Design Document

Simone Mosciatti & Sara Zanzottera

December 7, 2016

Contents

1 Introduction **3**
1.1 Purpose . . . 3
1.2 Scope . . . 3
1.3 Definitions, Acronyms, Abbreviations . . . 3
1.4 Document Structure . . . 5
1.5 Reference Documents . . . 5

2 Architectural Design **6**
2.1 Design Process Description . . . 6
2.2 Interfaces . . . 6
2.3 Components . . . 9
2.4 Deploying . . . 12
2.5 Logical Deploying . . . 13
2.6 Deploy . . . 13
Reason Technology Choice . . . 14
2.7 The rest . . . 14
2.8 Component View . . . 16
2.9 Deployement View . . . 16
2.10 Runtime View . . . 16
2.11 Component Interfaces . . . 16
2.12 Architectural Styles and Patterns . . . 16
2.13 Other Design Decisions . . . 16

3 Algorithm Design **17**

4 User Interface Design **18**
4.1 Mockups . . . 18
4.2 UX Diagrams . . . 19
4.3 BCE Diagrams . . . 21

5 Requirements Traceability **23**

6 Conclusions **25**
6.1 Tools used . . . 25
6.2 Hours of work . . . 25

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 overprices 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 the 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 started from the interface between the world and the machine. Given our goals we identified what interfaces the machine should provide in order to accomplish such goals.

Once the interfaces have been identified, we proceeded organizing those interfaces in components, caring at respecting the Single Responsability principle in order to provide highly decoupled components.

Once the components have been defined, we proceeded with their deployement on the logical elements of our system, and then on physical nodes building up the proposed architecture.

Once the components have been logically deployed on different parts of the system, we started reasoning about which technologies to use in order to actually implement them.

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

2.2 Interfaces

Now we enumerate which interfaces should be defined for the system to accomplish the required goals.

We reported goals definition for clarity's sake, but only requirement's codes: for requirements definitions, see RASD section 3.2: Functional Requirements.

REGISTRATION Users can register to PowerEnJoy . missing REG4, REG5

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**.

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**

2.3 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.

BOOK_MANAGER

Responsability Manages reservations.

BOOK/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.

BOOK/Unbook

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

CAR

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** Calculatesthe 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 Lets operators 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 the functionality let the operator take charge of a particular issues.

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

Output The operator is now responsable for the issue.

2.4 Deploying

Now that we have understand and named our componet we can start logically deploying them into our architecture.

As we have already anticipated in the RASD we are going to adopt a Client-Server between our main services and the users.

Then we have identify that is extremely practical to use a Pub-Sub mechanism for the communication between our fleet and the main server.

Moreover we decide to completely avoid all the digital communication between the cars and the final user.

The system is so divided into three parts, the fleet, composed by cars, the user applications and the main server.

Clearly most functionality are invoke in one part of the system and execute in another part of the system following standard comunication mechanism.

2.5 Logical Deploying

USER_MANAGER The user manager is logically deployed in the main server, while its functionality are invoke only by the users.

LOCATION The location service is logically deployed in the main server, its functionality are invoked by the users (LOCATION/AvailableCar), by the fleet (LOCATION/Areas) and by the staff (LOCATION/IssuesCar).

BOOK_MANAGER The book manager is deployed on the main server while its functionality are invoked by the users.

CAR This component is logically deployed in both the main server and the fleet. The functionality CAR/Unlock, invoked by the users, involve both the main server, that guarantee that the request is legitimate, and the cars, that actuall unlock the door. The functionalities CAR/Lock and CAR/TurnOff are both invoked by the server and actuatted by the cars. Finally the functionality CAR/Telemetry is logically invoked by the server and actuatted by the cars.

POSITION This component is deployed on the cars, on the users app and on 3rd part system. POSITION/Car is deployed on the car itself and invoked, indirectly, by the user. POSITION/User is 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 is deployed on 3rd part system and on the main server, its functionality are invoked by the car.

BILLING_SYSTEM The functionality of this component are logically deployed on the main server, the users' app and in a 3rd part system. BILL/Calculate is deployed and invoked by the server. BILL/Pay is invoked by the server but executed on the users application; while BILL/Collect is invoked in 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 staff and by the users.

2.6 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.

Reason Technology Choice

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.

2.7 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.

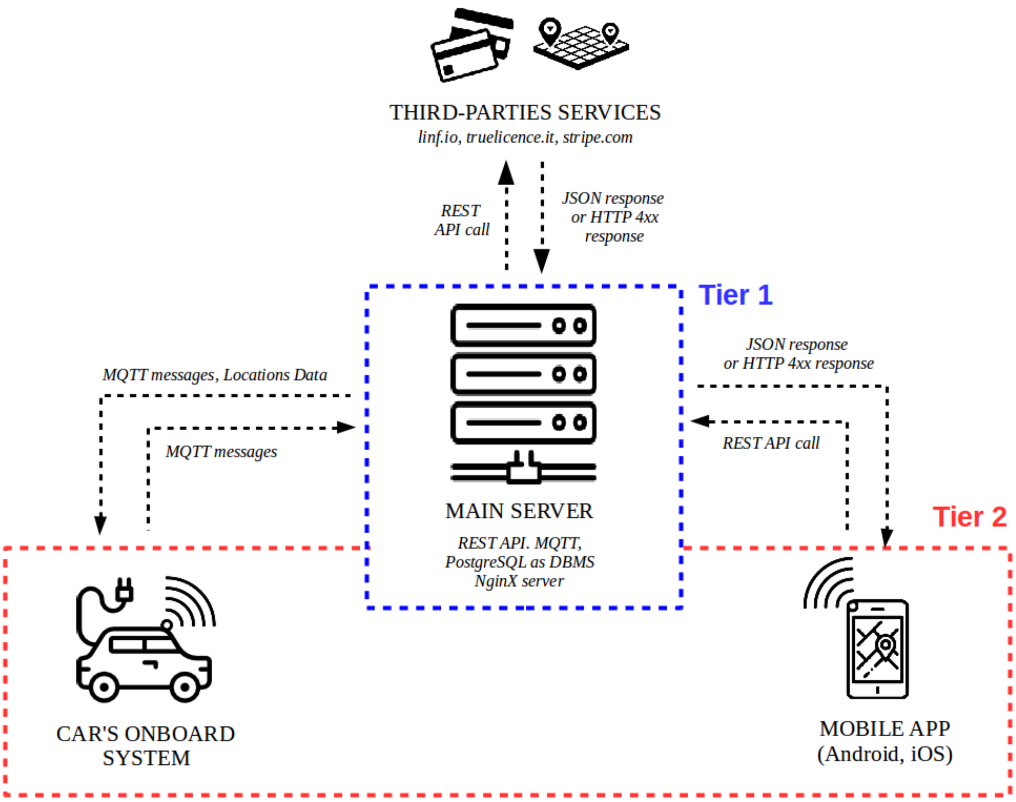
THIRD-PARTIES SERVICES
linf.io, truelicence.it, stripe.com
REST API call
JSON response or HTTP 4xx response
Tier 1
JSON response or HTTP 4xx response
MQTT messages, Locations Data
MQTT messages
REST API call
MAIN SERVER
REST API. MQTT, PostgreSQL as DBMS NginX server
Tier 2
CAR'S ONBOARD SYSTEM
MOBILE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

Figure 1: 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.

2.8 Component View

2.9 Deployement View

2.10 Runtime View

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

2.11 Component Interfaces

2.12 Architectural Styles and Patterns

[Explain patterns used above]

2.13 Other Design Decisions

3 Algorithm Design

[Definition of critical sections of code]

4 User Interface Design

4.1 Mockups

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

4.2 UX Diagrams

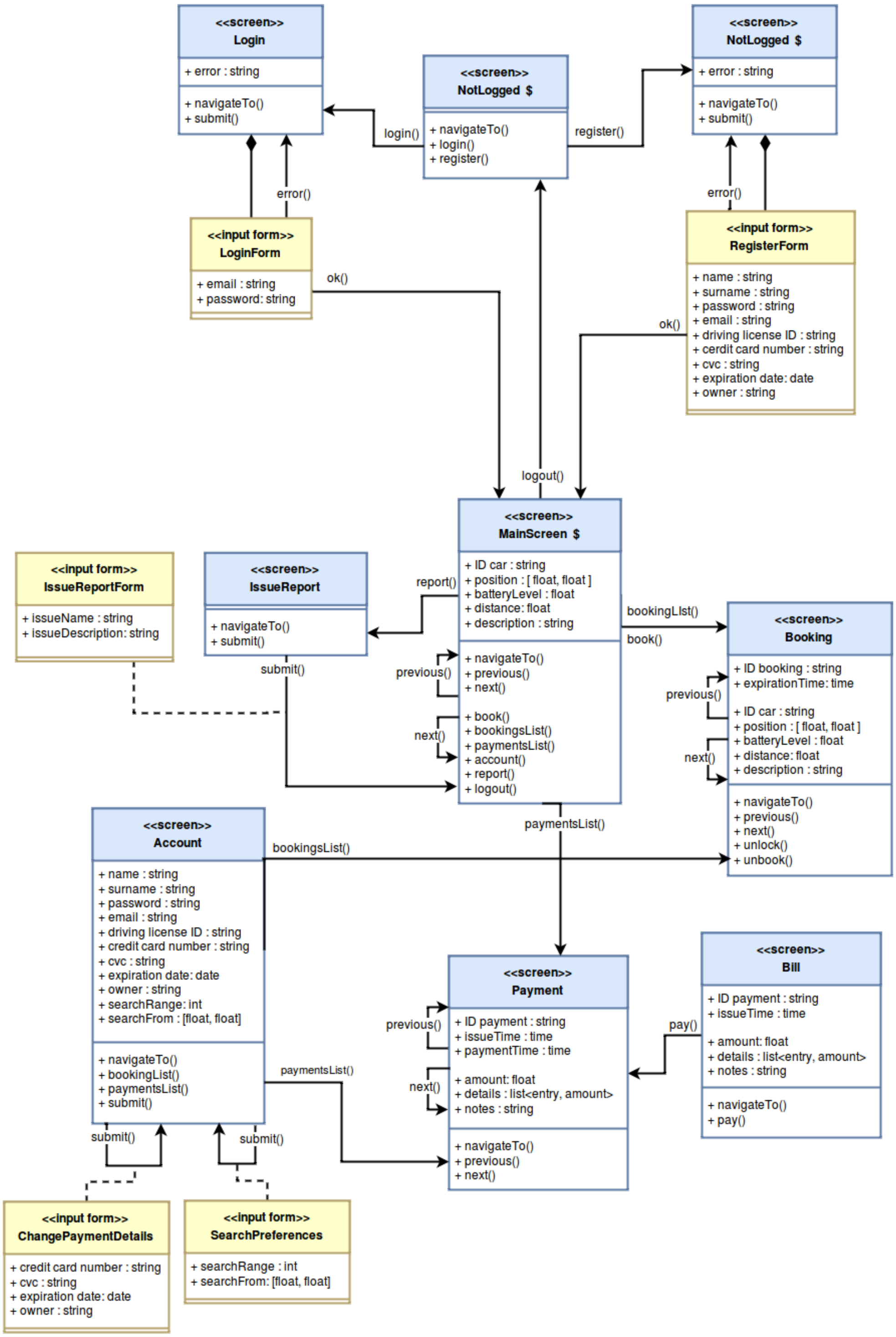

Figure 2: UX Diagram of the interface of customer's application

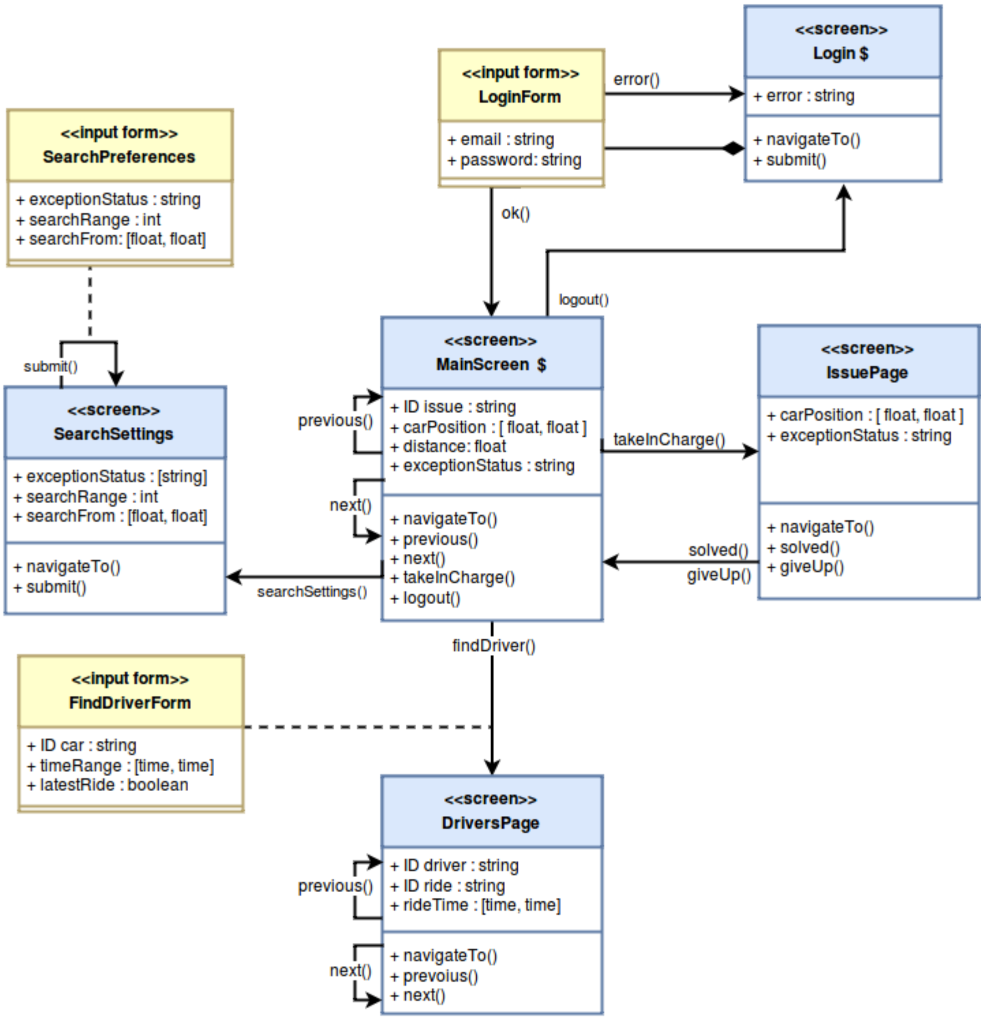

Figure 3: UX Diagram of the interface of staff's application

4.3 BCE Diagrams

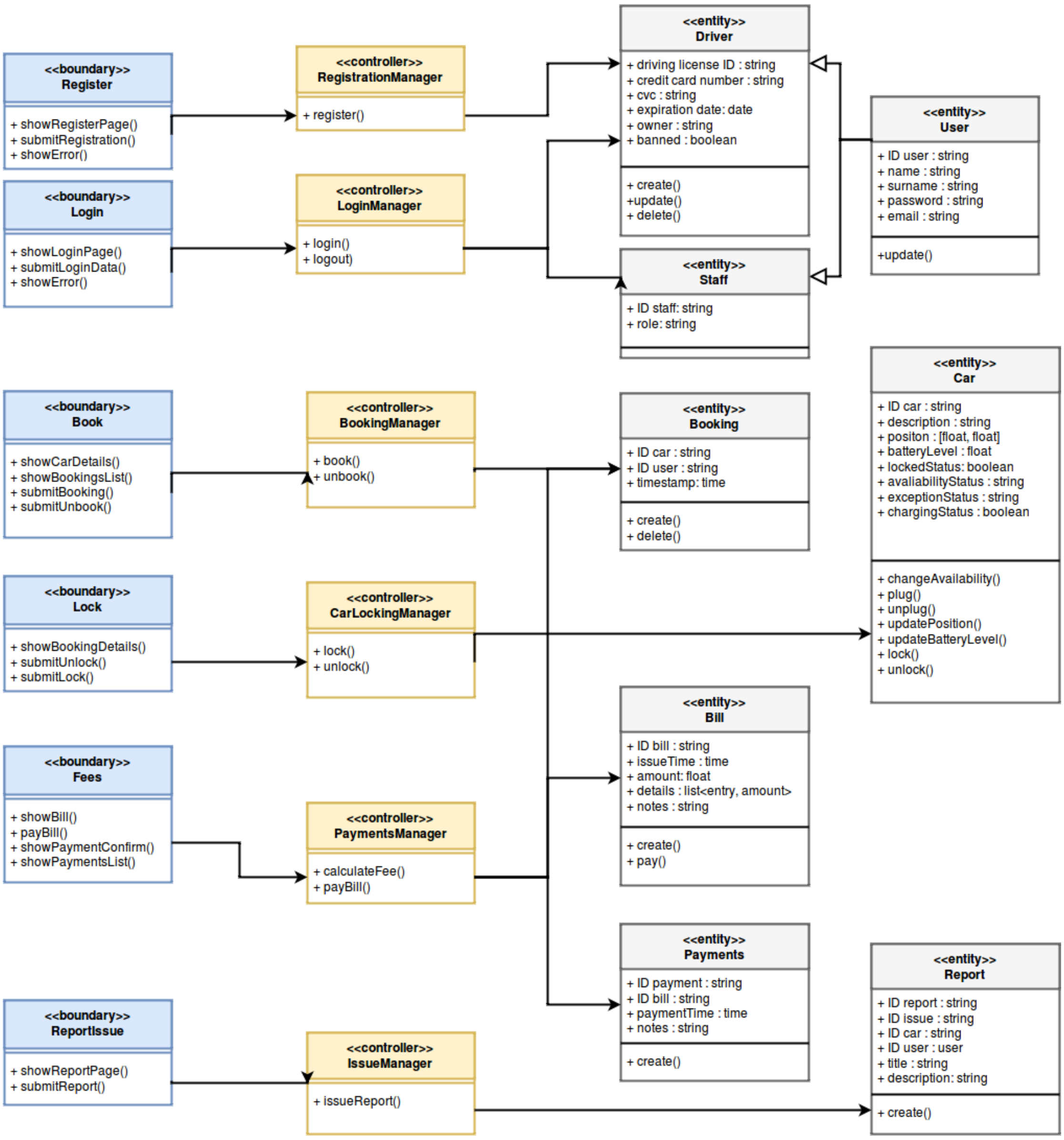

Figure 4: BCE Diagram of the interface of customer's application

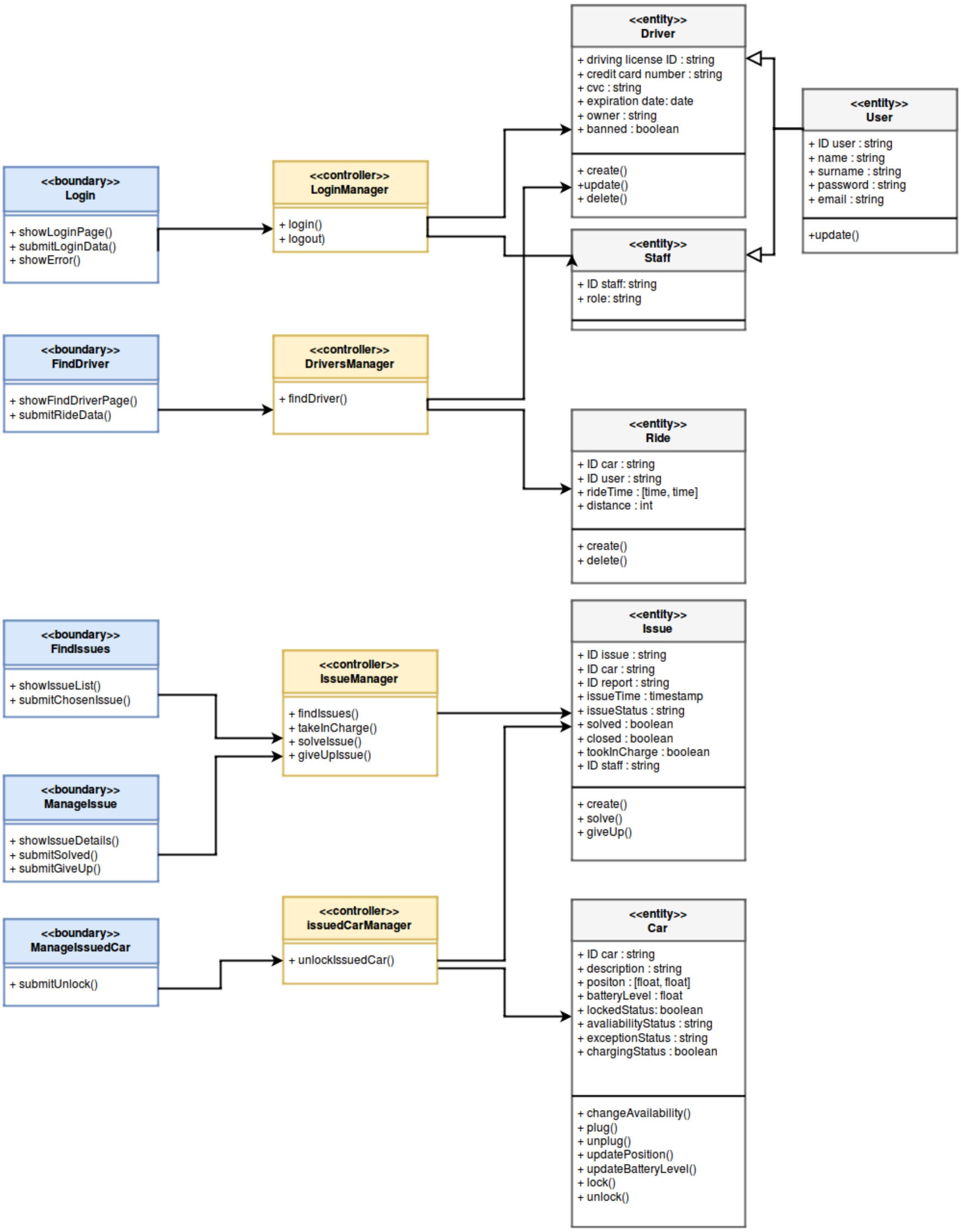

Figure 5: BCE Diagram of the interface of staff's application

5 Requirements Traceability

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 (?)

6 Conclusions

6.1 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

6.2 Hours of work

- SZ: 1h on 30/11
- SM: 5h on 2/12
- SZ: 5h on 2/12
- SZ: 3h on 5/12