations.
  Fulfills **RIDE4, SAFE1, SAFE2, PWRS1, PWRS2**
- **PARK** Users can lock the car once they finished the ride and exited from the car.
  Fulfills **RIDE5, SAFE3, SAFE4**

SAFE_AREAS Users can locate safe parking areas. Overlaps with RIDE!

- **SHOW_INFORMATIONS** Once the ride started, the system shows to users basic informations such as nearby safe parking areas and nearby charging stations.
  Fulfills **RIDE4, SAFE1, SAFE2, PWRS1, PWRS2**
- **PARK** Users can lock the car once they finished the ride and exited from the car.
  Fulfills **RIDE5, SAFE3, SAFE4**

UNSAFE_PARKING The system reacts to an unsafe parking. Missing UNSF1, UNSF2

- **TURN_OFF** The system turns off a car left in an unsafe area.
  Fulfills **UNSF3**
- **LOCK_CAR** The system locks a car left parked in an unsafe area.
  Fulfills **UNSF4**

POWER_STATION Users can locate and use charging stations correctly. Missing PWRS4

- **SHOW_INFORMATIONS** Once the ride started, the system shows to users basic informations such as nearby safe parking areas and nearby charging stations.
  Fulfills **RIDE4, SAFE1, SAFE2, PWRS1, PWRS2**
- **CAR_PLUGGED** The system detects whenever a car is plugged to a power station.
  Fulfills **PWRS3**

CHARGE At the end of the ride, the user is charged a fee.

- **CALCULATE_FEE** The system calculates the total fee that the user must pay.
  Fulfills **FEE1, FEE2, FEE3, FEE4, FEE5, FEE6**
- **SEND_FEE** The system communicates to the user the total cost of the ride when the car gets locked.
  Fulfills **FEE7**

PAYMENT Users can pay bills through the app. Missing PAY2, PAY3, PAY4

- **PAY** The system asks users to pay ride fares. No matching req. found .-.

SET_PAYMENT_METHOD Users can set their preferred paying method.
Fulfills **PAY1**

FIND_ISSUES The staff can locate cars that need their intervention. Missing ISS2, ISS4

FIND_ISSUE Staff operators can locate issued cars that needs their intervantion.
Fulfills **ISS1, ISS3**

REPORT_ISSUE Users can report issues about a car. No matching req. found .-.

SUPPORT The staff can identify and solve car's issues. Add CONFIRM_AVAILABILITY

TAKE_CHARGE The system allows operator to take charge of certain issues.
Fulfills **SUP1**

SET_STATUS Operators can change the statuses of issued cars.
Fulfills **SUP2, SUP4**

CONFIRM_AVAILABILITY The system checks if the issues marked as Solved are really related to cars that shows no issue.
Fulfills **SUP3**

3.2 Components

Given the interface we identified in the previous section, we organize such interfaces into higher level components as follows.

Note that all the components are responsible of returning meaningful error messages in case of error.

USER_MANAGER

Responsability Manages the users.

USER/Register

Responsability Registers a new user into the system.

Input Information from the user such as:

- Name
- Lastname
- Password
- Email
- License ID
- Credit card informations: credit card number, control code, expiry date, owner, etc.

Output The ID of the newly created user.

USER/Login

Responsability Allows users to log into the system.

Input Email (considered a unique user ID) and password.

Output A session key, meaning that the user is logged into the system.

LOCATION

Responsability Locates elements, points and areas of interest around a specific coordinate. "Search" service for elements of interest.

LOCATION/AvailableCar

Responsability Retrives the position of available cars.

Input Search parameters such as:

- Geographical coordinates of the center of the search range (latitude and longitude as provided by GPS sensors)
- Maximum walking distance from the specified position
- Other search settings, like minimum battery level, etc.

Output A set of available cars matching the search parameters.

LOCATION/Areas

Responsability Retrives the position of areas of interest, such as power stations and safe parking areas.

Input Geographical coordinates of the center of the search range (latitude and longitude as provided by GPS sensors) and a search radius.

Output A set of areas of interest inside the circle of radius provided centered on the coordinates provided.

LOCATION/IssuesCar

Responsability Retrives the position of cars with some issues.

Input Search parameters such as:

- Geographical coordinates of the center of the search range (latitude and longitude as provided by GPS sensors)
- Radius of the search
- Issue type, Exeption status, and other similar search settings.

Output A set of cars with issues matching the search parameters inside the circle of radius provided centered on the coordinates provided.

POSITION

Responsability Locatse elements of interest given their ID. "Lookup" service for elements of interest.

POSITION/Car

Responsability Retrieves the position of a specific car.

Input The ID of the car.

Output The coordinates of the car.

POSITION/User

Responsability Retrieves the position of an user.

Input The ID of the user.

Output The coordinates of the user.

POSITION/Areas

Responsability Retrieves the position of an area of interest.

Input The ID of the area.

Output The coordinates of the area as a set of boundary points.

BOOKING_MANAGER

Responsability Manages reservations.

BOOKING/Book

Responsability Books one available car.

Input The ID of the car and the ID of the user.

Output The car is booked and the ID of the reservation is provided.

BOOKING/Unbook

Responsability Removes a reservation.

Input The ID of the user and the ID of the reservation.

Output The reservation is cancelled.

CAR_MANAGER

Responsability Manages the iteractions between users and cars.

CAR/Unlock

Responsability Unlocks the car.

Input The ID of the car and the ID of the user asking to unlock.

Output The car is unlocked.

CAR/Lock

Responsability Locks the car.

Input The ID of the car.

Output The car is locked.

CAR/TurnOff

Responsability Turns off the engine of a car.

Input The ID of the car.

Output The car is turned off.

CAR/Telemetry

Responsability Retrieves real-time, updated informations about a car.

Input The ID of the car.

Output All the latest informations available about the car.

BILLING_SYSTEM

Responsability Manages all the fees.

BILL/Calculate

Responsability Calculates the amount of a riding fee.

Input The ID of the ride.

Output The final fee, including eventual discounts or overprices.

BILL/Pay

Responsability Requires user to pay a specific bill.

Input The ID of the user and the ID of the ride the bill refers to.

Output The request of payment.

BILL/Collect

Responsability Complete the money transaction.

Input Credit card informations of users and the ID of the bill they have to pay.

Output The fee is paid.

ISSUE_MANAGER

Responsability Manages car's issues.

ISSUE/New

Responsability Rise a new issue.

Input ID of the car, ID of the user raising the issue, a title and a description of the issue.

Output The ID of the reported issue.

ISSUE/Modify

Responsability Allows operators to modify the statuses of some issues. The operator may have fixed the issue, may decide that he is not capable to fix the issues or it may decide that the issues is **un-fixable**.

Input The ID of the issue, the ID of the operator, the affected status and the new status value.

Output The status of a issue is updated.

ISSUE/TakeCare

Responsability Allows operators to take charge of a particular issue.

Input The ID of the issue, the ID of the operator.

Output The operator is now responsable for the issue.

3.3 Logical Deploying

USER_MANAGER Logically deployed on the main server, while its API are invoked only by the apps.

LOCATION Logically deployed on the main server. Its API are invoked by customer's apps (LOCATION/AvailableCar), by the fleet (LOCATION/Areas) and by the staff's apps (LOCATION/IssuesCar).

BOOKING_MANAGER Logically deployed on the main server, while its API are invoked by the customer's apps.

CAR_MANAGER Logically deployed in both the main server and the fleet. Non dovremmo definire due componenti diversi, quello dell'auto e quello del server?

CAR/Unlock , invoked by the apps, involves both the main server, that guarantees that the request is legitimate, and the selected car, that actually unlocks the door.

CAR/Lock can be invoked by apps on directly by the server (in case of unsafe parking) and actuated by the cars.

CAR/TurnOff is invoked by the server and actuated by the cars.

CAR/Telemetry is invoked by the server and actuated by the cars.

POSITION Logically deployed partly on cars, partly on the users app and partly on a 3rd-part service.

POSITION/Car Deployed on the car itself and invoked, by the server on behalf of the customer's apps.

POSITION/User Deployed on the user application and invoked by the main server and, indirectly by the user itself (when the user ask to unlock a car he must be nearby the car itself).

POSITION/Areas Deployed on a 3rd-part service and invoked by the server on behalf of the cars.

BILLING_SYSTEM Logically deployed partly on the main server, partly on the users' app and partly on a 3rd-part system.

BILL/Calculate Deployed and invoked by the server.

BILL/Pay Invoked by the server but executed on the users application

BILL/Collect is invoked on the server but actually executed in a 3rd part system.

ISSUE_MANAGER This manager is deployed in the main server and it functionality are invoked by the apps, both customer's apps to report issues and staff's app to find issued cars.

3.4 Deploy

At this point we have understood, logically, where each component should be deployed and what part of the system invoke what functionality.

Now we are going to physically deploy all the functionality in the correct part of the system and we are going to define a communication mechanism between those functionality.

The interface between the several functionality provided by the components are already defined, from this point on we are going to pick a particular technology only for the sake of simplicity, all the possible communication technology may have been choosen. Obviously some choices are more apt than others with the respect of latency, throughput, elegance of the design and other factors; however the whole design will remain intact whichever technology we choose.

3.5 Selected Patterns and Technologies

The reason for the technology choice we made are expressed in this section.

The main server will expose its API in the most conventional possible way, using the classical HTTP/TCP/IP stack. In particular we focus ourselves in provide RESTfull interfaces. Model everything as an entity will provide enough capabilities to actuall implement the whole system while remaining constrainted to only the basic REST verb will help in keeping the whole API simple.

The users app will consume the REST interface provide by the server, however to implement the communication between the server and the app we will use the long polling strategy.

We believe that the car will need to communicate very often a lot of valuable information to the main server and in order to achieve high troughput and low latency we believe that the PubSub protocol is the most apt. Moreover, the PubSub protocol is pretty natural in this scenario, having the car publish messages about it own status and having the the server subscribe to those messages. Also most of the communication between the server and the car will happens using the PubSub protocol. Between all the implementation of the PubSub protocol we chose to use MQTT for its low overhead, its QoS and because it is widely used in the industry.

3.6 The rest

The high lever architecture of the system is made up of three main elements:

- The main server
- The mobile apps (user apps and staff apps alike)
- The car's onboard system

On top of these elements, there is a number of external services the servers interacts with in order to provide functionalities to the apps or to the cars' system.

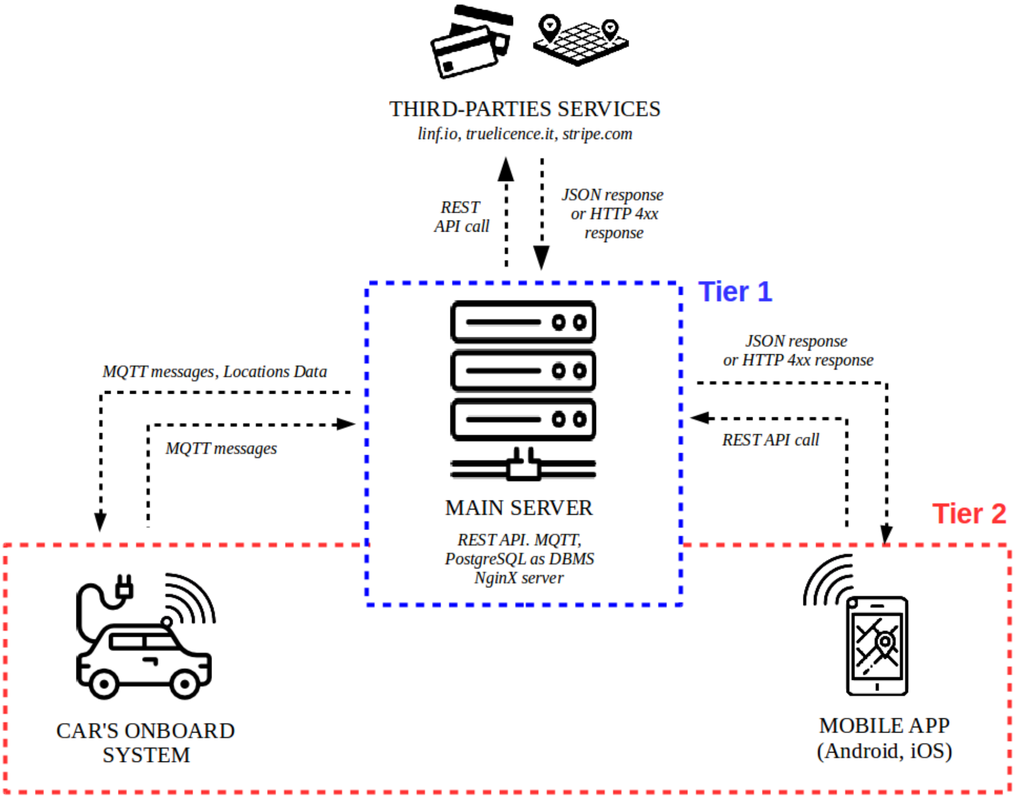
THIRD-PARTIES SERVICES
linf.io, truelicence.it, stripe.com
REST API call
JSON response or HTTP 4xx response
Tier 1
JSON response or HTTP 4xx response
REST API call
MQTT messages, Locations Data
MQTT messages
MAIN SERVER
REST API. MQTT, PostgreSQL as DBMS NginX server
Tier 2
CAR'S ONBOARD SYSTEM
MOBILE APP (Android, iOS)

Figure 5: High level overview of the architecture

At this stage, the system's architecture is clearly two-tier:

- Tier 1, the main server, handles the application logic and data management.
- Tier 2, comprising mobile apps and cars, hosts the User Interface.

The system blends three different interaction models for each of the three pair of components interacting. In detail, we designed:

- a pure **Client Server** approach when the main server interacts with the apps (customer's apps and staff's apps alike)
- a **Service Oriented** communication model between the main server interacting with external services like linf.io, truelicence.it, stripe.com etc.
- a **Publisher Subscriber** model between the main server and the cars' systems.

In subsequent sections we will provide more details about these components.

3.7 Component View

3.8 Deployement View

3.9 Runtime View

[Includes sequence diagrams to show how components interact to accomplish specific use cases]

3.10 Component Interfaces

3.11 Architectural Styles and Patterns

[Explain patterns used above]

3.12 Other Design Decisions

4 Algorithm Design

[Definition of critical sections of code]

5 User Interface Design

5.1 Mockups

Mockups have already been included in the RASD (section 3.3: Non Functional Requirements)

5.2 UX Diagrams

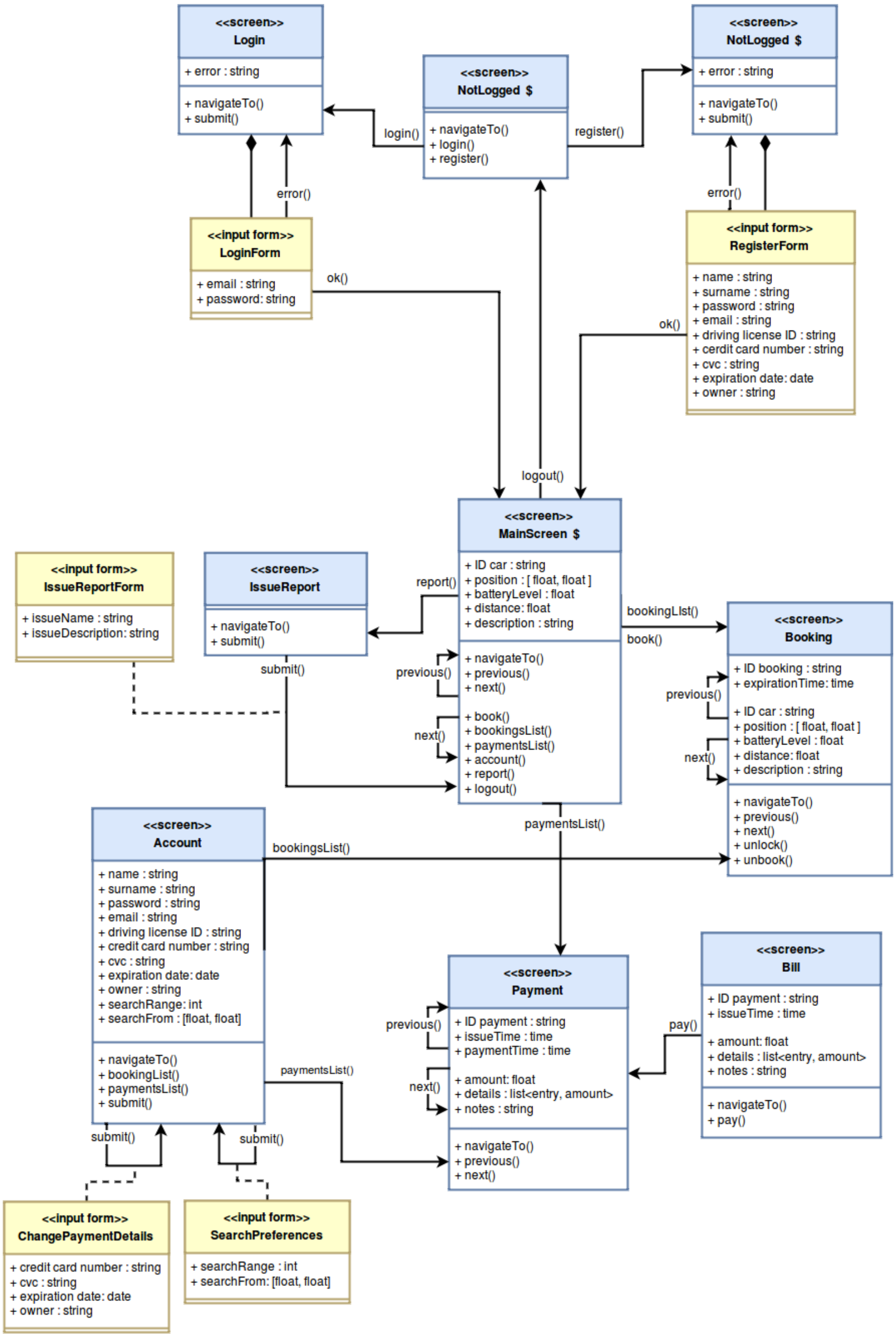

Figure 6: UX Diagram of the interface of customer's application

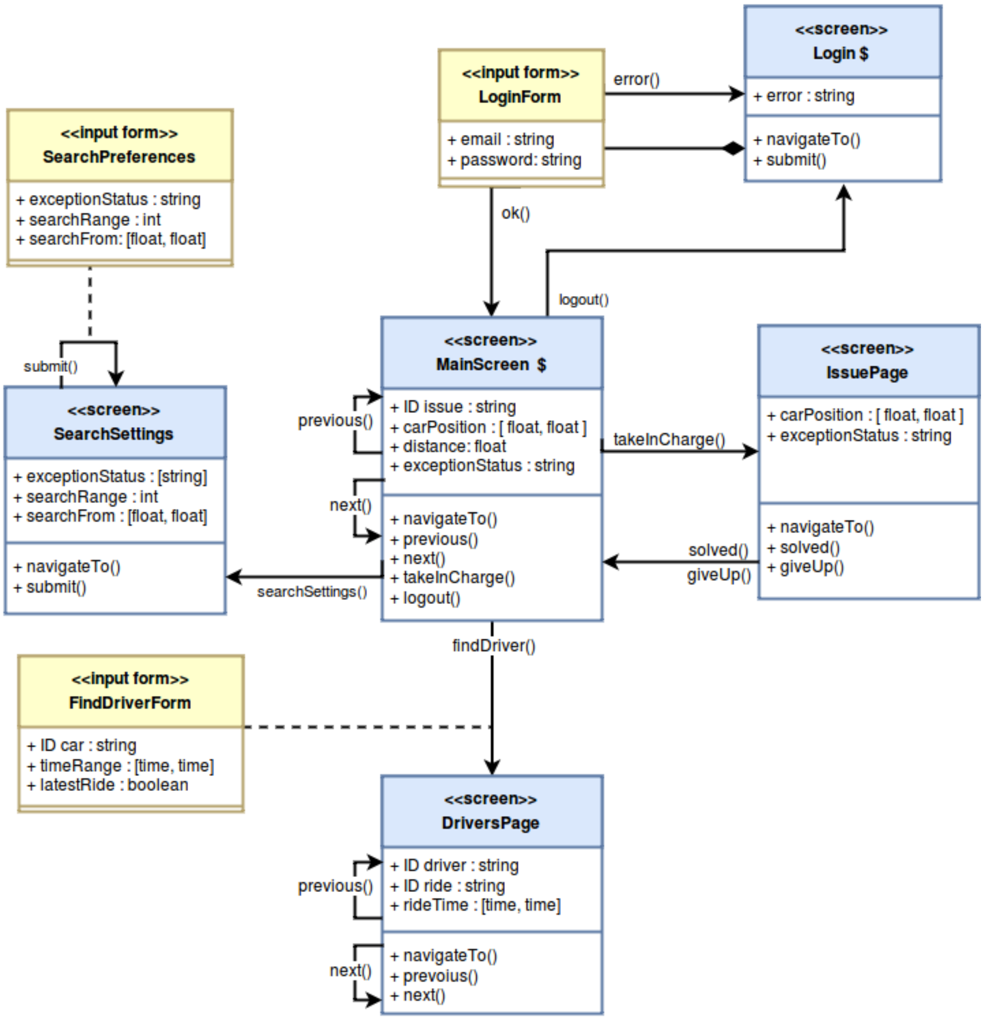

Figure 7: UX Diagram of the interface of staff's application

5.3 BCE Diagrams

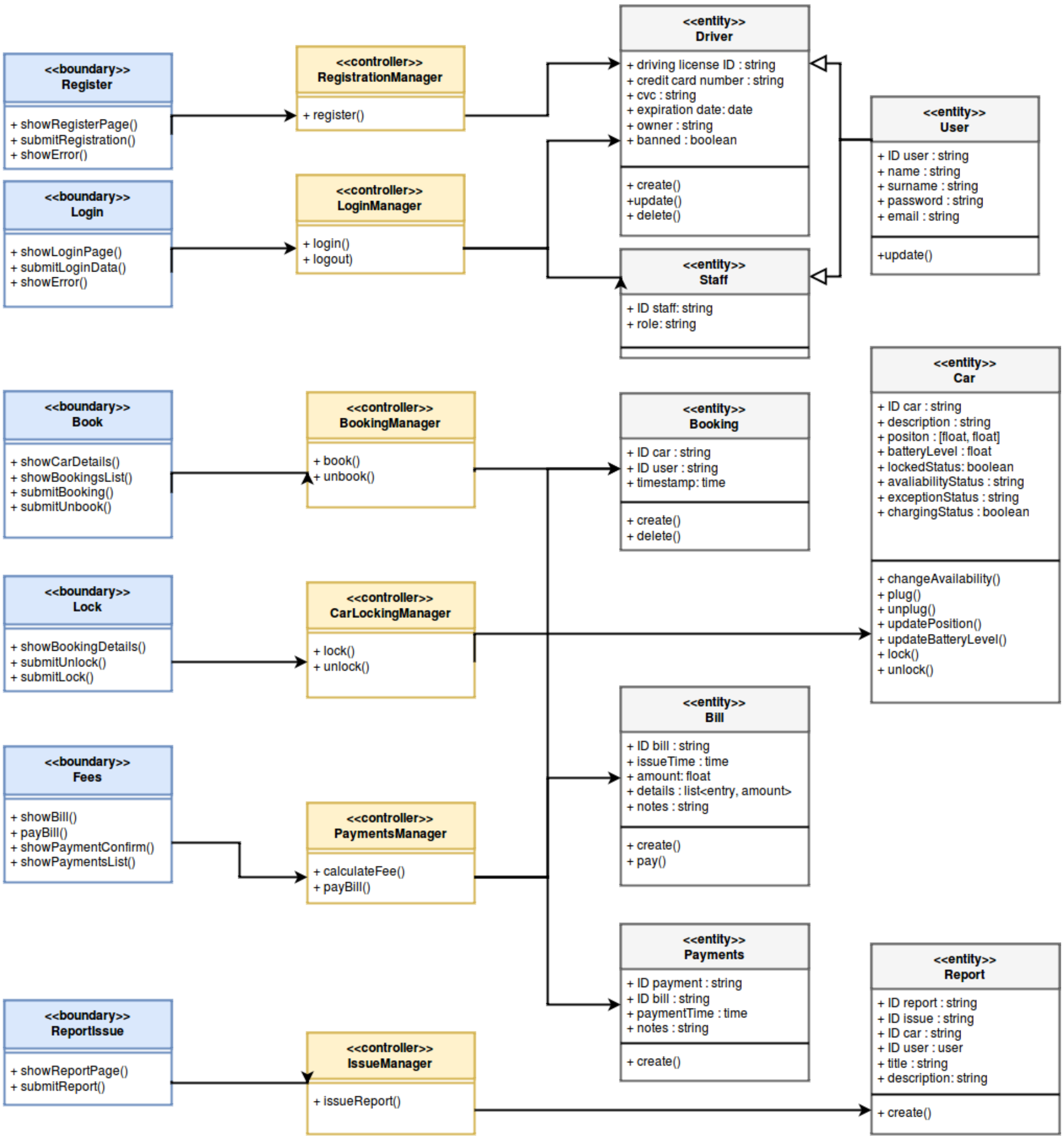

Figure 8: BCE Diagram of the interface of customer's application

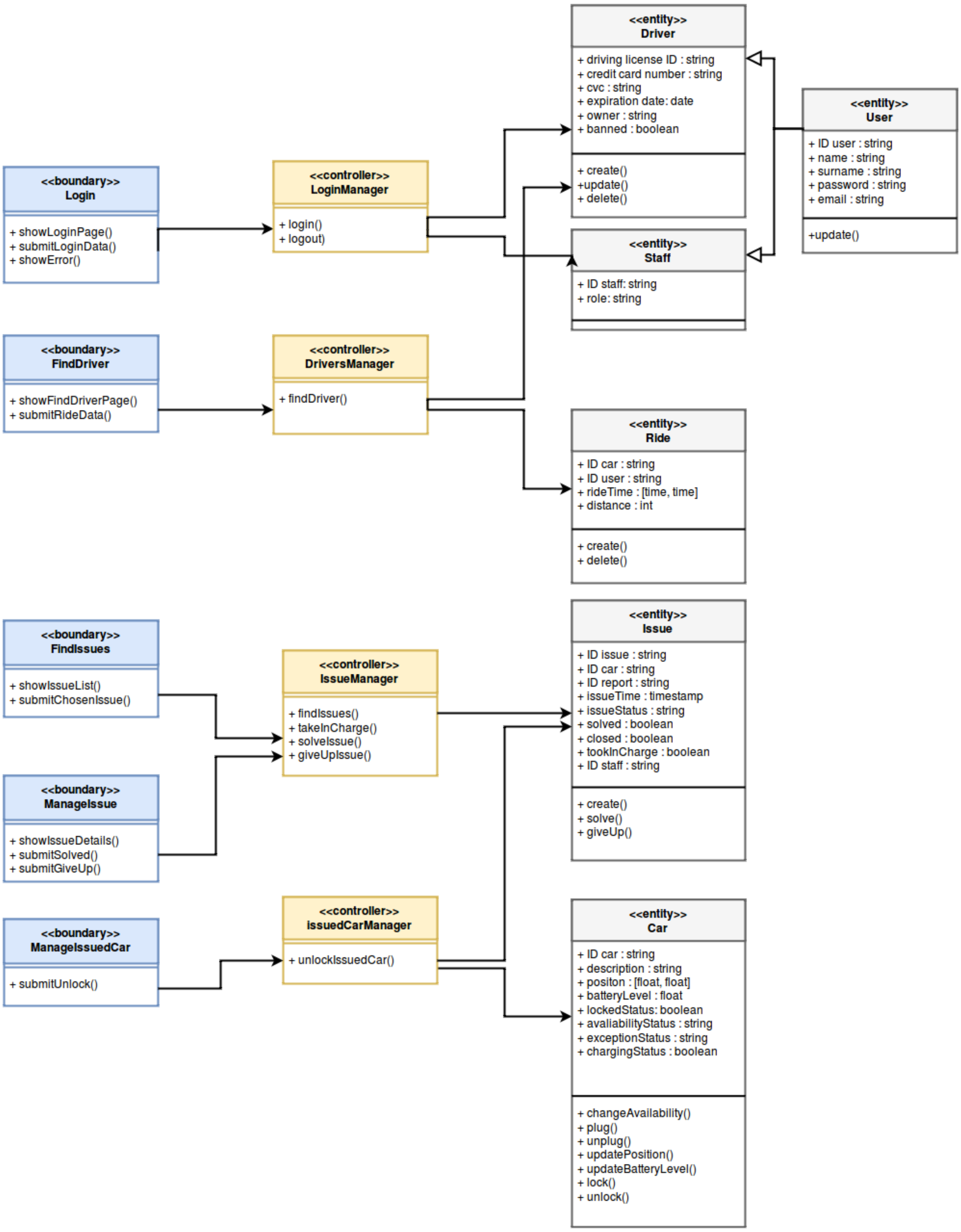

Figure 9: BCE Diagram of the interface of staff's application

6 Requirements Traceability

Lo stesso lavoro fatto nel definire le interfacce, ma a rovescio. Non piu' Interfaccia -¿ Req soddisfatti, ma Req-¿ interfaccia che lo soddisfa.

Considering we designed the system using a bottom-up approach, designed components maps in a straightforward way to the goals specified in the RASD. Hoever we provide an explicit mapping of the two.

REGISTRATION Users can register to PowerEnJoy .

- Server: RegistrationController
- Customer's App: RegistrationView (?)

LOGIN Users can login to PowerEnJoy .

- Server: LoginController
- Customer's App: LoginView (?)
- Staff's App: LoginView (?)

LOOKUP Users can find cars nearby a given position, it could be its position or a point in the map.

- Server: CarsLocation
- Customer's App: CarsView (?)

BOOK Users can book a car for a short amount of time.

- Server: BookingController
- Customer's App: BookingView (?)

UNLOCK When users are in proximity of the car they booked, the system can unlock it.

- Server: CarLockController
- Car: LockController
- Customer's App: CarNearby (———————— SEE ISSUE #14)

RIDE Users can drive to their destination.

- Server: AuthDriver
- Car: LicenceScanner, EngineController, MQTT publishers, CarGUI

SAFE_AREAS Users can locate safe parking areas.

- Server: AreasLocation
- External Services: linf.io

- Car: MQTT publishers, CarGUI

UNSAFE_PARKING The system must react to an unsafe parking.

- Server: UnsafeParkingController, IssuesController (?)
- Car: EngineController, LockController (MQTT publishers too?)

POWER_STATIONS Users can locate charging stations.

- Server: AreasLocation
- External Services: linf.io
- Car: CarGUI

CHARGE At the end of the ride, users are charged a fee.

- Server: BillingController
- Customer's App: PaymentDetailsView

PAYMENTS Users can pay bills through the app.

- Server: PaymentController
- External Services: stripe.com
- Customer's App: PaymentDetailsView

FIND_ISSUES The staff can locate cars that need their intervention.

- Server: IssuesLocation
- Staff's App: IssuesView (?)

SUPPORT The staff can identify and solve car's issues.

- Server: IssuesController
- Staff's App: IssueDetailsView (?)

FINES The system can provide enough details for the staff to manage correctly the fines they receive from local authorities.

- Server: FindDriverController (?)
- Staff's App: FindDriverView (?)

7 Conclusions

7.1 Tools used

During the development of this document we used the following tools:

- **Github** to version control the project
- **LaTeX** on TeXworks to redact this document
- **www.draw.io** to draw UML graphs
- **Gimp v.2.8** to mockup the application
- **LibreOffice Draw** to draw the system's overview at section 2.1

7.2 Hours of work

- SZ: 1h on 30/11
- SM: 5h on 2/12
- SZ: 5h on 2/12 SZ: 3h of work on 4/12 (RASD review)
- SZ: 3h on 5/12
- SZ: 4h on 6/12
- SZ: 7h on 7/12