Politecnico di Milano, A.A. 2016/2017

Software Engineering 2: PowerEnJoy
Design **D**ocument

Binosi Lorenzo 876022

December 11, 2016

Contents

1 Introduction

1.1 Purpose

The Design Document has the purpose to provide a functional description of the system. It defines the design of the software architecture, its components and their interactions, provided with the used algorithms and the user interfaces design.

The document is written for project managers, developers, testers and quality assurance. It can be used for a structural overview to help maintenance and further development.

1.2 Scope

PowerEnJoy is a large-scale car-sharing service. It requires to perform excellently, to be secure, easily modifiable and as available as possible. Minding these necessities, in the following chapters will be shown how the system is structured and and which were the reasons that led to such decisions.

1.3 Definitions, Acronyms, Abbreviations

DD: Design Document.

RASD: Requirements Analysis and Specification Document.

System: the whole software system to be developed, comprehensive of all its parts.

REST: REpresentational State Transfer. It's an architectural style and an approach to stateless communications, used in the development of client-server systems.

GUI: Graphical User Interface.

ACID: Atomicity, Consistency, Isolation, Durability. A set of properties of database transactions.

API: Application Programming Interface.

EJB: Enterprise Java Bean.

JPA: Java Persistence API.

JSP: JavaServer Pages.

HTTP: HyperText Transfer Protocol.

HTTPS: HyperText Transfer Protocol Secure.

1.4 Reference Documents

This document refers to the project rules of the Software Engineering 2 project [1] and to the DD assignment [2].

1.5 Document Structure

This document is structured in five parts:

section 1: Introduction. This section provides general information about the DD document and the used terms.

section 2: Architectural Design. This section shows the software architecture of the system, with its components and their interactions.

section 4: Algorithm Design. This section will present and discuss in detail the algorithms designed for the system functionalities, independently from their concrete implementation.

section 3: User Interface Design. This section shows how the user interface will look like and behave, by means of concept graphics and UX modeling.

section 5: Requirements Traceability. shows how the requirements in the RASD [3] are satisfied by the design choices of the DD.

2 Architectural Design

2.1 Overview

The PowerEnJoy's software architecture will be described starting from an high level view to an in detail view of a specific system component. We will discuss the role of these components and how can they interact through several interfaces.

A static and dynamic viewpoint are provided in order to better explain how the architecture works through its components.

2.2 High level view

In order to explain the role of the system's components, it's better considering the layers involved for each component. There are 3 different distinguishable logical layers that make the system architecture a 3 layer architecture. The layers are:

Presentation: this layer process data coming from the Application layer in order to display them on a GUI.

Application: this layer provides the core functionalities of the system.

Data: this layer provides a way to store the essential data for the system.

The main components of the system are the following:

Database: a data layer that supports a RDBMS.

Application server: an application layer responsible for the whole application logic of the system. All the policies, the algorithms and the computation are performed here. This layer offers a service-oriented interface.

Web server: a presentation layer responsible for translating HTTPS requests into API requests and API responses into HTTPS responses.

Mobile application: a presentation layer directly connected to the application server.

Car computer: two layers, a presentation and an application one, that provide respectively utilities to the user and an interface to the server in order to manage the car.

User's browser: a presentation layer consisting in a web browser. Not under control of the system.

The following figure shows all the components of the system, highlighting the logical layers for each component, grouped by the physical tiers.

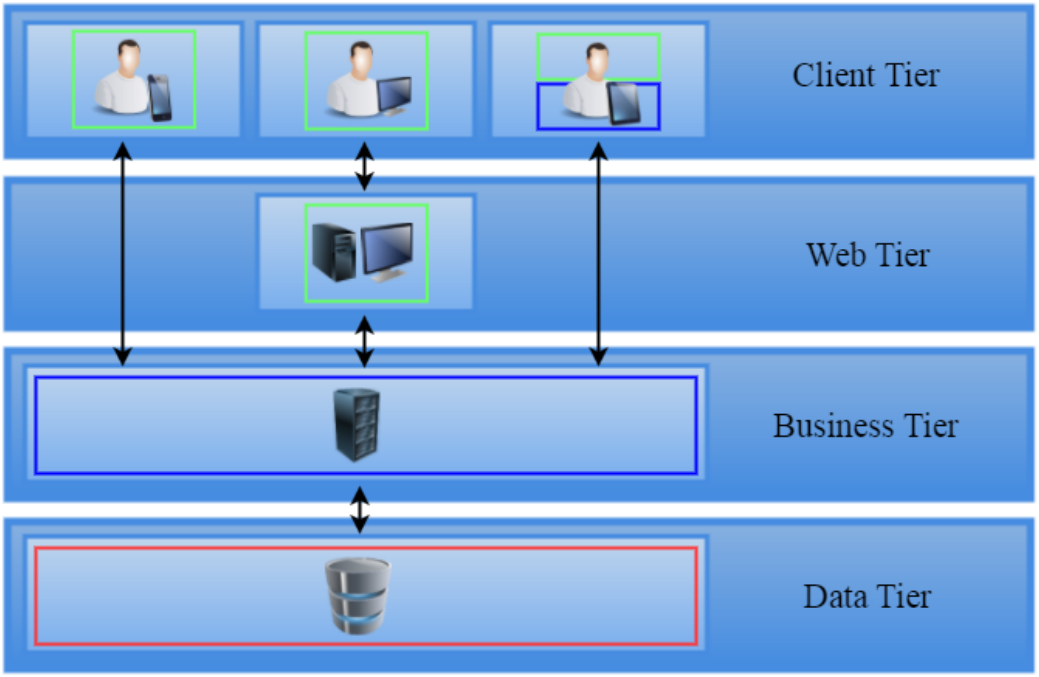

Figure 2.1: High level view of the PowerEnJoy architecture.

2.3 Component view

2.3.1 Database

The server database of the system runs MySQL Community Edition 5.7 with InnoDB as default storage engine. InnoDB provides the standard ACID-compliant transaction features.

The following properties are satisfied:

- users' passwords are not saved in plain-text, but they are salted with 8 random bytes and encrypted using the SHA-1 algorithm. The 8 random bytes are different for each password and obviously stored.
- access to the data must be granted only to authorized users possessing the right credentials.

- every software component that needs to access the DBMS must do so with the minimum level of privilege needed to perform the operations.

All the persistent application data are stored in the database. The relational model is illustrated in the following figure.

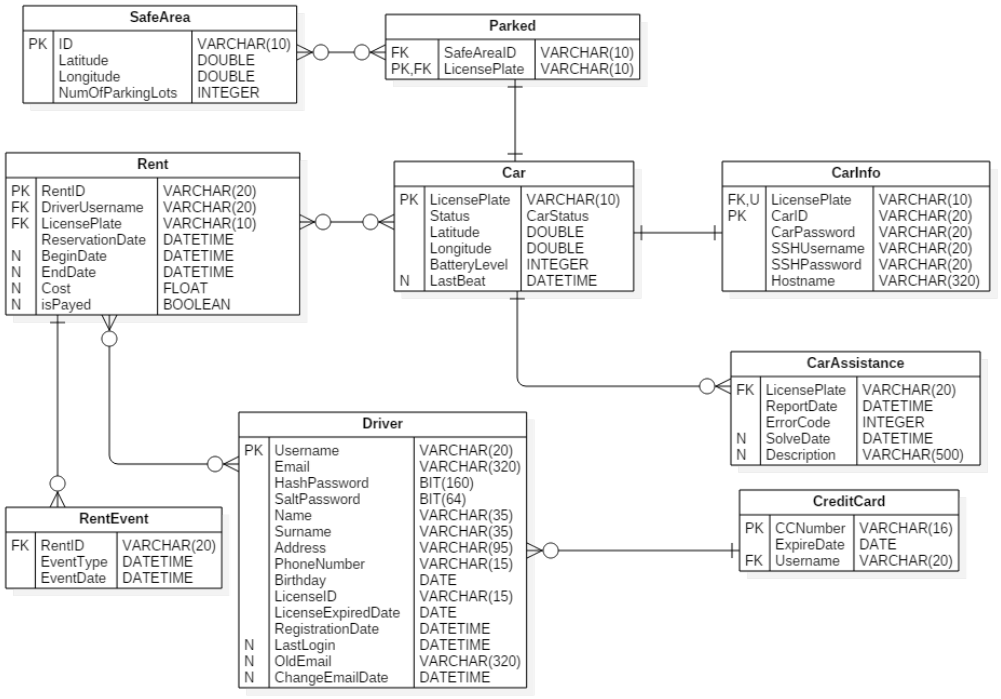

Figure 2.2: Relational model of the database schema.

Several triggers are implemented to manage some exceptions. For instance:

- the registration is cancelled if a driver has never logged into the system in 10 days since his/her registration.
- the rent is aborted if a driver doesn't unlock the reserved car within 1 hour. Rental cost is setted to 1 euro.
- a report of assistance to a car is generated if more than 10 minutes have passed from the last car heartbeat.

2.3.2 Application server

The application server runs on a GlassFish server and it's implemented using Servlets.

The business logic is implemented by custom-built EJB. Given that the state of users, cars and rents is stored in the database, all the EJB are stateless.

Session Beans used for the application server are shown in Figure 2.3.

DriverManager: manages all the user management features: user registration, user deletion, user profile editing and user salt bytes retrieving. User registration provides a function that sends an email, with a password and some guidelines, after the registration.

LogbookManager: allows users to fetch the history of their past rents.

SearchCarManager: allows users to get information about the nearby available cars.

DriverRentalManager: allows users to reserve cars, to unlock reserved cars and fetch information about the current rents.

CarHeartbeat: manages information coming from cars. It also provides a function responsible for locking cars.

CarRentalManager: manages rental information.

RentalFee: a singleton that reads the fee of the service from an XML configuration file.

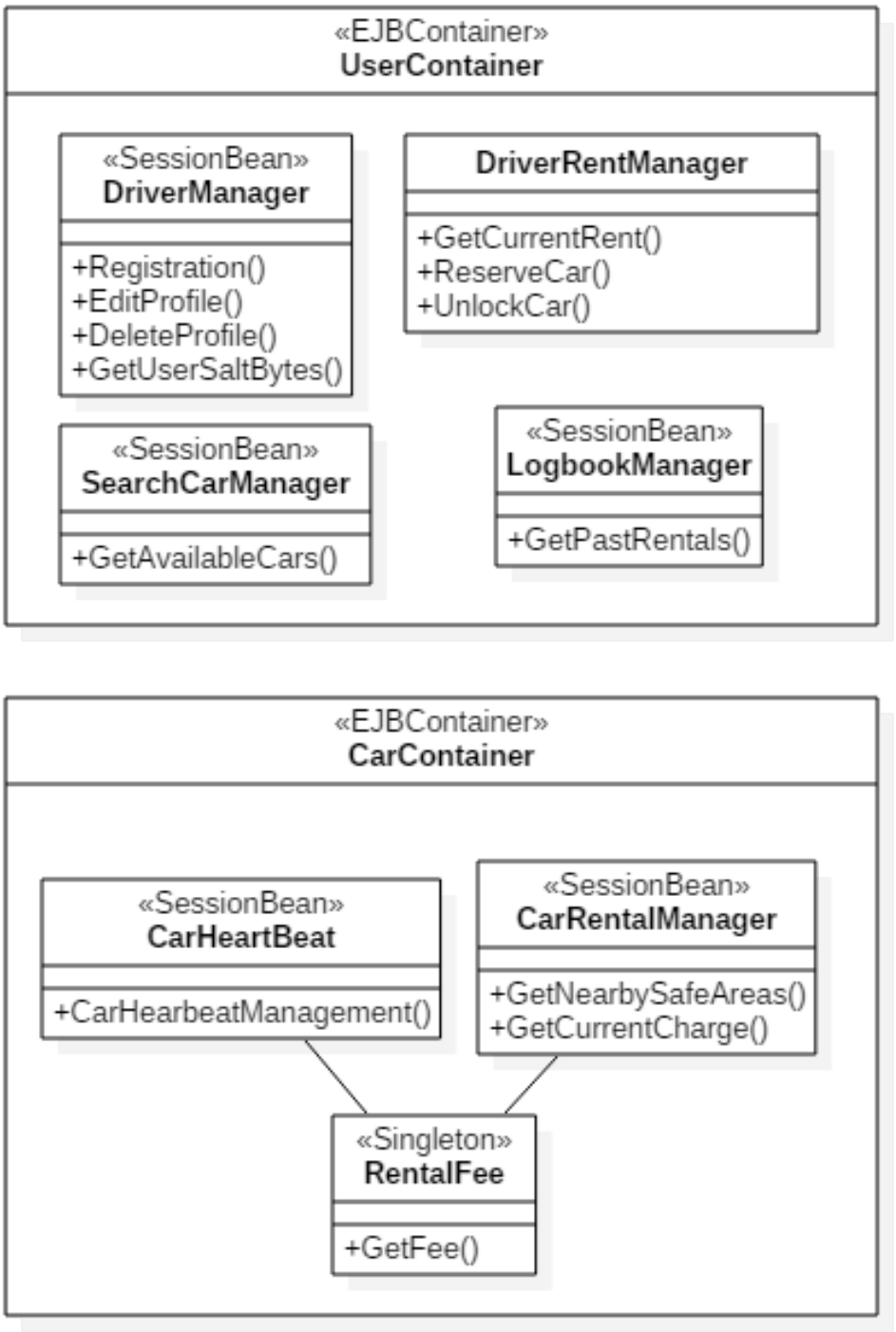

Figure 2.3: Session EJBs implemented server side.

The access to the DBMS is not implemented with direct SQL queries: instead, it is completely wrapped by the JPA. The object-relation mapping is done by entity beans. They are simple java classes without a constructor and fulfilled by getter and setter methods for each attributes, that correspond to fields of a database table.

2.3.3 Web server

The web server runs on GlassFish server. It's implemented using JSP; java EE web components useful when the goal of the web server is to translate RESTful API requests/responses into HTTP/HTTPS requests/responses. Thanks to JSPs, developers can focusing better on the GUI.

The API requests are managed by a translator, a stateless EJB that simplify the JSP job translating methods calls into API calls.

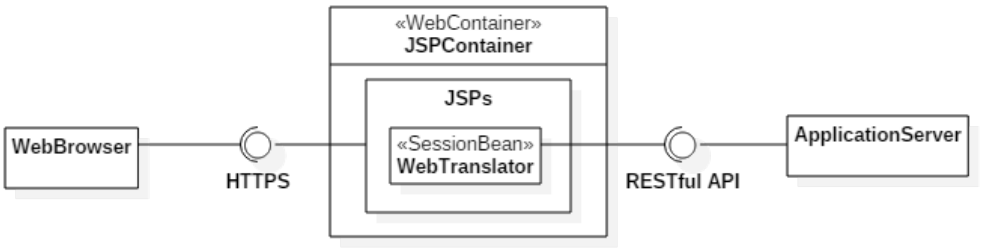

Figure 2.4: The components and the interfaces of the web tier.

Even if REST is a stateless architectural style, JSPs lead users to a stateful communication saving the session in order to avoid further insertion of user's credentials.

2.3.4 Car computer

Every car hosts a computer with a linux distribution of Ubuntu 16.10 as operating system. In any OS is installed openssh-server, in order to access to the car remotly, and two applications:

Heartbeat: this application is always running and sends, every 10 seconds, information about the current status, of both car and rent, to the application server through a single API call. The application server will ignore the rental information if the car is not rented.

CarGUI: this application is started everytime the car gets unlocked, and closed everytime the car gets locked. It asks, through API calls, for user information, user's current charges and nearby availabe safe areas, in order to display those information on the GUI.

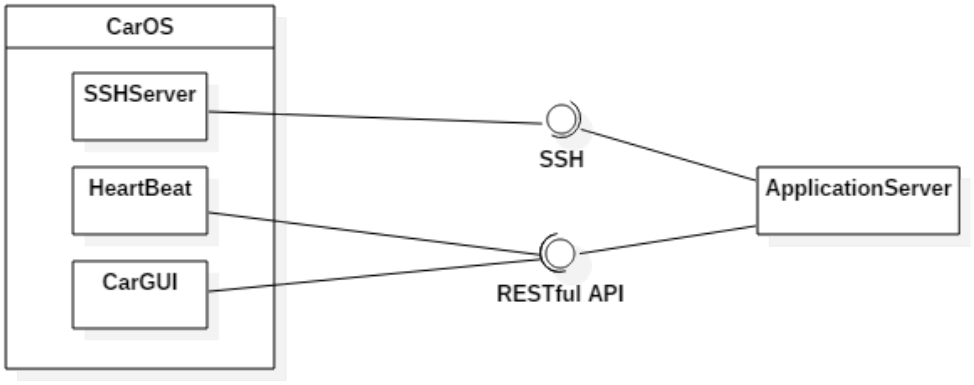

Figure 2.5: The components and the interfaces of the car client tier.

2.3.5 Mobile application

The mobile client implementation depends on the specific platform. The iOS application is implemented in Swift and mainly uses UIKit framework to manage the UI interface. Instead, the Android application is implemented in Java and mainly uses android.view package for graphical management.

The application core is composed by a controller which translates the inputs from the UI into remote functions calls via RESTful APIs. The controller also manages the interaction with the GPS component using CoreLocation framework in iOS app and LocationListener interface in the Android one.

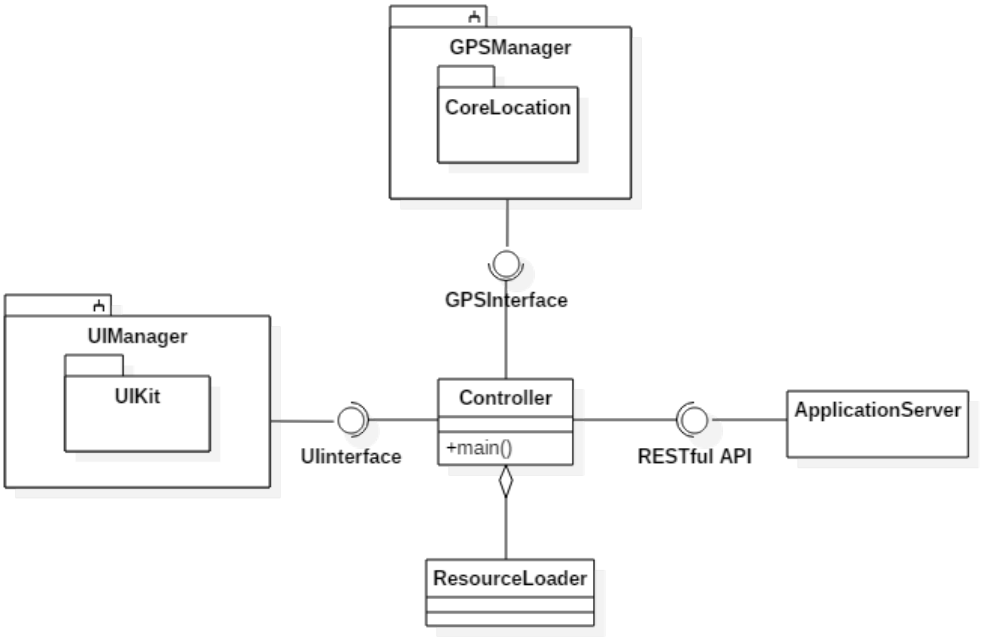

Figure 2.6: The components of the iOS application.

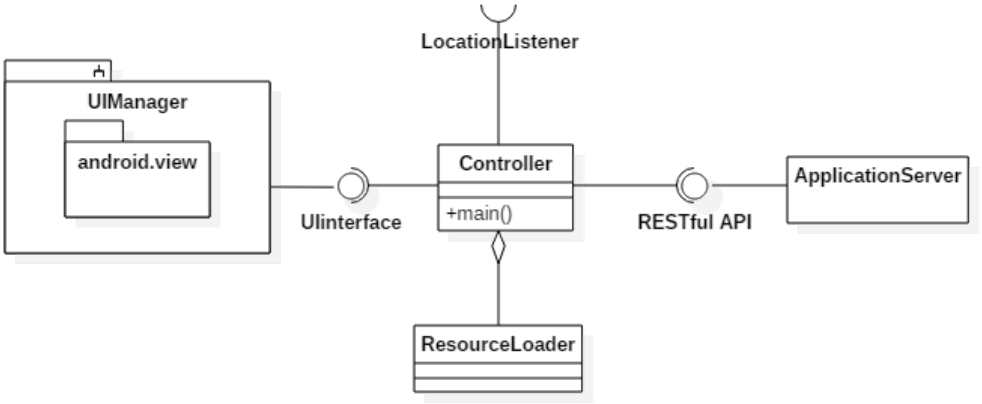

Figure 2.7: The components of the Android application.

2.4 Deployment view

The deployment diagram for the system is shown in Figure 2.8.

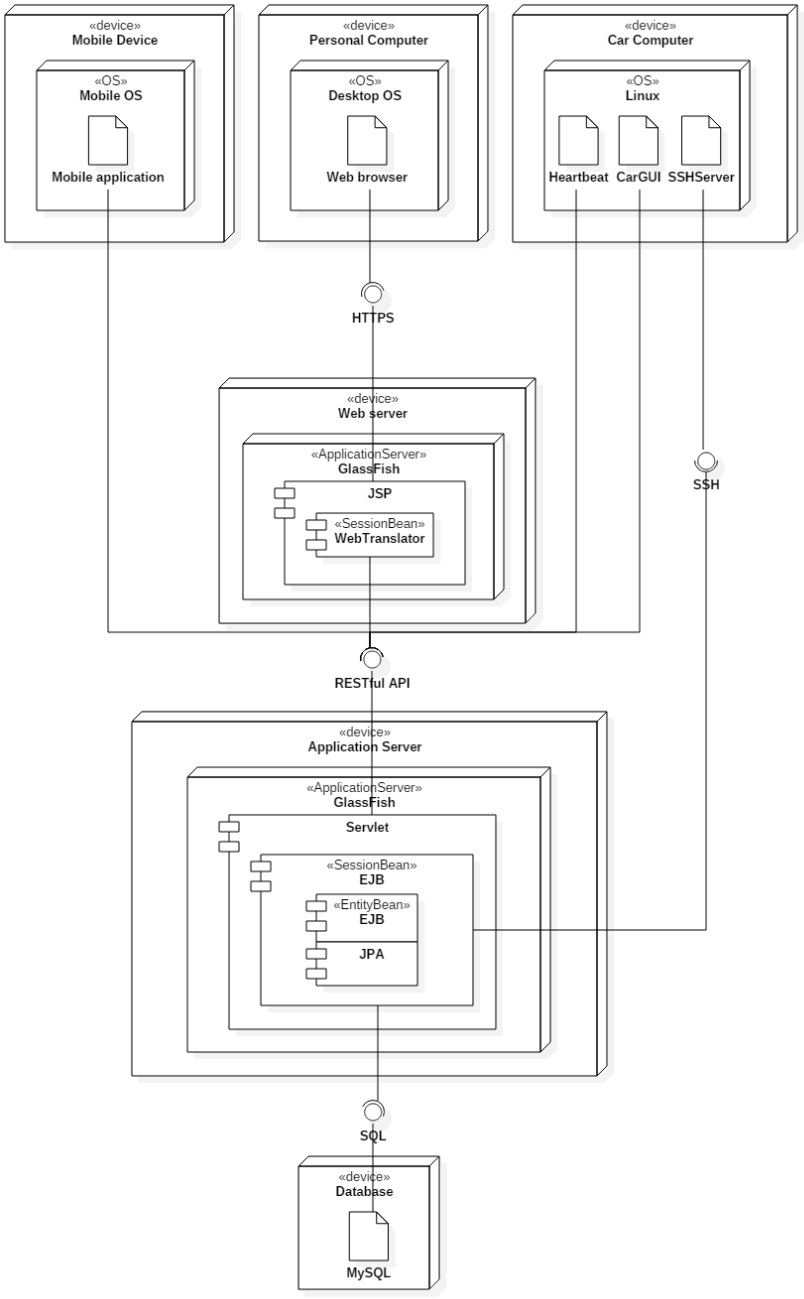

Figure 2.8: The deployment diagram of the PowerEnJoy system.

2.5 Component interfaces

2.5.1 Application server to database

The application server communicates with the DBMS via the Java Persistence API over standard network protocols.

The low-level technicalities about the specific dialect of SQL for the selected DBMS are abstracted by the Java Persistence API, which also deals with the O/R mapping.

2.5.2 Application server to cars

The application server communicates with the cars through the SSHv2 protocol. It provides an encrypted and remote communication over the standard network.

The credentials used for the remote access are stored in the database as plaintext and different for each car. There is no reason to protect the credentials hashing and/or salting them, because it won't add any security measure.

2.5.3 Front-ends to application server (API)

As already said, the front-ends applications communicate with the application server through RESTful API over the HTTPS protocol. They are provided by the application server and use JSON as data representation language.

The detailed RESTful API is described in the following pages.

Common to all the requests:

- the paths must be applied to the application server domain.
- a 422 error means that the JSON object has been not created correctly or the values of the objects' fields cannot be processed by the application server. For instance, the username choosen could be already taken or could not match to the regular expression choosen by the system.
- requests over HTTP protocol are denied.

Requests processed by cars also require an authentication process in order to avoid fake requests from unknown entities. As already said for the SSH credentials, they are stored in the database as plaintext, diffent for each car and from the SSH credentials.

2.5.3.1 [User] User creation

POST /api/user/add-user — Creates user (REST)

Request Classes

Request Value: User

User – This object contains all the user's information.

Name	Data Type	Description
username	string	The username of the user.
password	string	The password of the user.
name	string	The name of the user.
surname	string	The surname of the user.
address	string	The address of the user.
phoneNumber	number	The mobile phone number of the user.
birthday	string	The birthday of the user.
creditCard	CreditCard	The credit card associated to the account.
drivingLicense	DrivingLicense	The driving license of the user.
isAccepted	boolean	If the user accepted the terms and conditions or not.

CreditCard – This object contains credit card information.

Name	Data Type	Description
creditCardNumber	number	The number of the credit card.
expireDate	string	The credit card date of expiry.

DrivingLicense – This object contains driving license information.

Name	Data Type	Description
drivingLicenseID	string	The ID of the driving license.
expireDate	string	The driving license date of expiry.

Response Errors

HTTP Status Code	Reason
400	Bad Request
403	Forbidden – SSL or TLS required
422	Unprocessable Entity – Error reasons
500	Internal Server Error
503	Service Unavailable

2.5.3.2 [User] User's salt bytes retrieval

GET /api/user/{username}/salt-bytes — Retrieves user's salt bytes (REST)

Response Classes

Return Value: SaltBytes

SaltBytes – This object contains the user's salt bytes.

Name	Data Type	Description
saltValues	Array[number]	Salt bytes to use in the hash function.

Response Errors

HTTP Status Code	Reason
400	Bad Request
403	Forbidden – SSL or TLS required
404	Not Found – User not found
500	Internal Server Error
503	Service Unavailable

Path Parameters

Name	Data Type	Description
username	string	The username of the user.

2.5.3.3 [User] User deletion

DELETE /api/user/delete-user — Deletes user from the system (REST)

Response Errors

HTTP Status Code	Reason
400	Bad Request
403	Forbidden – Authentication error
403	Forbidden – SSL or TLS required
422	Unprocessable Entity – error reasons
500	Internal Server Error
503	Service Unavailable

Query Parameters

Parameter Name	Value Data Type	Value Data Description
username	string	The username of the user.
hashed_password	string	The hash of the user's salted password in hexadecimal notation.

2.5.3.4 [User] User's rental logbook retrieval

GET /api/logbook/past-rents — Retrieves user's rental logbook (REST)

Response Classes

Return Value: Logbook

Logbook – This object contains the user's rental logbook.

Name	Data Type	Description
rents	Array[Rent]	This list of the past rents.

Rent – This object contains the information about a single rent.

Name	Data Type	Description
licensePlate	string	The license plate of the rented car.
beginDate	string	The rental start date and time.
endDate	string	The rental end date and time.
cost	number	The rental cost.
isPayed	boolean	Is payed or not.

Response Errors

HTTP Status Code	Reason
400	Bad Request
403	Forbidden – Authentication error
403	Forbidden – SSL or TLS required
500	Internal Server Error
503	Service Unavailable

Query Parameters

Parameter Name	Value Data Type	Value Data Description
username	string	The username of the user.
hashed_password	string	The hash of the user's salted password in hexadecimal notation.

2.5.3.5 [User] Available cars' retrieval

GET /api/search-cars/{latitude}/{longitude} — Retrieves nearby available cars (REST)

Response Classes

Return Value: AvailableCars

AvailableCars – This object contains information about the available cars.

Name	Data Type	Description
cars	Array[Car]	The list of the available cars.

Car – This object contains information about a car.

Name	Data Type	Description
licensePlate	string	The license plate of the car.
batteryLevel	number	The battery level of the car.
position	Position	The position of the car.

Position – This object contains geographic latitude and longitude.

Name	Data Type	Description
latitude	number	The latitude value.
longitude	number	The longitude value.

Response Errors

HTTP Status Code	Reason
400	Bad Request
403	Forbidden – Authentication error
403	Forbidden – SSL or TLS required
500	Internal Server Error
503	Service Unavailable

Path Parameters

Name	Data Type	Description
latitude	number	The latitude value of the user's position.
longitude	number	The longitude value of the user's position.

Query Parameters

Parameter Name	Value Data Type	Value Data Description
username	string	The username of the user.
hashed_password	string	The hash of the user's salted password in hexadecimal notation.

2.5.3.6 [User] Car reservation

GET /api/rental-operations/{licensePlate}/reserve — Reserves car (REST)

Response Errors

HTTP Status Code	Reason
400	Bad Request
403	Forbidden – Authentication error
403	Forbidden – SSL or TLS required
404	Not Found – Car not found
422	Unprocessable Entity – Error reasons
500	Internal Server Error
503	Service Unavailable

Path Parameters

Name	Data Type	Description
licensePlate	string	The license plate of the car that the user wants reserve.

Query Parameters

Parameter Name	Value Data Type	Value Data Description
username	string	The username of the user.
hashed_password	string	The hash of the user's salted password in hexadecimal notation.

2.5.3.7 [User] Current reservation retrieval

GET /api/rental-operations/current-reservation — Retrieves current reservation (REST)

Response Classes

Return Value: CurrentReservation

CurrentReservation – This object contains information about the current reservation.

Name	Data Type	Description
car	Car	The reserved car.
reservationDate	string	The rental reservation date and time.

Car – This object contains information about the reserved car.

Name	Data Type	Description
licensePlate	string	The license plate of the car.
batteryLevel	number	The battery level of the car.
position	Position	The position of the car.

Position – This object contains geographic latitude and longitude.

Name	Data Type	Description
latitude	number	The latitude value.
longitude	number	The longitude value.

Response Errors

HTTP Status Code	Reason
400	Bad Request
403	Forbidden – Authentication error
403	Forbidden – SSL or TLS required
404	Not Found – Reservation not found
500	Internal Server Error
503	Service Unavailable

Query Parameters

Parameter Name	Value Data Type	Value Data Description
username	string	The username of the user.
hashed_password	string	The hash of the user's salted password in hexadecimal notation.

2.5.3.8 [User] Unlocks car

PUT /api/rental-operations/unlock-reserved-car — Unlocks car (REST)

Requeste Classes

Request Value: Position

Position – This object contains user's geographic latitude and longitude.

Name	Data Type	Description
latitude	number	The latitude value.
longitude	number	The longitude value.

Response Errors

HTTP Status Code	Reason
400	Bad Request
403	Forbidden – Authentication error
403	Forbidden – SSL or TLS required
404	Not Found – Car not found
422	Unprocessable Entity – Error reasons
500	Internal Server Error
503	Service Unavailable

Query Parameters

Parameter Name	Value Data Type	Value Data Description
username	string	The username of the user.
hashed_password	string	The hash of the user's salted password in hexadecimal notation.

2.5.3.9 [Car] Car Heartbeat

PUT /api/car/heart-beat — Car Heartbeat (REST)

Request Classes

Request Value: Heartbeat

Hearbeat – This object contains information about the current status of the car and the rent.

Name	Data Type	Description
carInformation	CarInformation	The information about the car.
rentalInformation	RentalInformation	The information about the current rent.

CarInformation – This object contains information about the car.

Name	Data Type	Description
batteryLevel	number	The battery level of the car.
position	Position	The position of the car.
brokenParts	Array[string]	The list of the broken parts.

Position – This object contains geographic latitude and longitude.

Name	Data Type	Description
latitude	number	The latitude value.
longitude	number	The longitude value.

RentalInformation – This object contains information about the current rent.

Name	Data Type	Description
events	Array[Event]	The list of the events occurred between the past and the current heartbeat.
isLockable	boolean	Is true if the car is turned off and closed correctly.
isEmpty	boolean	Is true if there is no one in the car.

Event – This object contains an event occurred on the car.

Name	Data Type	Description
eventType	string	The type of the event.
eventDate	string	The event date and time.

EventType

Value	Description
TURNED_ON	The car got turned on.
TURNED_OFF	The car got turned off.
PASSENGERS_DETECTED	The sensors detected at least 1 passenger.
ALL_PASSENGERS_LEFT	The sensors detected that all passangers left the car.

Response Errors

HTTP Status Code	Reason
400	Bad Request
403	Forbidden – Authentication error
403	Forbidden – SSL or TLS required
422	Unprocessable Entity – Error reasons
500	Internal Server Error
503	Service Unavailable

Query Parameters

Parameter Name	Value Data Type	Value Data Description
car_id	string	The id of the car.
car_password	string	The password of the car.

2.5.3.10 [Car] Available safe areas' retrieval

GET /api/car/safe-areas/{latitude}/{longitude} Retrieves nearbyavailable safe areas (REST)

Response Classes

Return Value: SafeAreas

SafeAreas – This object contains information about the available safe areas.

Name	Data Type	Description
areas	Array[Area]	The list of the available areas.

Area – This object contains information about a parking area.

Name	Data Type	Description
AreaID	string	The ID of the area.
position	Position	The position of the area.

Position – This object contains geographic latitude and longitude.

Name	Data Type	Description
latitude	number	The latitude value.
longitude	number	The longitude value.

Response Errors

HTTP Status Code	Reason
400	Bad Request
403	Forbidden – Authentication error
403	Forbidden – SSL or TLS required
500	Internal Server Error
503	Service Unavailable

Path Parameters

Name	Data Type	Description
latitude	number	The latitude value of the car's position.
longitude	number	The longitude value of the car's position.

Query Parameters

Parameter Name	Value Data Type	Value Data Description
car_id	string	The id of the car.
car_password	string	The password of the car.

2.5.3.11 Other requests

There are other requests, but they are not relevant as the previous requests. For instance:

- user's information updating.
- payment execution.
- current charging information retrieval.
- car information retrieval.

2.5.4 Web server to browser

The users' browsers communicate with the web server via HTTPS requests. Any unencrypted request will be denied.

2.5.5 External interfaces

The application server has two external interfaces:

SMSGateway: to send SMS to users.

PaymentGateway: to execute credit card payments of users.

2.6 Runtime view

The dynamic behaviour of the system is described through sequence diagrams. The following diagrams highlight the most interesting functionalities of the system.

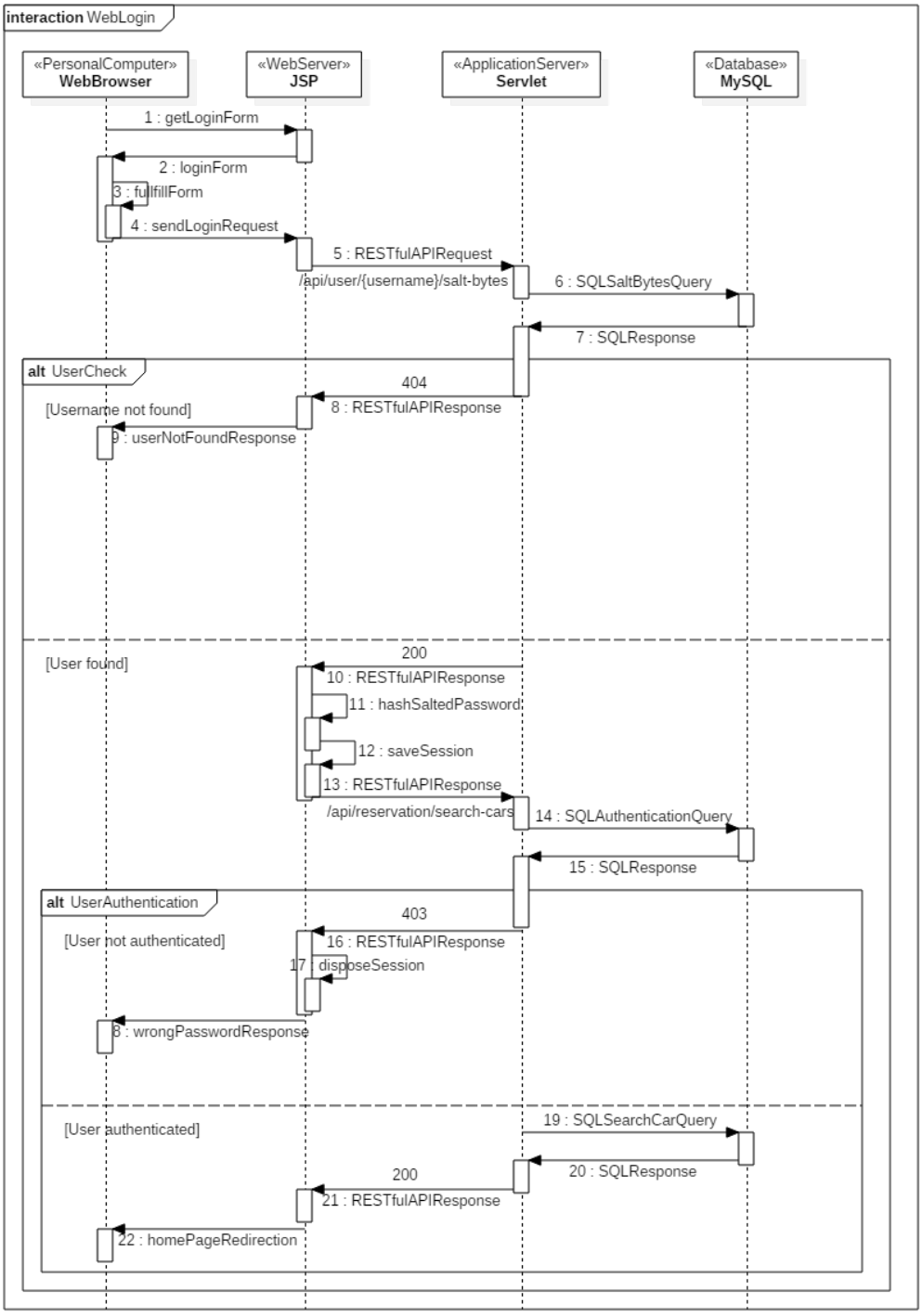

Figure 2.9: Sequence diagram of the login through the web interface.

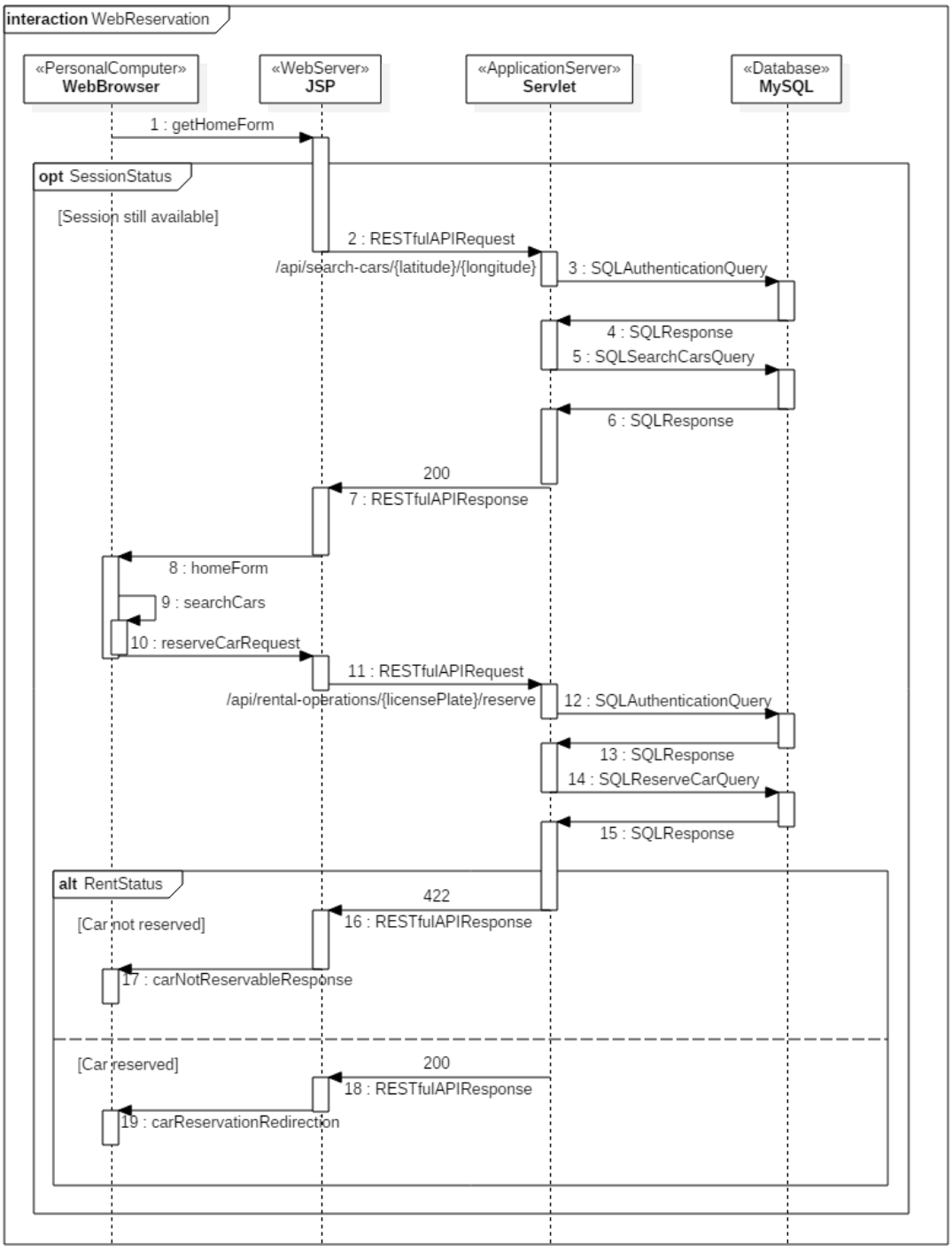

Figure 2.10: Sequence diagram of the reservation through the web interface.

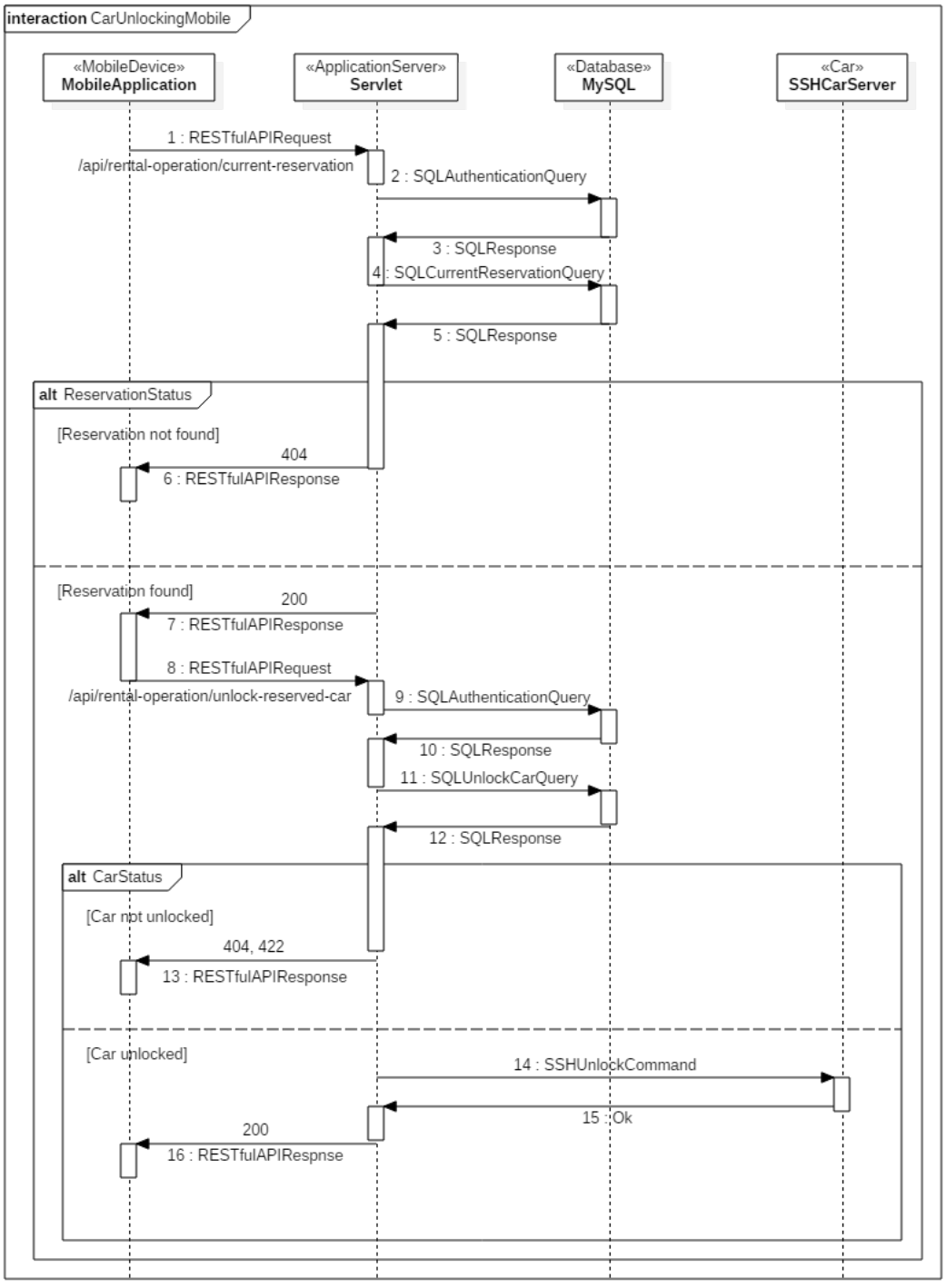

Figure 2.11: Sequence diagram for car unlocking through the mobile application.

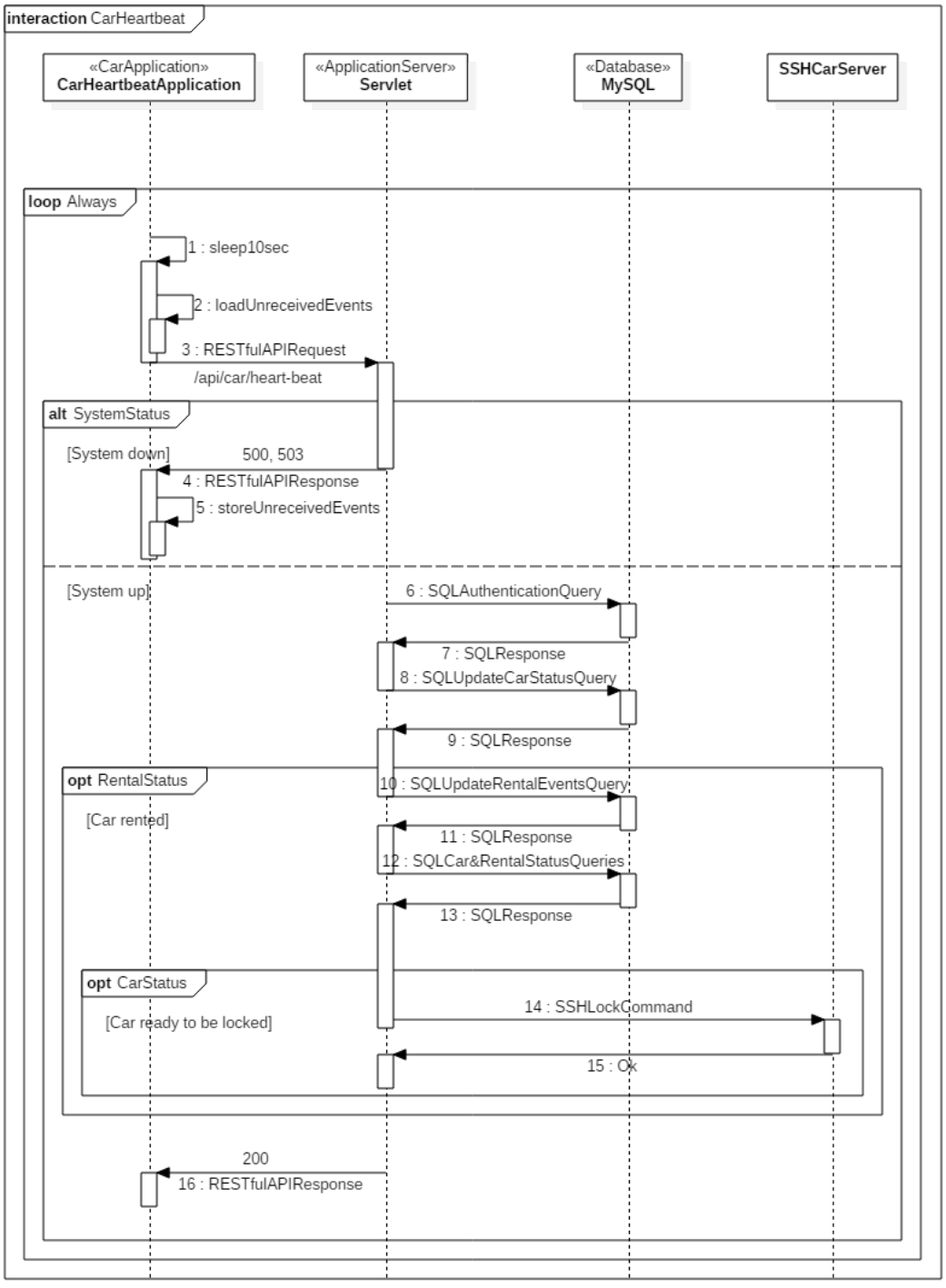

Figure 2.12: Sequence diagram of the car heartbeat process and car locking.

2.7 Why this architecture

A good starting point, as an inexperienced software designer, was to keep the level of abstraction of my software as high as possible.

The first step has been to define the different logical layers in order to assign all the functionalities of the system to the respective logical layer. Each layer have been assigned to a physical tier, creating a multitier architecture.

The choice of the REST architectural style increased the level of abstraction of the entire software architecture and brought properties like:

Performance: application server doesn't save any session, so, after replied to a request, is ready to reply to a new request from a different user. The number of active threads is equal to the number of requests that the application server is processing.

Scalability: it's possible to add new components, without modifying those that already exist, in order to manage more requests from users.

Simplicity: the RESTful API is an uniform interface.

Modifiability: it's simple to modify the components.

Portability: components are duplicated easily.

Reliability: if there are more web servers, application servers and databases, the system will be able to provide the service even if a component fails.

Last but not least, Java EE offered a set of components that hugely increased the level of abstraction.

3 User Interface Design

The mockups have already been shown in the RASD [3].

4 Algorithm Design

The proposed algorithms are the most interesting, and less intuitive, algorithms of the business logic. We won't be using any reference to the Java EE components in order to look at these algorithms in the most abstract way possible. The convention used for them contains:

- *variables*.
- not_implemented_methods(): intuitive methods that often interact with other components like the database or the car.
- implemented_methods(): methods described later.
- COSTANTS.

4.1 Car Heartbeat

The car hearbeat management is a core function of the businnes logic. It checks the cars' status and execute actions based on their status.

The associated request provides the following infomation to the application server:

JSON Object: it contains:

- $carInformation$: the information about the car.
- $rentInformation$: the information about the current rent.

Path Parameters: they are:

- $carID$: the car ID used to get authenticate by server.
- $carPassword$: the car's password used to get authenticate by server.

The algorithm used to manage the car hearbeat is the following.

```
if (verifyCredentials (carID, carPassword)) then
    updateCarStatus (carID, carInformation);
    if (rentedCar (carID)) then
        updateRentalStatus (carID, rentInformation);
    end
    HTTPResponseCode(200);
else
    HTTPResponseCode(403);
end
```

4.1.1 Car status updating

The parameters of this function are:

- $carInformation$: the information about the car.
- $carID$: the car ID used to get authenticate by server.

The $carInformation$ object contains:

- $batteryLevel$: the battery level of the car.
- $position$: an object that contains the geographical postion of the car.
- $brokenParts[]$: the list of the broken parts of the car.

The $position$ object contains:

- $latitude$: the latitude value.
- $longitude$: the longitude value..

The algorithm is the following.

```
updateCarValues(carID, batteryLevel, position);
if (brokenParts != null) then
    if (getOpenedAssistanceReport (carID, 105) == null) then
        createAssistanceReport (carID, 105, brokenParts[]);
    end
end
if (batteryLevel == 0) then
    if (getOpenedAssistanceReport (carID, 103) == null) then
        createAssistanceReport (carID, 103);
    end
else
    if (batteryLevel <= 25 && !rentedCar (carID)) then
        if (getOpenedAssistanceReport (carID, 104) == null) then
            setCarOutOfService (carID);
            createAssistanceReport (carID, 104);
        end
    end
end
```

4.1.2 Rental status updating

The parameters of this function are:

- $rentInformation$: the information about the current rent.
- $carID$: the car ID used to get authenticate by server.

The $carInformation$ object contains:

- $events[]$: the list of rental events.
- $isEmpty$: is true if there is no one in the car.
- $isLockable$: is true if the car is turned off and closed correctly.

The algorithm is the following.

```
rentID = getRentID (carID );
if (events != null) then
    updateEvents (rentID, events);
end
if (isLockable && isEmpty) then
    if (inASafeArea (carID)) then
        closeCar (carID);
        closeRent (rentID);
        calculateCost (rentID);
        checkCarStatus (carID); Similar to updateCarStatus without 1st
        line
        executePayment (rentID);
    end
end
Other error cases...
```

4.1.3 Rental cost calculation

The parameters of this function are:

- $rentID$: the rental ID.

The algorithm is the following.

```
endTime = getRentalEndDate (rentID);
beginTime = getFirstRentalEventByType (rentID, TURNED_ON);
if (beginTime == null) then
|   beginTime = getRentalBeginDate (rentID);
end
rentalTime = (endTime - beginTime);
totalCost = (rentalTime : 60) * getFee();
passengerTimes[] = orderArraysByTime (getRentalEventsByType
 (rentID, PASSENGERS_DETECTED), getRentalEventsByType
 (rentID, ALL_PASSENGERS_LEFT));
if (passengerTimes != null) then
|   passengerTotalTime = 0;
|   for i = 0 to passengerTimes.length() do
|   |   passengerTotalTime = passengerTimes[i + 1] -
|   |    passengerTimes[i];
|   |   i = i + 2;
|   end
|   if (passengerTotalTime >= (rentalTime * 0.5)) then
|   |   totalCost = totalCost * 0.8;
|   end
end
setCost (rentID, totalCost);
```

5 Requirements Traceability

All the decisions in the DD have been taken following functional and non-functional requirements written in the RASD [3]. Some minor tweak must still be brought to the RASD.

5.1 Functional requirements and components

Table 2 maps the functional requirements contained in specific sections of the RASD to the components in the Design Document that fulfill them.

Component (DD)	**Requirements (RASD)**
DriverManager	3.2.1 User registration
SearchCarManager	3.2.2 Search and reservation
DriverRentalManager	3.2.2 Search and reservation 3.2.3 Car usage 3.2.5 Service discounts and fees
CarHeartbeat	3.2.4 Car parking 3.2.5 Service discounts and fees
CarRentalManager	3.2.4 Car parking

Table 1: Table that links components to functional requirements.

5.2 Non functional requirements and components

Table 2 maps the non functional requirements contained in specific sections of the RASD to the components in the Design Document that fulfill them.

Component (DD)	**Requirements (RASD)**
Mobile application	3.1.1 User interfaces (Mobile application) 3.1.3 Software interfaces (Mobile application)
Web application Web server	3.1.1 User interfaces (Web application) 3.1.3 Software interfaces (Web application)
Mobile application Web application Web server	3.1.1 User interfaces (Common to web and mobile applications)
Car computer	3.1.1 User interfaces (Car application) 3.1.2 Hardware interfaces 3.1.3 Software interfaces (Car application)
CarRentalManager	3.1.1 User interfaces (Server back-end)
LogbookManager	3.1.1 User interfaces (Rental logbook)

Table 2: Table that links components to functional requirements.

A Appendix

A.1 Used tools

Tools used to create the DD:

- **MikTex**
  Distribution of the typesetting system LaTeX
  http://miktex.org/
- **StarUML**
  UML modelling tool we used to build the graphs
  http://staruml.io/
- **Draw.io**
  Desktop application for mockups and diagrams creation
  http://draw.io/
- **GitHub desktop**
  Desktop application of the web-based Git repository hosting service. Used to collaborate in the team and to have a track of the changes.
  https://desktop.github.com/

A.2 Hours of work

This is the time spent redacting the DD

- Lorenzo Binosi - 68 hours

References

[1] Software Engineering 2 Project, AA 2016/2017 - *Project goal, schedule and rules*

[2] Software Engineering 2 Project, AA 2016/2017 - *Assignments 2*

[3] *Software Engineering 2: PowerEnJoy Requirements Analysis and Specification Document* Binosi Lorenzo Politecnico di Milano