POLITECNICO DI MILANO
2016-2017

SOFTWARE ENGINEERING 2: POWERENJOY

DESIGN DOCUMENT
VERSION 1.0

PEVERELLI FRANCESCO
REPPUCCI FEDERICO

TABLE OF CONTENT

1. INTRODUCTION

1.1. PURPOSE

The aim of this document is to provide a complete specification of the PowerEnJoy system's architecture which fulfills the requirements identified during the requirements specification phase. More specifically, this document is meant to serve both as a way to clarify to the stakeholders how the specified system will fulfill the requirements and how the requirements have motivated the design decisions presented, as well as a reference for the developers of the system that will guide the implementation phase.

1.2. SCOPE

The PowerEnJoy project aims to develop a car-sharing service run exclusively employing electric cars. The system will provide a mobile application by means of which the users, once registered, will be able to use the car sharing services. The main goals of the service are to provide a sustainable and environmentally-friendly car sharing service as well as to promote virtuous behaviors from its users.

1.3. GLOSSARY

Acronyms and abbreviations

App: mobile application

Pub-sub: publish subscribe

UML: Unified Modelling Language

OS: Operating System

DBMS: Database Management System

API: Application Programming Interface

J2EE: Java 2 Enterprise Edition

IDE: Interactive Development Environment

Requirements specification terminology

non registered person: as far as our system is concerned, we consider only those people who possess a smartphone with GPS functionalities.

user: person who is logged into his/her account.

driver: person who enters a car in the driver seat.

system: the server side software providing the core functionalities of the application.

application or app: the client side of the software present on the user's phone.

registration: iter through which the user can create a personal account in order to access the services of the application.

credentials: set of information provided by the user during the registration. These include: the user's first name, family name, gender, "Comune" of birth, "Provincia" of birth, his/her "Codice Fiscale", his/her identity card number, date of release and date of expiration, a valid driving license (B or higher or equivalent) a valid e-mail address and a mobile phone number.

payment info: a valid credit card number, verification value (CVV), expiration date and the holder's full name.

external payment service: a software system which allows the company to charge the users.

car: electric powered vehicles owned by PowerEnJoy.

range: distance value selected by the user.

available car: a car that can be reserved by a user for a future ride.

reserved car: a car that cannot be reserved and can only be used by the one who performed its reservation.

'in use' car: a car is in this state from the moment it is turned on by the user who reserved it until the moment such user exits from it while the car is in a safe area, which causes it to automatically close.

parked car: a car that is left in a safe area by the driver.

dislocated car: a car that is left in a non-safe area by the driver.

retrieve a car: action performed by an employee that can be described as follows:

-the employee is notified that a car has been left outside of a safe area

-the employee reaches the car, possibly manually recharges it and drives it back to a safe area.

safe area: legal parking spots according to the driving regulations within a limited area defined by the system administrator.

'nearby' the car: a user is 'nearby' the car when he/she is distant from the car less than 10 meters.

special parking area: part of safe area in which user can recharge the electrical car.

power grid station: little tower that provides electrical current situated in a special parking area that allows the user to recharge a car.

1.4. REFERENCES

[1] Francesco Peverelli and Federico Reppucci. "*RASD_v1.1.pdf*", 13 Nov. 2016.

[2] IEEE Computer Society. "*IEEE Standard for Information Technology—Systems Design—Software Design Descriptions*." Institute of Electrical and Electronics Engineers, Inc., 29 July 2009.

[3] Fakhroutdinov, Kirill. “The Unified Modeling Language”. *Unified Modeling Language (UML) Description, UML Diagram Examples, Tutorials and Reference for All Types of UML Diagrams - Use Case Diagrams, Class, Package, Component, Composite Structure Diagrams, Deployments, Activities, Interactions, Profiles, Etc.*, www.uml-diagrams.org/.

1.5. SUMMARY

1. OVERVIEW: this section explains how the document is structured, and the reasoning behind the choice of this particular approach.

2. COMPONENT VIEW: this section shows the external software services that interact with the system and the structure of the system's main software components.

3. HIGH-LEVEL SYSTEM ARCHITECTURE: this section describes the Software Architectural Pattern of choice as well as the physical architecture of the system.

4. COMPONENTS INTERACTION: this section describes more precisely how the software components are connected, which interfaces they provide and how they interact with each other.

5. COMPONENTS ARCHITECTURE AND PATTERNS: this section specifies for each software component its architecture, as well as the main design patterns used to construct it and their specific purpose.

6. USER INTERFACE DESIGN: this section describes in detail the functionalities of the user interfaces.

7. SELECTED TOOLS: this section specifies all the main frameworks, languages, libraries and tools to use during the development of the system.

8. DEPLOYMENT VIEW: this section describes how the software components are mapped onto the system's hardware and how they interact with each other at runtime. Moreover, a set of requirements regarding the execution environment of the software is identified.

9. ALGORITHM DESIGN: this section describes the most critical algorithms to implement, providing a pseudo-code implementation.

10. REQUIREMENTS TRACEABILITY: this section explains how the requirements previously identified are met by the system described in the document.

2. BODY

2.1. OVERVIEW

The structure of this document is meant to reflect the actual stages of the development of the architecture and design of the system, in order to illustrate not only the final result of the design phase, but also the intermediate steps of this process, and the rationale behind every major design decision. In order to achieve this, the document does not follow a strictly top-down structure, but starts from the design decisions that follow most directly from the requirements, namely the design of the system's main functional components. It is important to note that despite this difference in the rationale behind the structure of the

document, for the most part the approach is still top-down, since such an approach is the most natural way of obtaining a cohesive design.

2.2. COMPONENT VIEW

In this section two different levels of abstraction are presented, in the form of component diagrams. The first level describes how the main software components of the system interface themselves with the external software components necessary to achieve some of the required functionalities, while the second level illustrates a set of selected software components that achieve all the required functionalities of the system.

2.2.1. EXTERNAL COMPONENTS AND INTERFACES

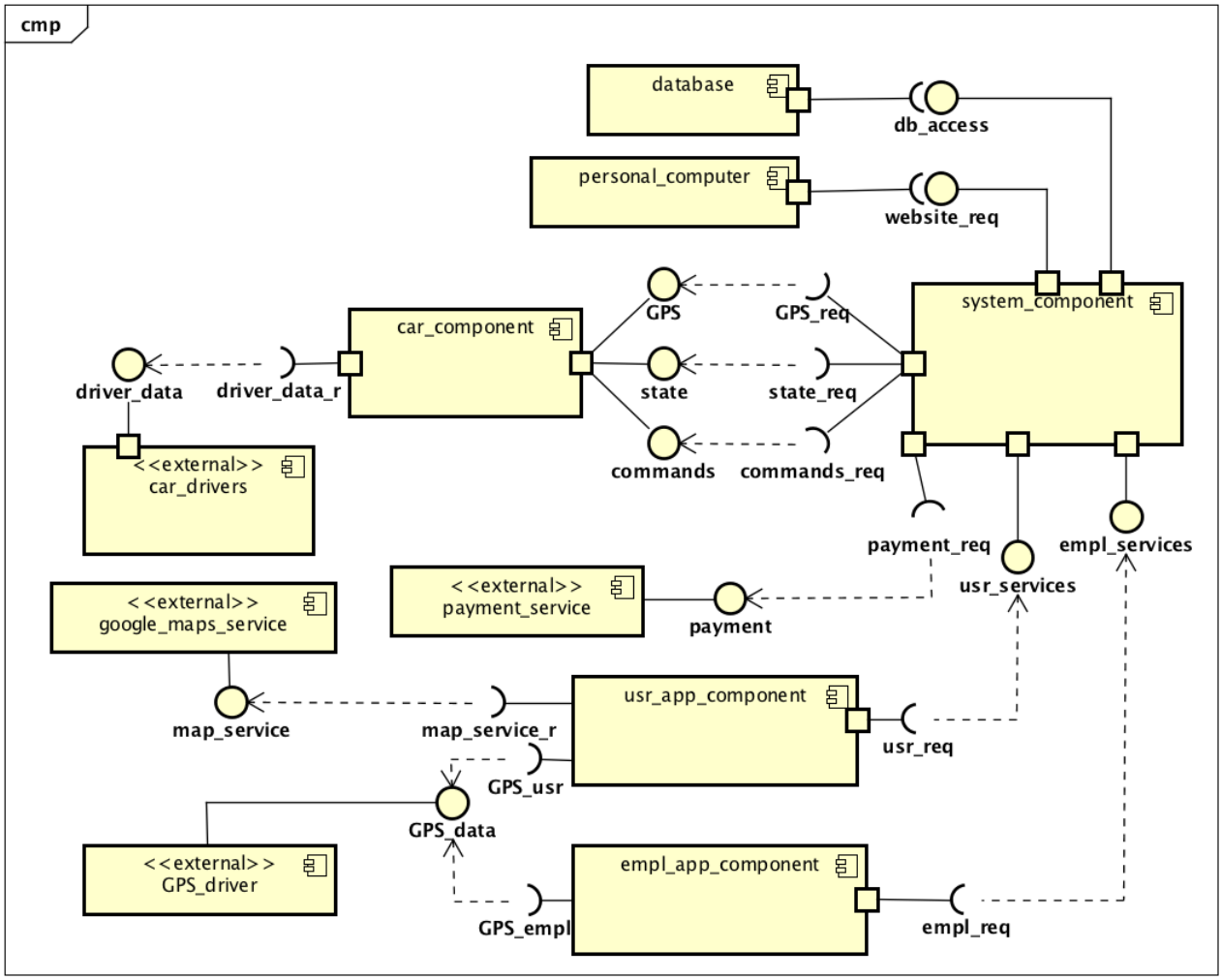

Fig. 1: high level components

- system_component: system's server software component

- car_component: software component handling the logic, presentation and communication in a car

- car_drivers: external component representing all the software drivers in a car

- usr_app_component: software component running on the user's phone

- empl_app_component: software component running on an employee's phone

- personal_computer: represents the software component of a home PC or similar device requesting a web page from the server

- GPS_driver: external component providing an interface for GPS localization

- google_maps_service: external software component providing the maps for the application

- payment_service: external component providing an interface to charge the users

- database: the persistent data storage unit

2.2.2. SOFTWARE COMPONENTS VIEW

Car component

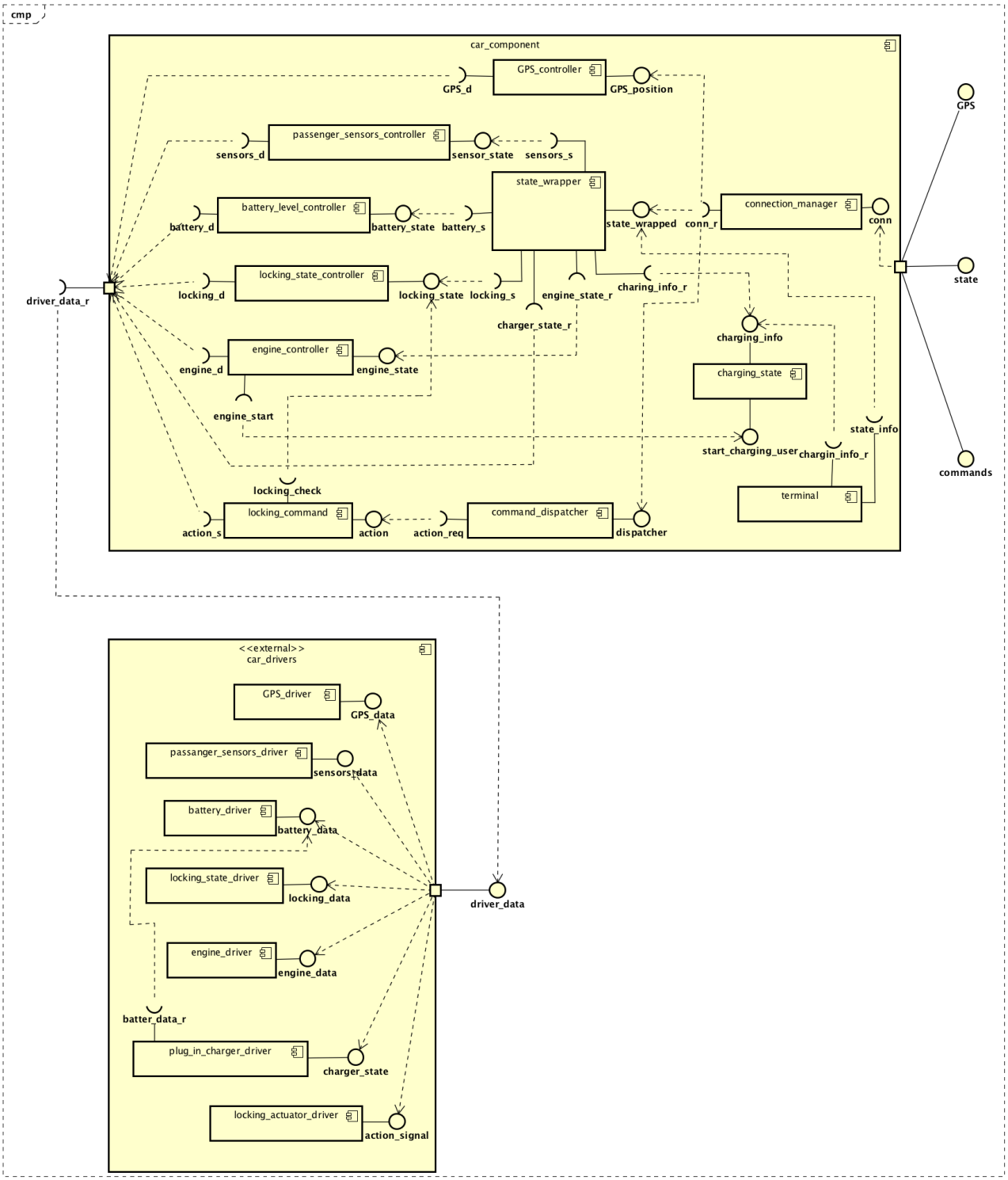

Fig. 2: car component

- connection_manager: component which handles the connection with the system's server, also manages the communication encryption

- state_wrapper: components which collects and manages all the information regarding a car's state coming from the various controllers

- GPS_controller: software components which interacts with the GPS device of the car to communicate the car's position to the system

- passengers_sensors_controller: manages the information provided by the car's sensors in the seats

- battery_level_controller: manages the information about the car's battery level

- locking_state_controller: manages the locking state of the car

- engine_controller: manages the information about the car's engine state

- command_dispatcher: component designed to handle all the commands from the system's server to the car

- locking_command: accepts locking/unlocking commands from the system and manages the car's locking_actuator_driver

- charging_state: keeps track of the current charges to the user

- terminal: displays all the relevant information about the car's state and the current charges to the user

- locking_actuator_driver: driver to lock or unlock the car

- plug_in_charger_driver: external driver which signals directly to the state_wrapper whether the car is plugged in a power supply

- *_driver: driver for the corresponding '*' controller

User application

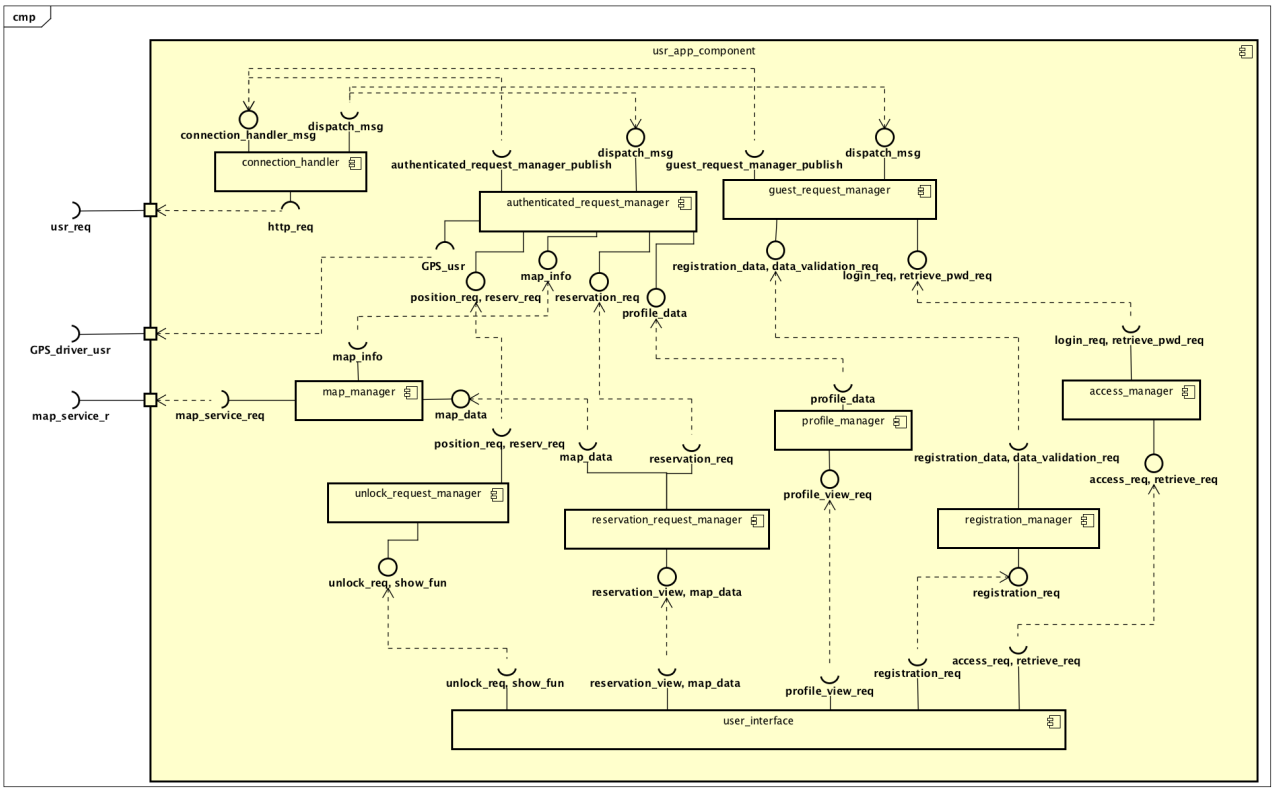

Fig. 3: user application component

- user_interface: manages the interaction with the app's user
- access_manager: generates the password retrieval and login requests of the user
- registration_manager: generates the registration requests by the user
- profile_manager: generates the requests to the user's profile pages
- reservation_requests_manager: generates reservation requests
- unlock_request_manager: generates requests to unlock a car
- map_manager: manages the interaction with the external service providing the maps
- guest_request_manager: forwards all the unautenthicated requests
- authenticated_request_manager: forwards all the authenticated requests
- connection_handler: manages the connection with the system's server, also manages the communication encryption
- GPS_driver: the phone's GPS device

Employee application

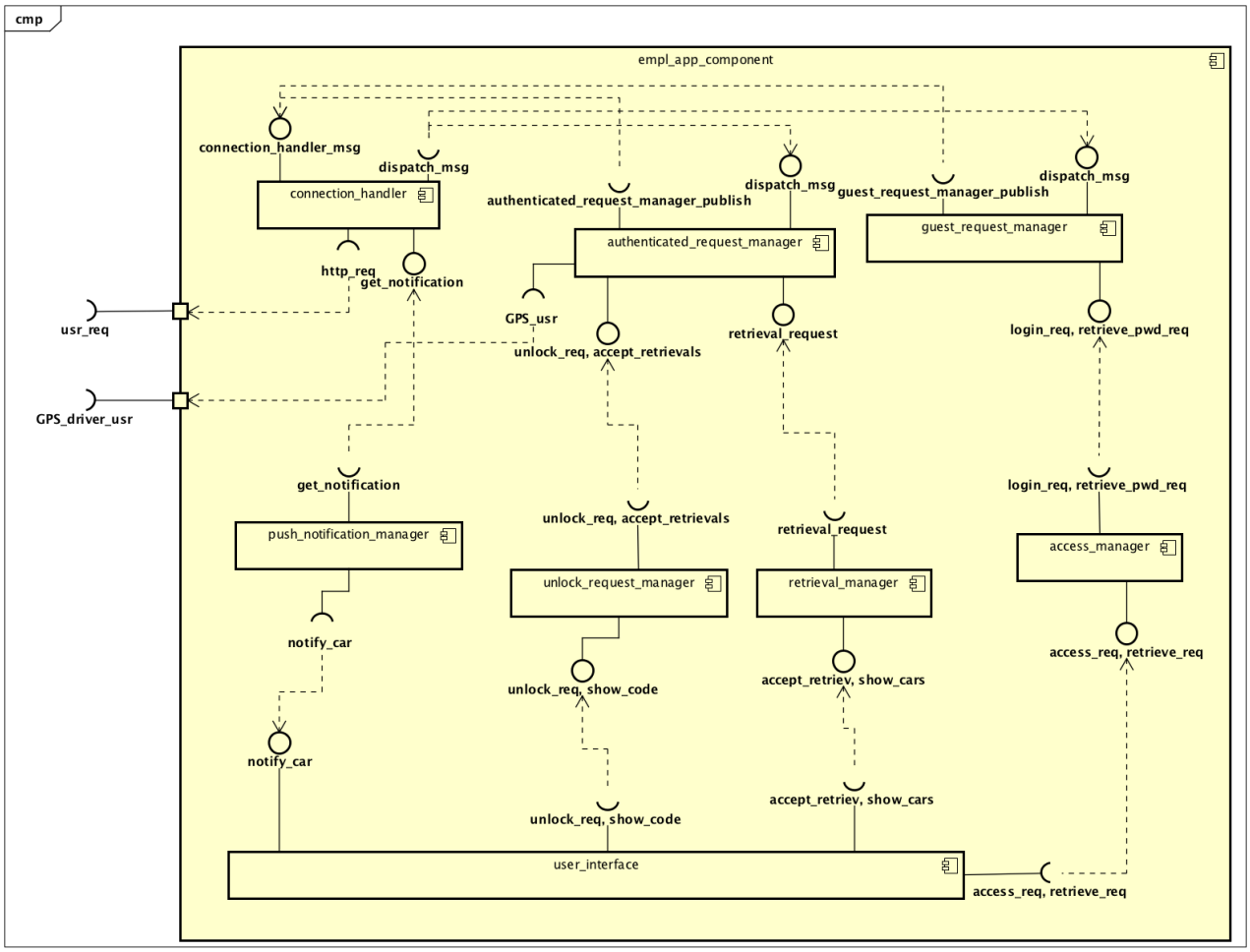

Fig. 4: employee application component

NOTE: only the component which are absent or serve a different purpose from the ones in the previous diagram are listed

- access_manager: in this case the component only generates login requests

- push_notification_service: generates notifications when the server sends a message stating that a car needs to be retrieved

- retrieval_manager: generates requests to accept a car retrieval task

System

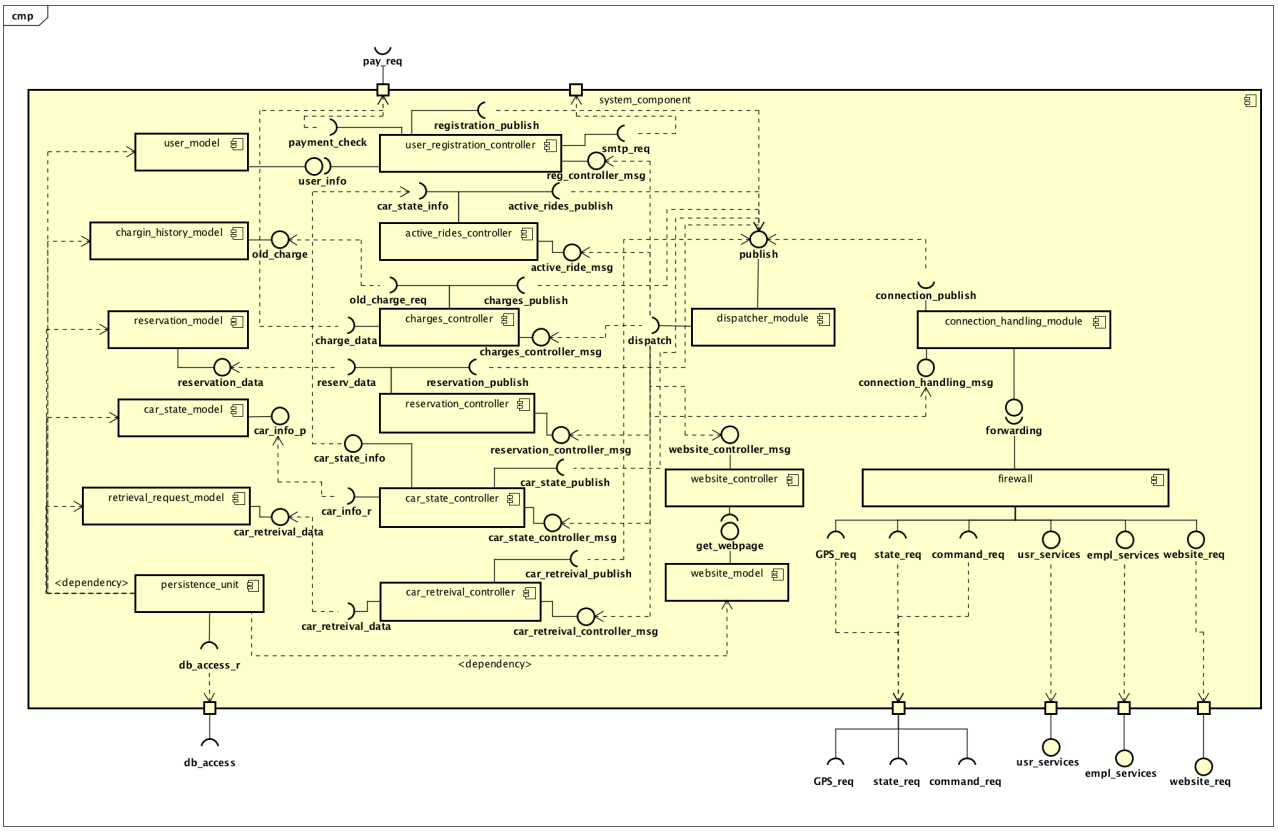

Fig. 5: system component

- firewall: security module filtering incoming requests

- connection-handling_module: manages all the connections with the cars, the users, and the employees, also manages the communication encryption

- dispatcher_module: filters and forwards the incoming messages to the relative subjects of interest

- user_registration_controller: authorizes and operates modifications to the user's model

- active_rides_controller: allows other component to obtain information on the rides currently active

- charges_controller: authorizes and operates modifications to the charging history model

- reservations_controller: authorizes and operates modifications to the reservation model

- cars_state_controller: authorizes and operates modifications to the cars state model

- car_retrieval_controller: authorizes and operates modifications to the retrieval requests model

- website_controller: grants access to the website's model

- user_model: the model for all the data relative to the users

- charges_history_model: the model for the data relative to the history of charges to the users requested to the external payment service

- reservations-model: the model for all the car reservations

- cars_state_model: the model for the information relative to the state of a car

- retrieval_request_model: the model for a car's retrieval request

- website_model: the model for the website

- persistence_unit: the component which manages the interface with the database and ensures that all the necessary data is retrieved from or sent to the database

It is important to note that the process of selection and arrangement of the components has highlighted some critical aspects of the non-functional requirements of the system, and some architectural consideration can be based upon them:

- the user's application as well as the employees' application does not need to store any persistent data, and can be basically reduced to a presentation layer and some control logic

- the system needs to handle a number of heterogeneous communication channels, which can potentially represent a bottleneck for the system's performance

- in the software architecture of the system's server the data model and the control logic can be identified and decoupled

2.3. HIGH-LEVEL SYSTEM ARCHITECTURE

2.3.1. SOFTWARE ARCHITECTURAL PATTERN

Based on the components individuated to carry out all the tasks required of the system, the architectural patter most suited for the system is the event-based (or pub-sub) model. The main reason for this choice is the need for the system to communicate with different software components, such as cars, the users and the employees' applications, and receive and send messages related to different topics. One main objection to the adoption of this pattern may be that there are not so many one-to-many event-driven communications, but we can for example point to the notification system for car retrieval and the system requesting the car's position to allow a user to reserve a car as the two main examples.

2.3.2. SYSTEM ARCHITECTURE

The system architecture naturally follows from the functionality of the components previously described. The result is a three-tier architecture, where the presentation layer is located on the mobile applications and the car, the business logic is almost entirely on the server's side (although both the mobile apps and the software systems on the car contain some control logic, it is mostly used to formulate requests for the server to evaluate, or to pass on messages to act upon), and the persistency layer comes in the form of a database component.

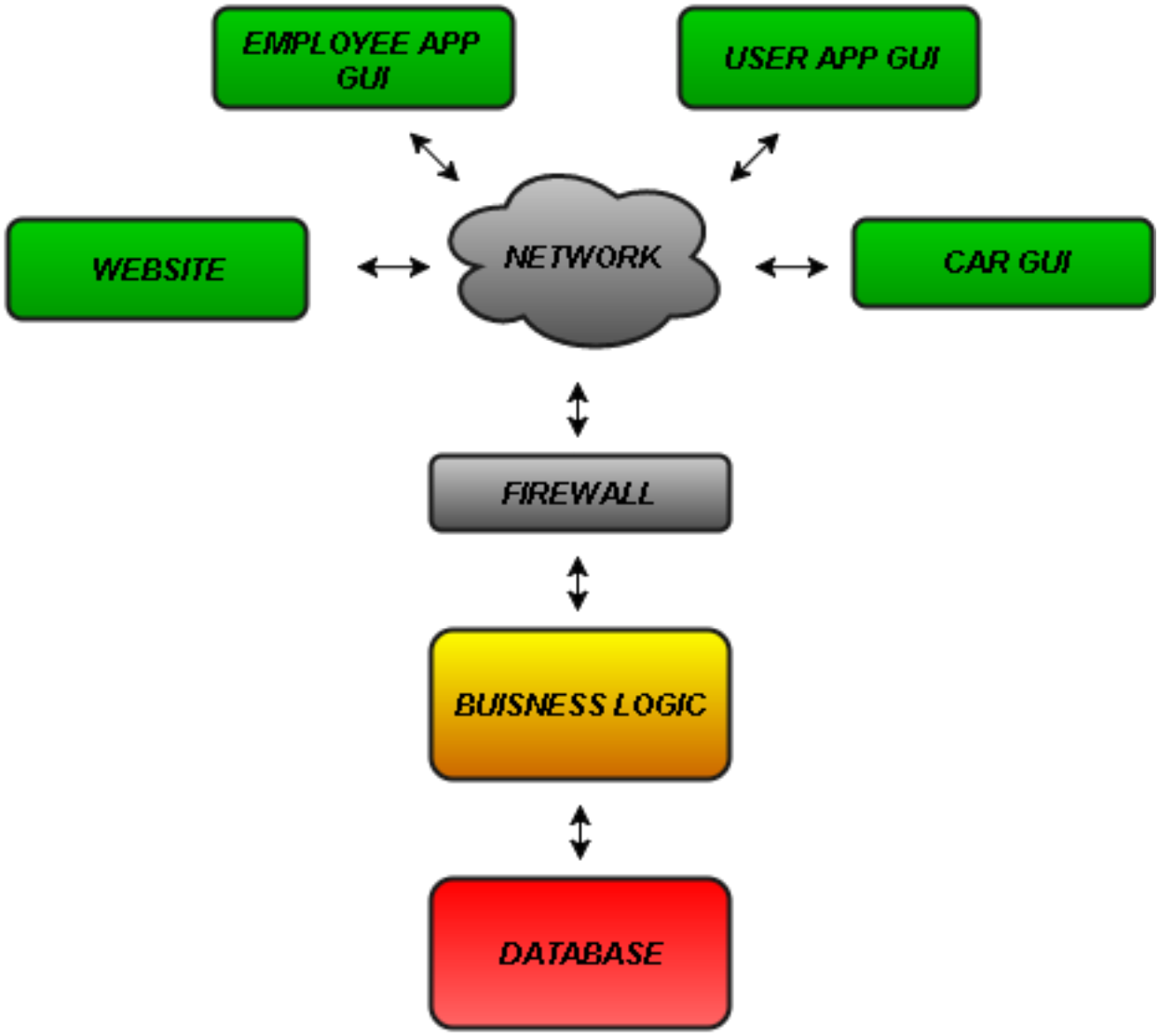

Fig. 6: high level system architecture

2.4. COMPONENTS INTERACTION

2.4.1. COMPONENTS INTERFACES

The following diagrams specify the interfaces of some of the main business-logic components of the system. The components are represented as classes of a UML class diagram, where each component has public methods representing the interfaces it offers to other components, and private attributes representing all the components which can be accessed by that specific component.

RequestManager Interface

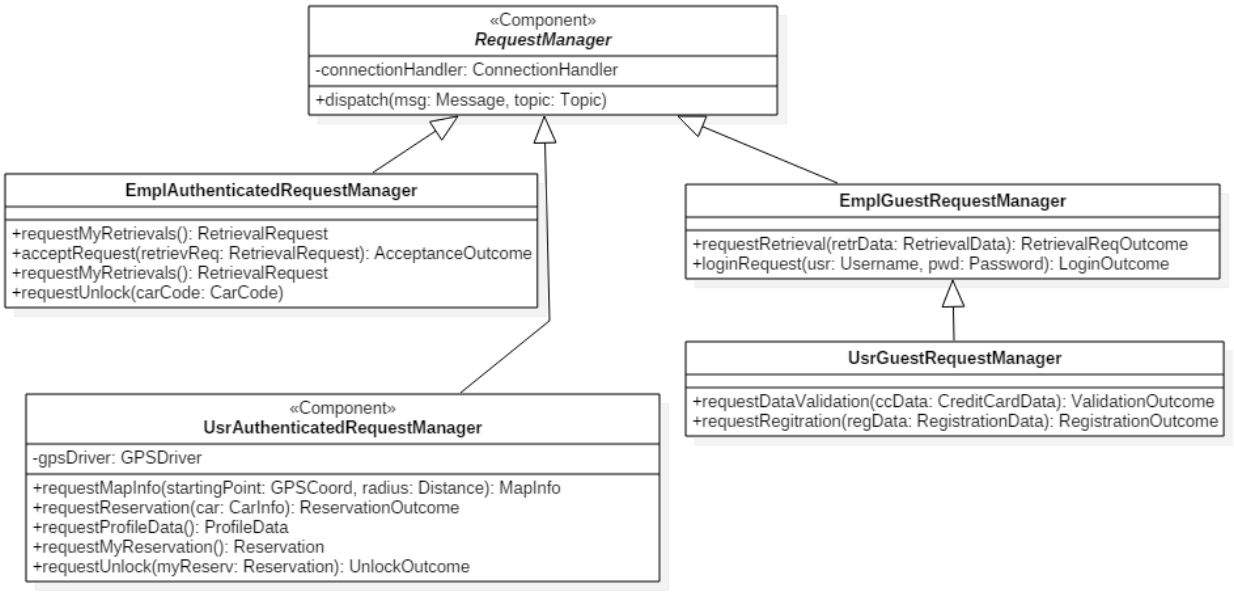

Fig. 7: request manager interfaces

A RequestManager is a client-side component which receives messages and sends requests to the server through the ConnectionHandler component. The dispatch(msg, topic) method is called by the connection handler to deliver a message from the server, while the request manager calls the publish(msg, topic) method offered by the ConnectionHandler. This serves the purpose of implementing a messaging system where the connection-related details remain not visible to the more functionality-oriented components, in this case the RequestManager.

Depending on the specific purpose, different implementation of the RequestManager accept different types of requests to deliver.

Dispatcher Interface

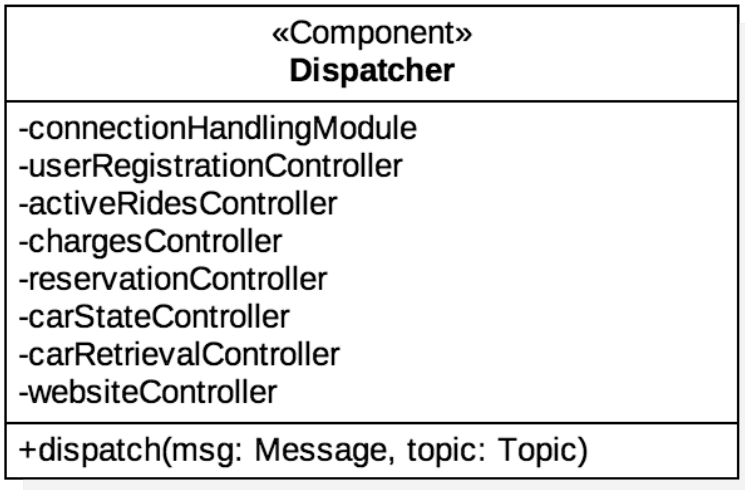

Fig. 8: dispatcher interface

This server-side component is the core of the messaging system. A publisher calls it to publish a message, and the Dispatcher takes care of delivering the message to all the subscribers to the specific topic of the message.

Controller Interface

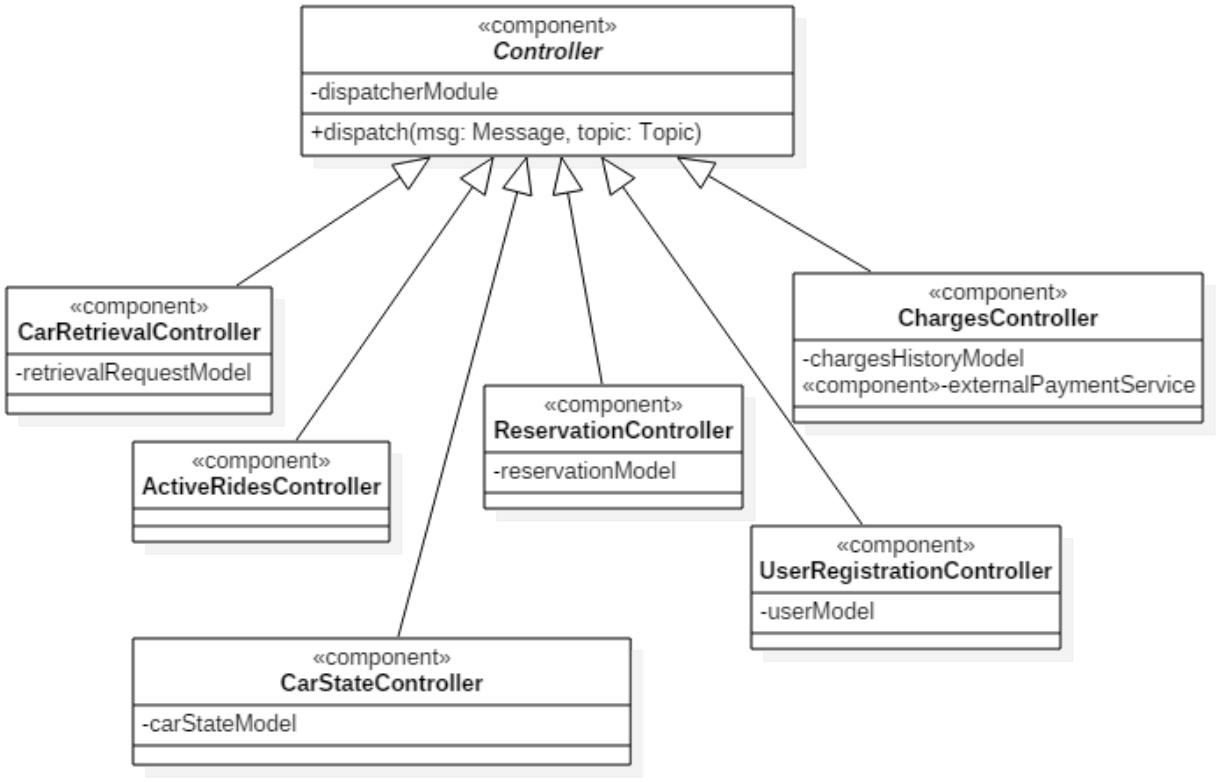

Fig. 9: controller interface

A Controller is the component which implements the core of the business logic. It connects the classes which contain the data of the system with the messaging infrastructure, processes each incoming message and retrieves the necessary data to perform various actions, usually resulting in another message being sent to the appropriate recipients. For these reasons a Controller has access to the Dispatcher, and also to the data model components necessary to perform its designated tasks.

DataModel Interface

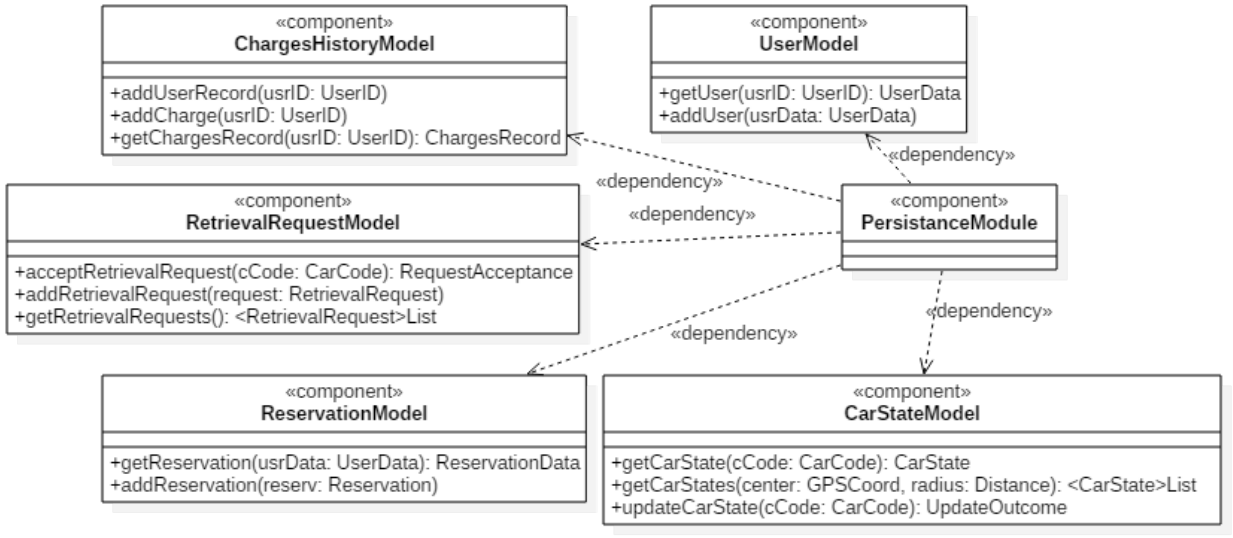

Fig. 10: data model interface

A data model component initializes and manages the run-time classes which store the system data in main memory, providing a series of methods for other component to access such data. It also interacts with the PersistenceModule, in order to access the database and retrieve the necessary data not yet stored in memory.

2.4.2. RUNTIME SEQUENCE DIAGRAMS

In this section a series of sequence diagrams show how the main components of the system interact with one another to realize some of the required functionalities.

Retrieval acceptance procedure

Describes the acceptance of a retrieval request from an employee, from the point of view of the server software.

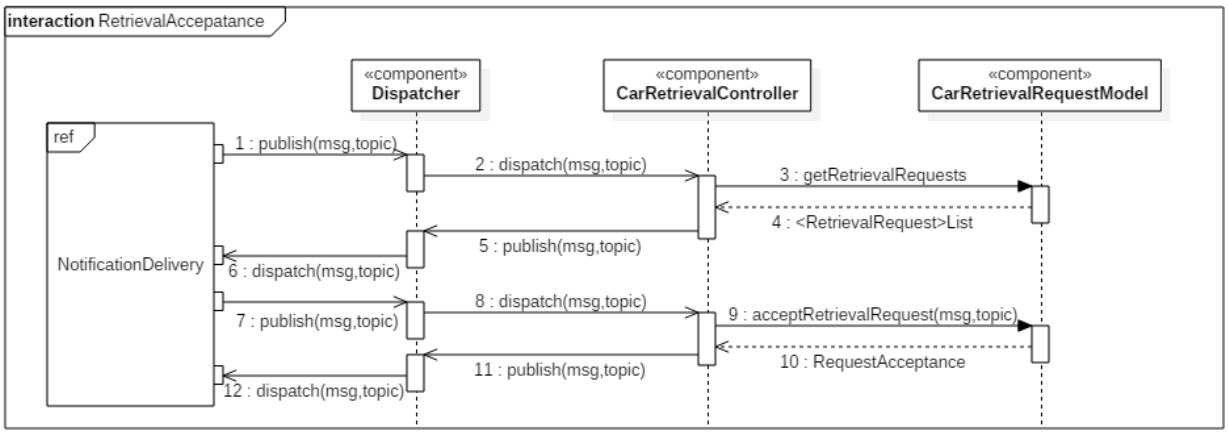

Fig. 11: retrieval acceptance procedure in the system component

Notification delivery procedure

Describes how a notification is sent to the employee's phone and he/she can accept the corresponding retrieval request.

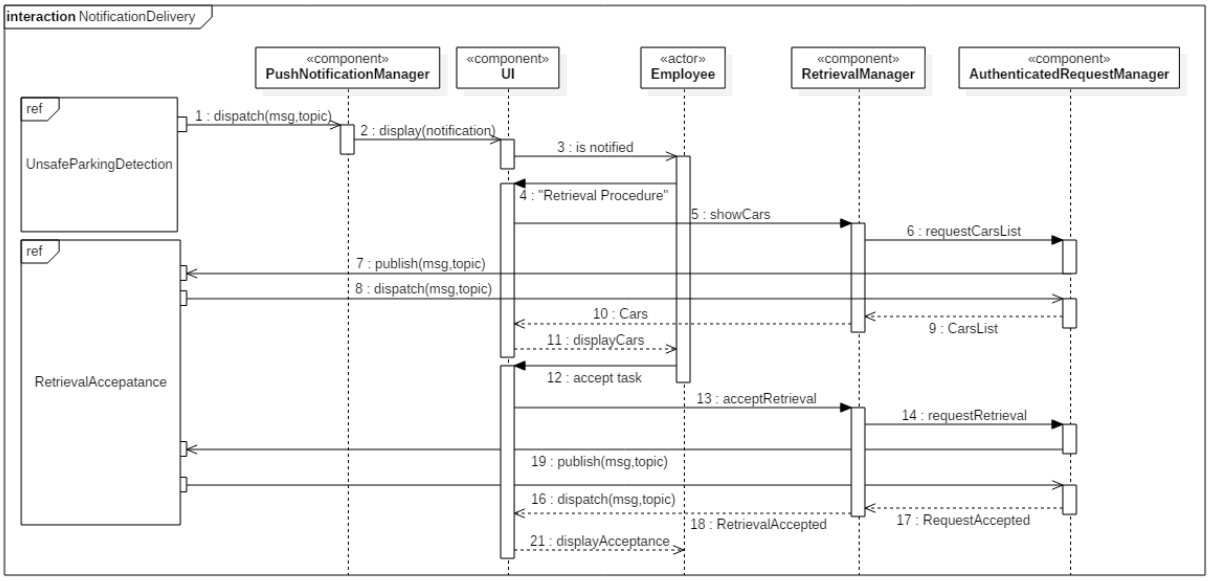

Fig. 12: notification delivery procedure in the employee application

Unlocking user procedure

Describes the interactions of the components in the client software on a user's phone when the user unlocks a car he/she has previously reserved.

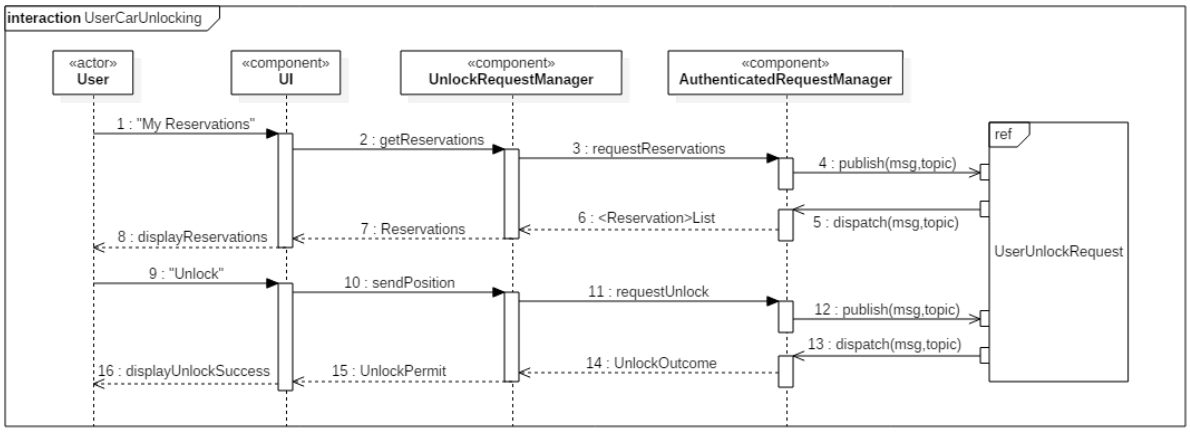

Fig. 13: unlocking user procedure in the user application

Unsafe parking detection procedure

Describes how, after receiving an update on the state of a car parked in a non-safe area, the system sends a notification to the employees' phones.

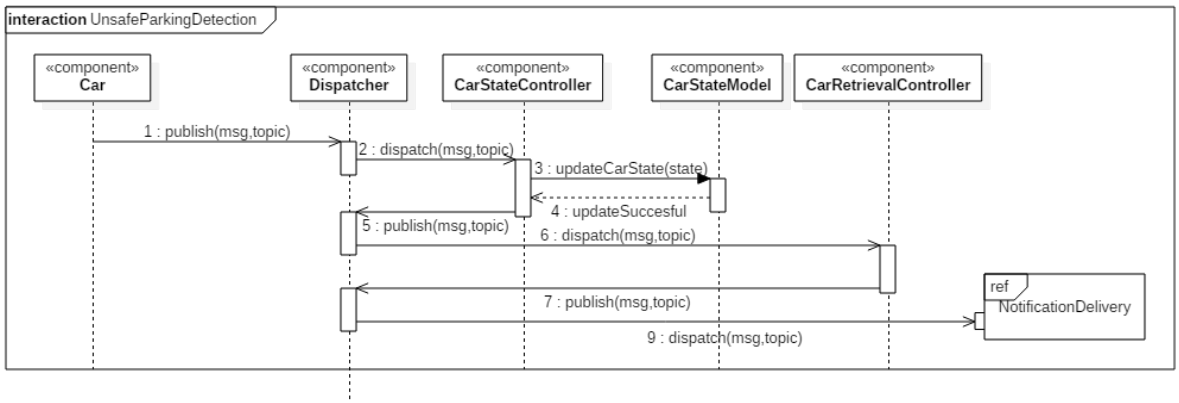

Fig. 14: unsafe parking detection procedure in the system

Reservation procedure

Describes the interaction between the request manager on a user's phone and the server-side software components during the reservation of a car.

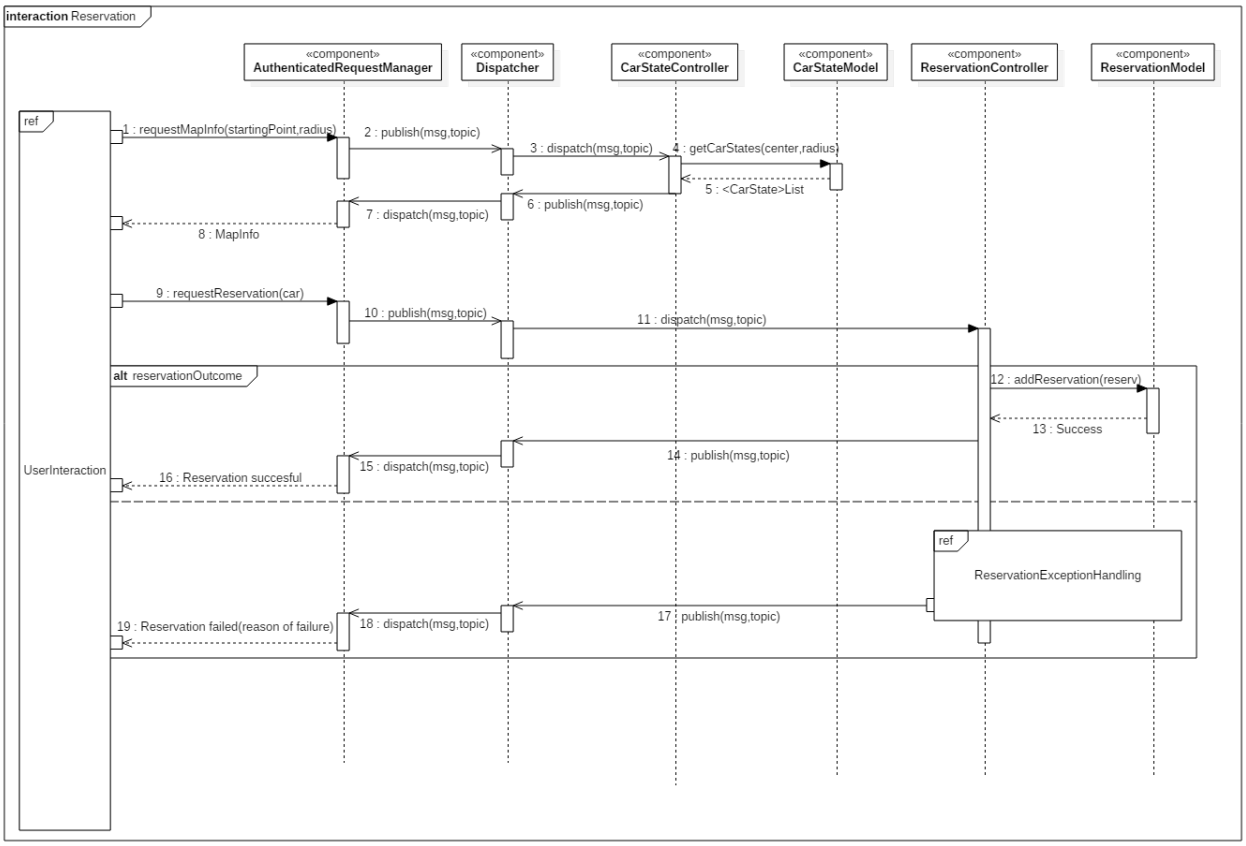

Fig. 15: reservation procedure in both the system and user application

2.5. DATA MODEL

This Entity-Relation diagram displays the main structure of the database for the application.

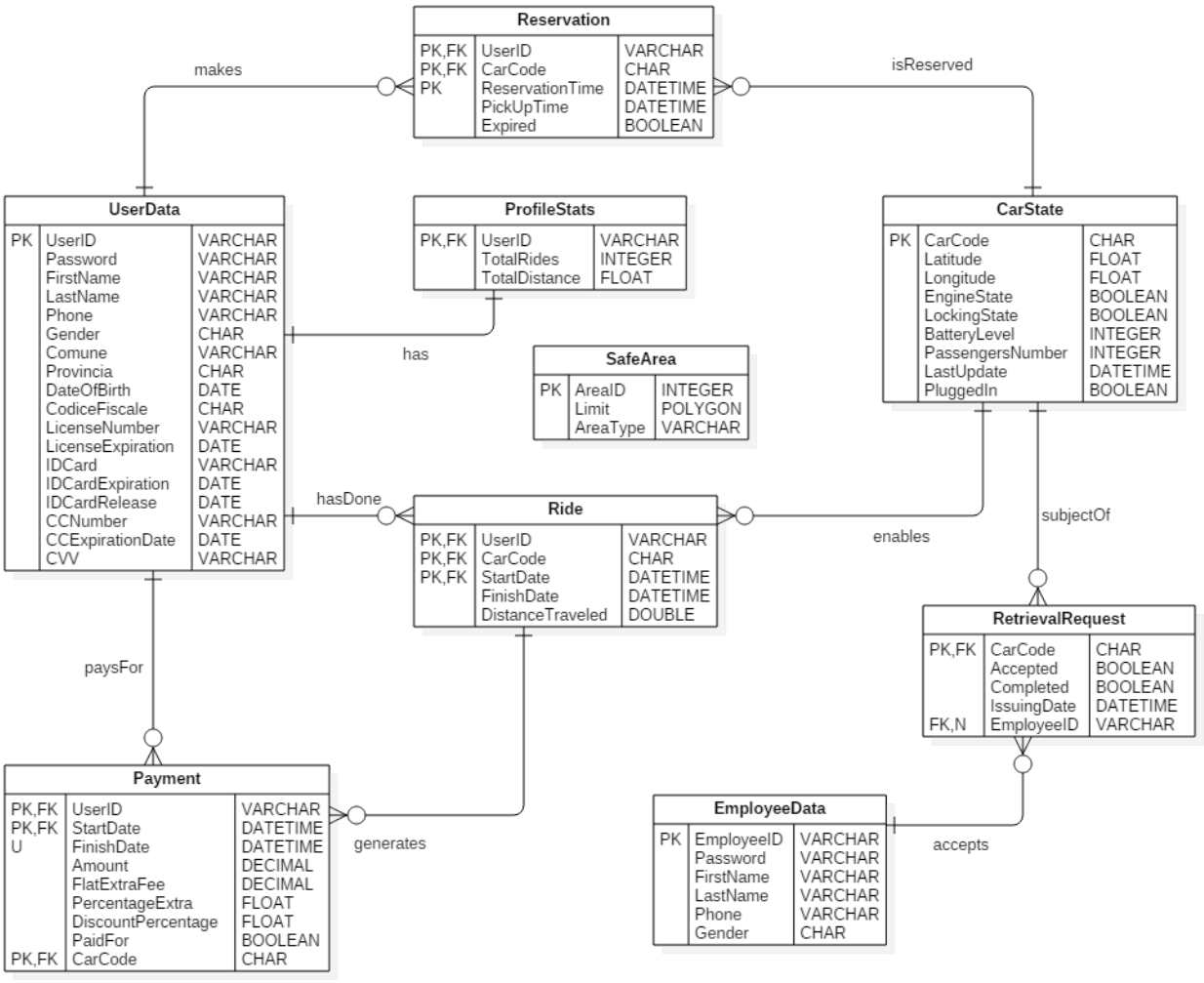

Fig. 16: E-R model diagram of the database

2.6. USER INTERFACE DESIGN

User experience diagram

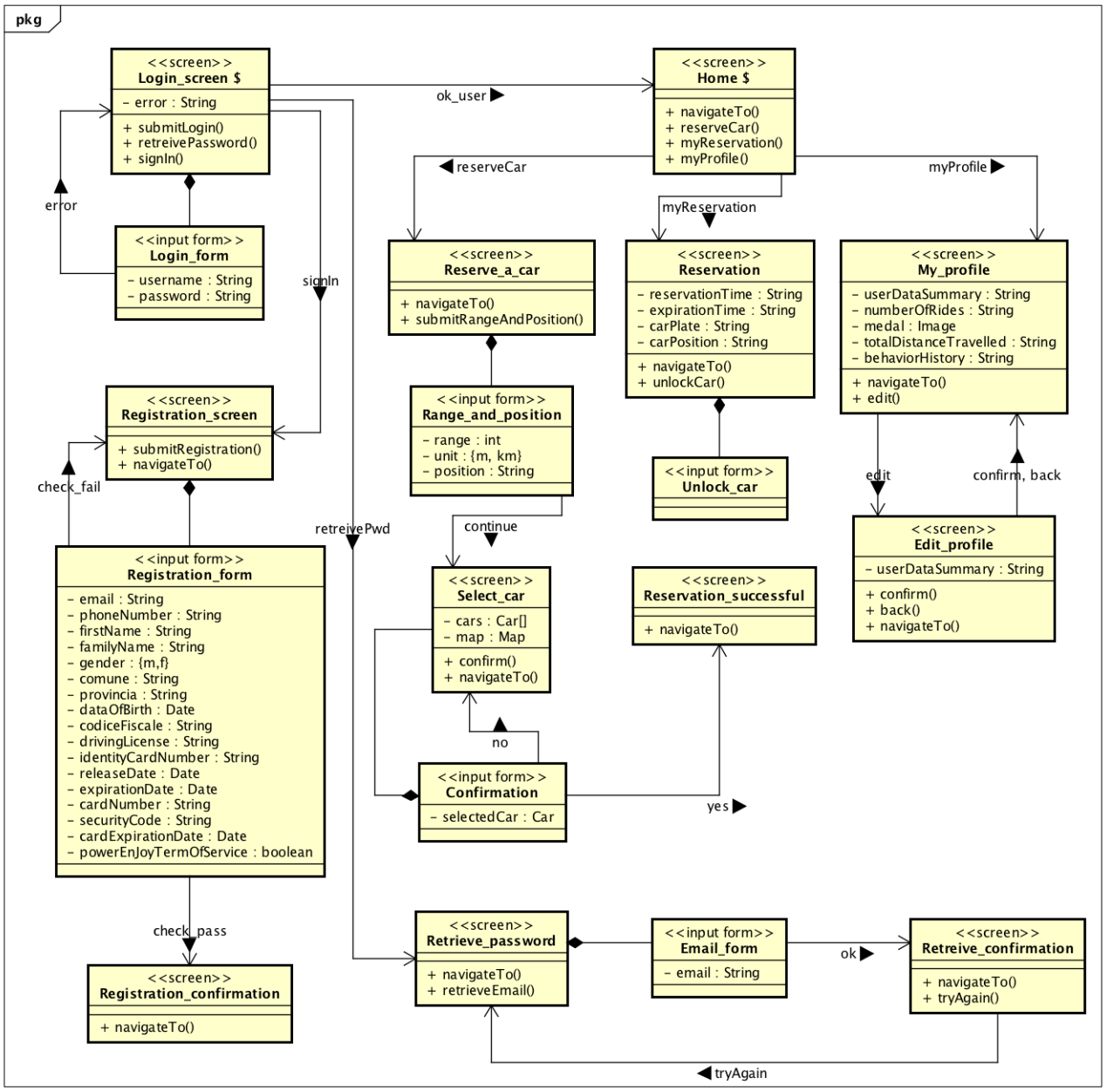

Fig. 17: UX diagram of user interface

This user experience diagram has the aim of describing in details the behavior of the user interface. There are two kinds of boxes: the "*screen*" and the "input form".

The former represents a visual screen and shows static and dynamic information. It contains some "attributes" and "methods". An attribute represents a dynamic information that the screen is able to show. Dynamic information is text or image that is not fixed and usually is different for each user, such as the *reservationTime* in the *Reservation* screen. A method represents an action that the user can perform from the from, such as moving to another screen.

The latter represents in details which are the information that the user can insert in a screen.

One screen is connected to another if from the first one through an action is possible to reach the second one.

2.7. SELECTED TOOLS

Operating systems

SUSE Linux Enterprise 10: is the operating system running on the server machines

Microsoft Windows Embedded Automotive: is the operating system that the provided cars come with.

Windows, iOS and Android are identified as the operating systems for which the mobile application will be developed.

Application server

Payara: is an open-source application server derived from Glassfish, and although it's not Java EE certified it is effectively Java EE 7 compliant and offers far more stable patch releases, security fixes, production support and developer support. It also has a very responsive community.

Database

MySQL: among all the available DBMSs, MySQL stands out for its scalability and flexibility, which also comes along with good performances and availability. In addition, it is open source, just like Payara, and can run on multiple platforms. All these reasons contribute to make MySQL our DBMS of choice.

Frameworks

J2EE: is a solid framework which will be used to ease the development of the application logic and the presentation layer for both the website and the apps on the server side.

Windows Automotive Application Framework: is chosen as a native framework that our windows developers are familiar with.

Communication

On the server side JMS is used as the messaging API, and the messages are exchanged in a text-based XML format via SOAP Transport Protocol.

On the client side Kaazing WebSocket Gateway APIs are used to support JMS messaging.

Ides

The selected IDEs are NetBeans for the development of the server-side application, VisualStudio for the development of the software running on the cars, and AndroidStudio, XCode and VisualStudio are the integrated environments selected to develop for Android, iOS and Windows respectively.

2.8. DEPLOYMENT VIEW

2.8.1. SOFTWARE COMPONENTS MAPPING

The deployment of the software components follows naturally from the previous design decisions: the native application clients will be installed on the clients and employee's phones, according to the phone's OS. For the application server a device of the IBM Power System series has been selected, which provides good flexibility and scalability potential, as well as reliability and is in general a solid choice for medium-sized enterprise applications. The application server software modules are organized in an .ear archive to guarantee a coherent deployment. Another .war file has been created to separate the software components which realize the website service, in case this module needs to be moved to another machine in the future.

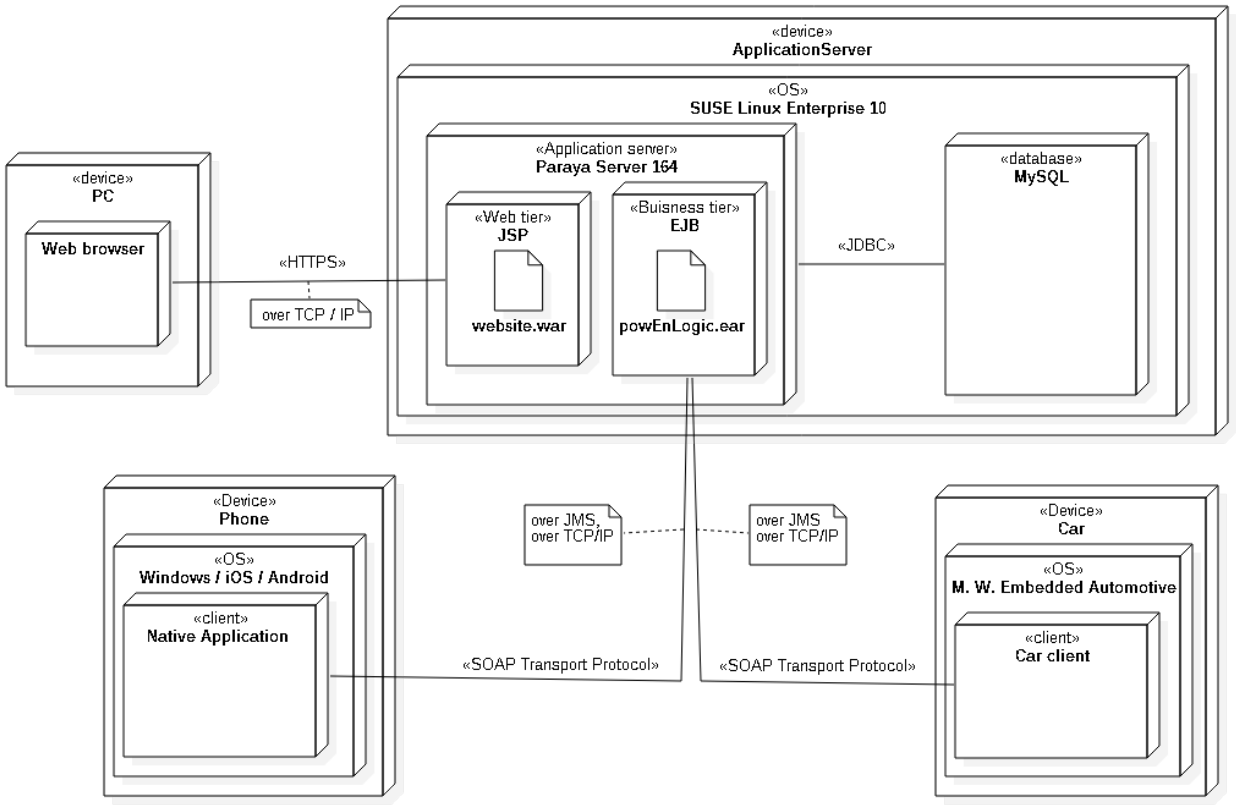

Fig. 18: deployment diagram

2.8.2 TECHNICAL ENVIRONMENT REQUIREMENTS

No particular conditions need to be met to allow the system to function properly, barring some basic needs such as a suitable physical space to collocate the application server and a good enterprise Ethernet connection.

2.9. ALGORITHM DESIGN

In this section the algorithmic steps to either accept or reject a reservation from a user are displayed in objective oriented pseudo-code.

```
/**
 * This class controls whether the reservation is well-formed.
 * A reservation is well-formed if:
 *    1) the user hasn't already an active reservation, and
 *    2) the user hasn't some pending payment, and
 *    3) the user's credit card and identity card aren't expired,
 *    4) the selected car is actually available
 */
class ReservationController implements Publish, Subscriber{
    attribute ReservationModel reservationModel

    method void checkAndAddReservation(Reservation reservation) {
        try{
            doubleReservationCheck(reservation);
            lastPaymentCheck(reservation);
            validityUserDataCheck(reservation);
            carAvailabilityCheck(reservation);

            reservationModel.addReservation(reservation);

        } catch(DoubleCheckReservation or
                PendingPaymentException or
                InvalidUserDataException or
                NotAvailableCarException){

               throw ReservationException
        }
    }

    method void doubleReservationCheck(Reservation reservation) {
        user = reservation.getUser();
        reservationToCheck =
               reservationModel.getNotExpiredReservation(user);

        if (reservationToCheck != null)
            throw DoubleCheckReservation
    }
```

```
    method void lastPaymentCheck(Reservation reservation){
        user = reservation.getUser();
        payment = user.getLastPayment();

        if (payment.isPending())
            throw PendingPaymentException
    }

    method void validityUserDataCheck(Reservation reservation){
        userData = reservation.getUser().getUserData();

        if (not userData.areValid())
            throw InvalidUserDataException
    }

    method void carAvailabilityCheck(Reservation reservation){
        carState = reservation.getCar().getCarState();

        if (not carState.isAvailable())
            throw NotAvailableCarException
    }
}
```

2.10. REQUIREMENTS TRACEABILITY

In this section for each functional requirement are listed the components which achieve its fulfillment. The connection handling components, the firewall and the dispatcher are always omitted, since they do not serve any specific functionality, despite being involved in all the communications between the server and the different clients and devices.

[R1.1]: The app is available for any person to download and run on his/her phone

This requirement will be achieved at a later stage.

[R1.2]: From the home page of the app any person can carry out the registration procedure

[R1.3]: The registration procedure requires a person's credentials and payment info to be carried out

[R1.4]: The registration procedure uses the external payment service to verify the validity of the provided payment info

[R1.5]: At the end of the registration procedure the person whose credentials were used is registered in the system

[R1.6]: At the end of the registration procedure the person receives an e-mail containing a password which he/she can use to access the system

[R1.7]: At the end of the registration procedure the person can ask the system to send another mail

COMPONENTS: UserInterface, RegistrationManager, GuestRequestManager, UserRegistrationController, UserModel

[RA1.1]: The app allows any person to log in by providing a valid e-mail and password

[RA1.2]: The app does not allow any person who does not provide a valid e-mail and password to log in

[RA2.1]: From the home page of the app the password retrieval procedure can be initiated by any person

[RA2.2]: If a person provides a valid e-mail address during the password retrieval procedure the system sends an e-mail to that address containing the associated password

COMPONENTS: UserInterface, AccessManager, GuestRequestManager, UserRegistrationController, UserModel

[RA3]: Access to the PowerEnJoy's website (a static page) is granted upon request by the system (no login required)

COMPONENTS: WebsiteController, WebsiteModel

[R2.1]: The "Reserve a car" function can be accessed by the user from the home page of the app

[R2.2]: The "Reserve a car" function allows the user to select a range (distance)

[R2.3]: The system acquires the user's current position through the GPS coordinates of the user's phone

[R2.4]: The system tracks all available cars' current position through their GPS coordinates

[R2.4.1]: The cars must possess a device which can be tracked via GPS

[R2.5]: The "Reserve a car" function allows the user to select a starting position for the search, which can be either their current location or a given address

[R2.6]: When the user confirms the inserted parameters the search is carried out and the "Reserve a car" function displays to the user the data of the search acquired from the system in a Google provided map

[R3.1]: The app allows the user to tap on any available car on the map displayed as the result of a search conducted through the "Reserve a car" function.

[R3.2]: When a user taps on a car the app generates a pop-up asking the user if he/she wants to confirm the reservation.

[R3.3]: As long as the car was not reserved by another user in the meantime, when the user confirms the car is marked as reserved by the system and the user can see the "Reservation successful!" message on the app.

[R3.4]: When the system marks a car as reserved any reservation request from any user is rejected by the system while the car is in the reserved state.

[R3.5]: A car is in reserved state for one hour from the moment it was marked as reserved.

[R3.6]: A car in reserved state is not signaled by the system during the "Reserve a car" procedure.

[R3.7]: After one hour from its reservation a car is no longer in reserved state.

[R3.8]: A car not in reserved state is considered available by the system only if it is parked in a safe area less than 3 km away from a power grid station and has more than 20% of its battery.

COMPONENTS: UserInterface, ReservationRequestManager, MapManager, AuthenticatedRequestManager, ReservationController, ReservationModel, CarStateController, CarStateModel,

+ StateWrapper and the controllers on the car providing status updates.

[R4.1]: One hour after a car has been reserved if it was never ignited the system charges for 1 EUR the user who reserved it

COMPONENTS: ReservationModel, ReservationController, ChargesController

[R5.1]: From the home page of the app the user can access the "My reservations" section

[R5.2]: In the "My reservations" section if the user has reserved a car less than an hour ago an active reservation is displayed with an "Unlock" button

[R5.3]: If the user is less than 10 meters away from the car and presses the unlock button the car unlocks

COMPONENTS: UserInterface, UnlockRequestManager, AuthenticatedRequestManager, ReservationController, ReservationManager, CarStateController, CarStateModel, CommandDispatcher, LockingCommand

[R6.1]: When a car is ignited the system starts charging the last user who reserved the car

[R6.2]: When the charging starts, the display on the car shows a "Current charge" field with a number representing the current total charge, which starts from 0

[R6.3]: Once a minute the "Current charge" value is incremented by a set amount

COMPONENTS: EngineController, Terminal, StateWrapper, ChargesController, ChargesHistoryModel, ChargingState

[R7.1]: When a car is stopped and the sensors in the car detect no one inside, if a user was being charged for the car the system stops charging him/her.

[R7.2]: One minute after a car has been stopped and the sensors in the car detect no one inside, the system locks the car.

COMPONENTS: PassengerSensorsController, LockingStateController, StateWrapper, ChargesController, ChargesHistoryModel

[R8]: The moment the car is stopped, if the sensor in the car detected two passengers the system records it as a possible discount of 10%

[R9]: The moment the car is stopped and the sensors in the car detect no one inside, if the car has more than 50% of its maximum battery the system records it as a possible discount of 20%.

[R10]: If before 2 minutes since the moment the car has been stopped and its sensors detected no one was inside the car is plugged in a power grid and its position is within a special parking

[R11.0]: The moment the car is stopped and the sensors in the car detect no one inside, if the safe area nearest to the car is more than 3km away from it, the system records an extra fee of 30%

[R11.1]: The moment the car is stopped and the sensors in the car detect no one inside, if the car has less than 20% of its maximum battery, the system records an extra fee of 30%.

[R12]: After two minutes since the car has been stopped and its sensors detected no one was inside, the system applies all the extra fees and if there are none it applies the highest discount among the possible ones to the cost of the ride.

COMPONENTS: GPS_Controller, PassengerSensorsController, BatteryLevelController, EngineController, StateWrapper, ChargesController, ChargingHistoryModel

[RA4]: If a user with a pending payment procedure tries to reserve a car, a pop-up lets the user know that he/she needs to pay for his/her last ride to be able to reserve a car and the app does not allow the user complete the reservation procedure.

COMPONENTS: ChargesController, ChargesHistoryModel + components for a reservation

[RA5]: If in the user's profile either the credit card or identity card expiration date has already passed, when the user tries to reserve a car a pop-up lets him/her know that the data in the user's profile need to be updated and the app prevents the reservation procedure from being completed

COMPONENTS: UserRegistrationController, UserModel + components for a reservation

[RA6.1]: From the home page of the app the user can access the "My profile" section

[RA6.2]: From the "My profile" section the user can use the "Edit profile" button to modify his/her credential and payment info

[RA6.3]: The system can check via the external payment service whether the payment info inserted are valid

[RA6.4]: If the inserted payment info is valid the user can save the changes by tapping the "Confirm" button.

COMPONENTS: UserInterface, ProfileManager, AuthenticatedRequestManager, UserRegistrationController, UserModel

[RA7]: If the user has already a reservation which is not expired yet when he/she tries to reserve a car, a pop-up lets the user know that he/she cannot reserve a car and the app does not allow the user complete the reservation procedure

COMPONENTS: ReservationController, ReservationModel + other components for a reservation

[RA8]: When a car is locked the system checks its GPS coordinates, and if they correspond to those of a non-safe area the last user who reserved the car is charged for a set extra fee.

COMPONENTS: GPSController, CarStateController, CarStateModel,

[R13.1]: Each employee has access to an application, AdminPowerEnJoy, on their phone

[R13.2]: When a car is locked the system checks its GPS coordinates, and if they correspond to those of a non-safe area all employees are notified through AdminPowerEnJoy that the car needs to be retrieved

[R13.3]: AdminPowerEnJoy allows an employee to accept a retrieval request through the "Retrieval procedure" function

[R13.4]: If an employee has already accepted a retrieval request, the retrieval request can no longer be accepted

[R13.5]: After 12 hours, if an employee has accepted a retrieval request but has not retrieved the car, the request is issued again by the system and another employee can accept it

[R13.6]: When an employee is notified of a car to retrieve, the notification contains the information necessary to set up the navigator of the company's cars to find the position of the car to retrieve

[R13.7]: AdminPowerEnJoy allows an employee to unlock any car for which he/she has accepted a retrieval request

COMPONENTS: UserInterface, RetrievalManager, PushMotificationManager, UnlockrequestManager, AuthenticatedRequestManager, CarStateController, CarStateModel, CarRetrievalController, RetrievalRequestModel

[R14]: When the employee ignites a car which was opened through AdminPowerEnJoy the system does not initiate any charging procedure

COMPONENTS: UnlockRequestManager, AuthenticatedRequestManager, CarStateController, CarStateModel, CommandDispatcher, LockingCommand

3. EFFORT SPENT

Together: 2h + 2.5h + 3.5h +3h +3h +4h [18h]

Reppucci: 2h + 2h + 2,5h + 3h + 2h + 3h [14,5]

Peverelli: 2h +2h +5h +2h + 2h +3h +1h [16h]

= [48,5h]