DESIGN DOCUMENT

POLITECNICO
MILANO 1863

Figure 1: Politecnico di Milano

version 1.0

Artemiy Frolov, mat. 876373

autumn 2016

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to provide more technical details about the CarSharing System software application
This document is directed to developers and is necessary to state these aspects of the developing system:

- high level architecture
- runtime view
- choosed architectural styles and patterns
- algorithm design of key components
- possibly include some extensions of the user interface defined in RASD

1.2 Scope

CarSharing is a web-based software application that helps car-sharing companies to increase usability of their service, by providing more convenient way of renting electric cars for clients via smartphones, hence helping clients to use the service in a more comfortable way. Thus, the software is targed only to:

- Users

System allow clients(Users) to locate available electrical cars, with all relevant information about it (inluding current battery fulness, address, registered number) nearby or in the specific area.
After selecting the car, user can reserve it for up to one hour. When a user reaches the reserved car, system allows the user to unlock the car via button in the web-app. As soon as the engine ignites, the system confirms that the car is now occupied and user can see current charges through the screen in the car.
User can leave the car for a short period of time without missing the car occupation. When the user is no more needs the car, he presses the "Stop the trip" button, system locks the car and collect the money from the bank

account, provided by user during registration. From this point the car is no more controlled by the user no more and it be becomes available again. System, in order to restrain the behaviour of users, and to encourage virtuous behaviours of users, carries out some reward and punishment features. Also the system uses external web-services to present the location of cars and to manage payments.

1.3 Definitions, Acronyms, Abbreviations

- RASD: requirement analysis and specification document
- DD: design document
- SMS: short message service; used to notify users 15 minutes before the reservation time expires, also used for a short period suspendance warning. A SMS gateway is needed to use it.
- SMS gateway: it is a service which allows to send SMS via standard API.
- MVC: model view controller.
- URL: uniform resource locator

1.4 Reference Documents

- RASD produced before 1.1
- Specification Document: Assignments 1 and 2 (RASD and DD).pdf

1.5 Document Structure

- Introduction
  - Purpose
  - Scope
  - Definitions, Acronyms, Abbreviations
  - Reference Documents
  - Document Structure

- Architecture Design
  - Overview: High level components and their interaction
  - Component view
  - Deployment view
  - Runtime view: mostly contain sequence diagrams to describe the way components interact
  - Component interfaces
  - Selected architectural styles and patterns
  - Other design decisions
- Algorithm Design
- User Interface Design
- Requirement Tracebility
- Effort Spent
- Reference

2 Architectural Design

2.1 Overview

The CarSharing system would have 3 tier client-server architecture.

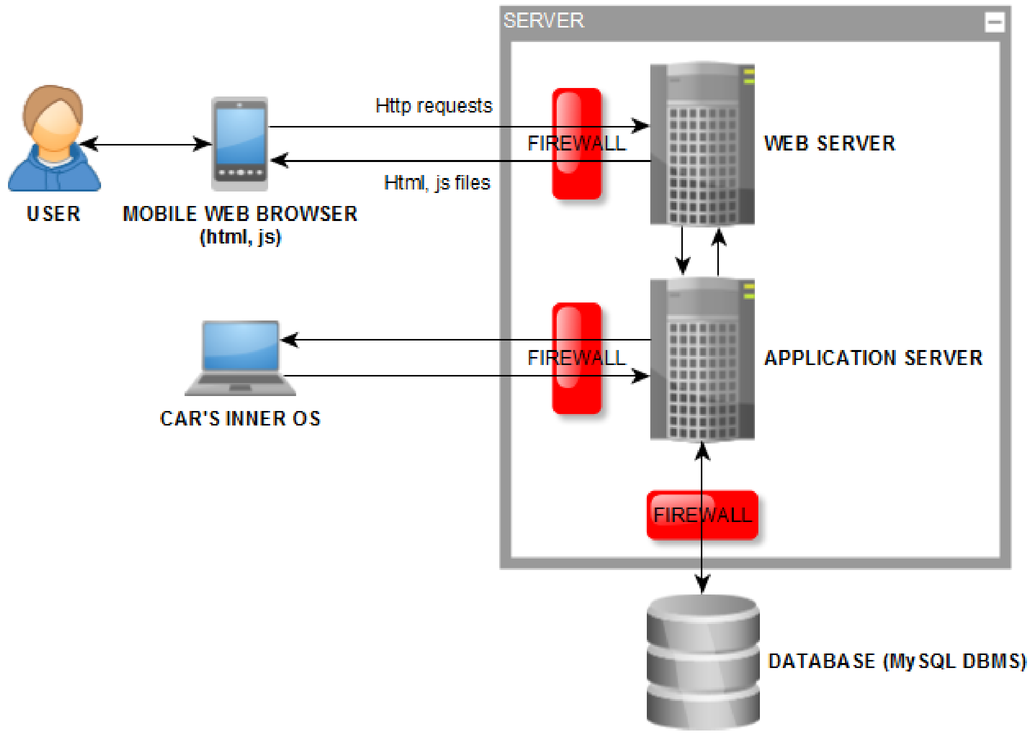


Figure 2: High-level architecture representation

Mobile web browser: browser that user uses on his/her smartphone.
Web server: Server that processes http requests.
Mobile web browser sends http requests to the web server, which responds by sending proper html and js files.
Mobile web browser and web server form the GUI and both correspond to the Presentation tier of the cliet-server high-level architecture.

Application server: server that processes the data and manage application logic.
Application server corresponds to the Logic tier of the cliet-server high-

level architecture.

Car's inner OS: operating system of the electrical car.
Database: Persistent data storage. Implemented as DBMS.
Database corresponds to the Data tier of the cliet-server high-level architecture.

Application server receives user's activity information from the web server via RESTful API. Received information is then processed and can be stored in database or/and affect electrical car states.
Application server communicates with the electrical car's inner OS via 3G.

Moreover each tier is strongly secured with firewalls to avoid illegal system breaches.

2.2 High level components and their interactions

2.3 Component view

- RequestController: manages the incomming http requests from users and redirect them to system services.
- RegistrationController: maintains collected registration information, incuding checking validity of inputed data.
- LogInController: checks if user and password exist in database.
- Reservationontroller: manages car reservation. The most important part of the system. It:
  - updates reservation status of the car in database.
  - includes backward timer that restricts the time of car reservation during "reserved" status and suspended car during "occupied" status. It provides time to NotificationController.
  - communicates with LocationController(and implicitly with Car) to activate page car control buttons in user's browser.
  - activates PaymentCalculator to sum up the cost of the ride.

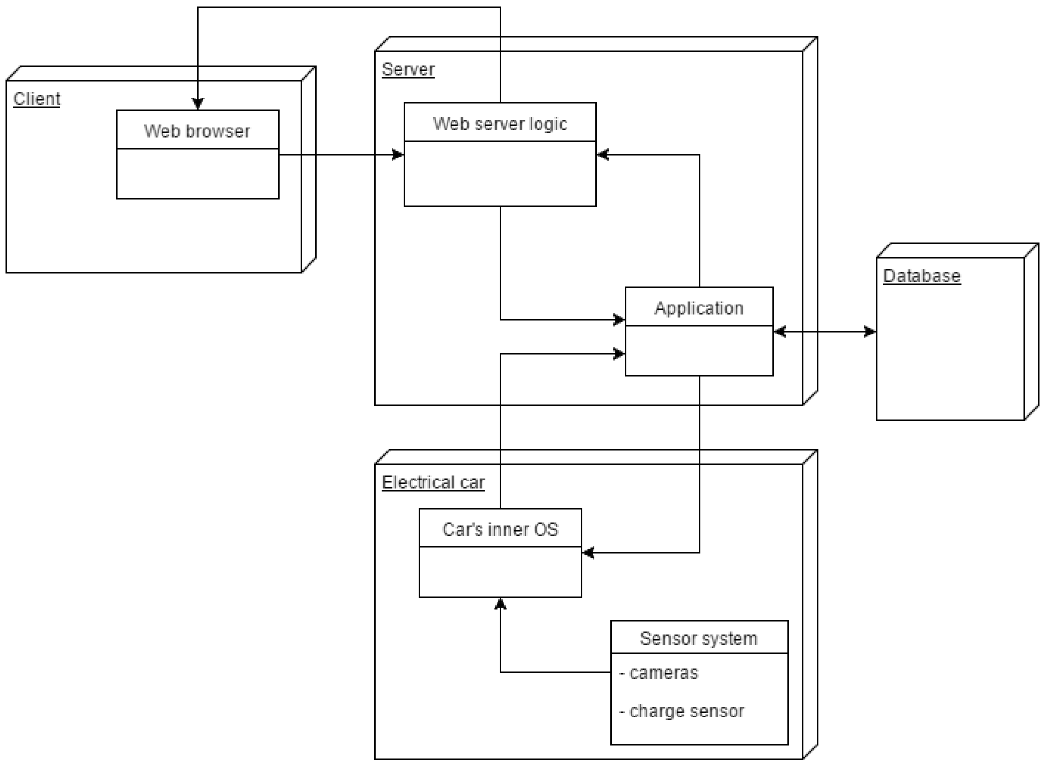


Figure 3: High-level components

- NotificationController: sends warning messages to the user about soon expire time of reservation or time of car being suspended via SMS. Has access to the database to acquire telephone number of the user.
- LocationController: compares user's and car's locations to enable car controlling features. Communicates with CarController(and implicitly with the Car) to acquire GPS data of the car. Have access to database to acquire general information about the car.
- CarController: Acquires information of the car from sensors, GPS and other relevant information for the CarSharing system. Sends request to the car for manipulation.
- PaymentCalculator: sum up the cost of the ride. Sends requests to the CarController to collect sensors information. Collected informa-

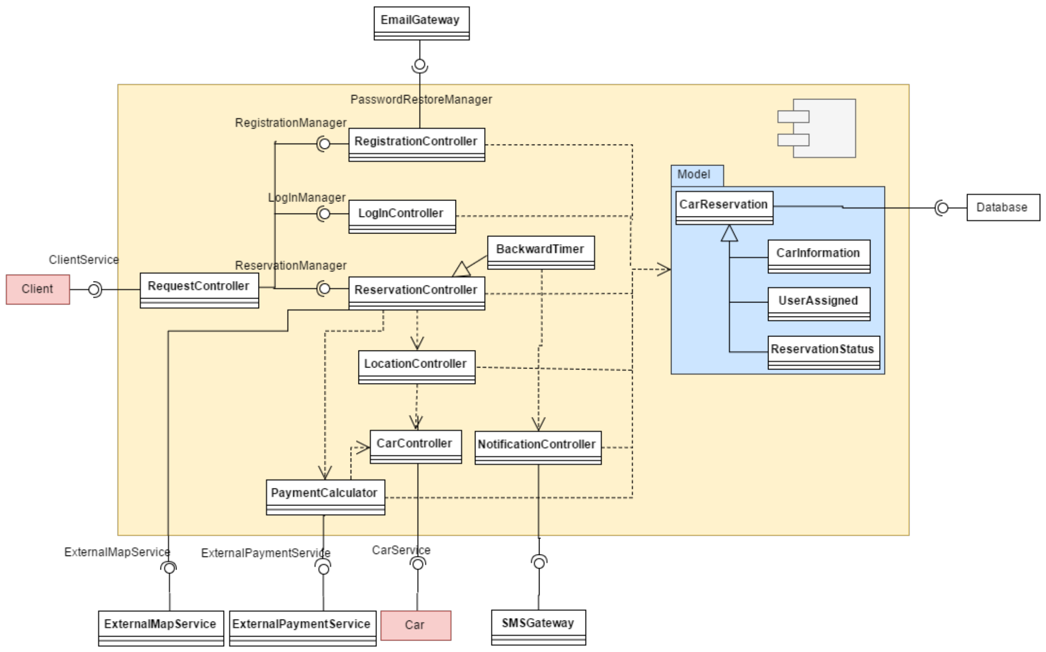


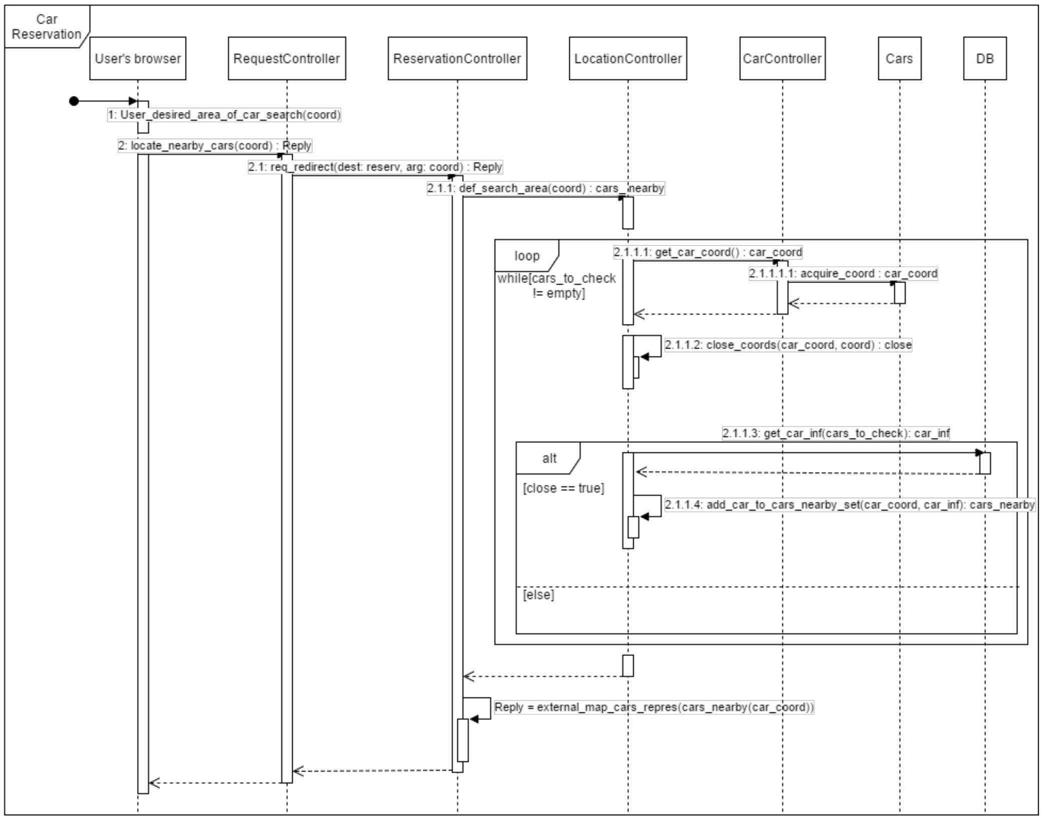
Figure 4: Components View

tion is then transformed into charges and discount of the ride. Has access to the database to collect payment information of the user. Uses external payment web services, sends requests to it with the amount of payment to take and address of the charged bank account.

2.3.1 Deployment View

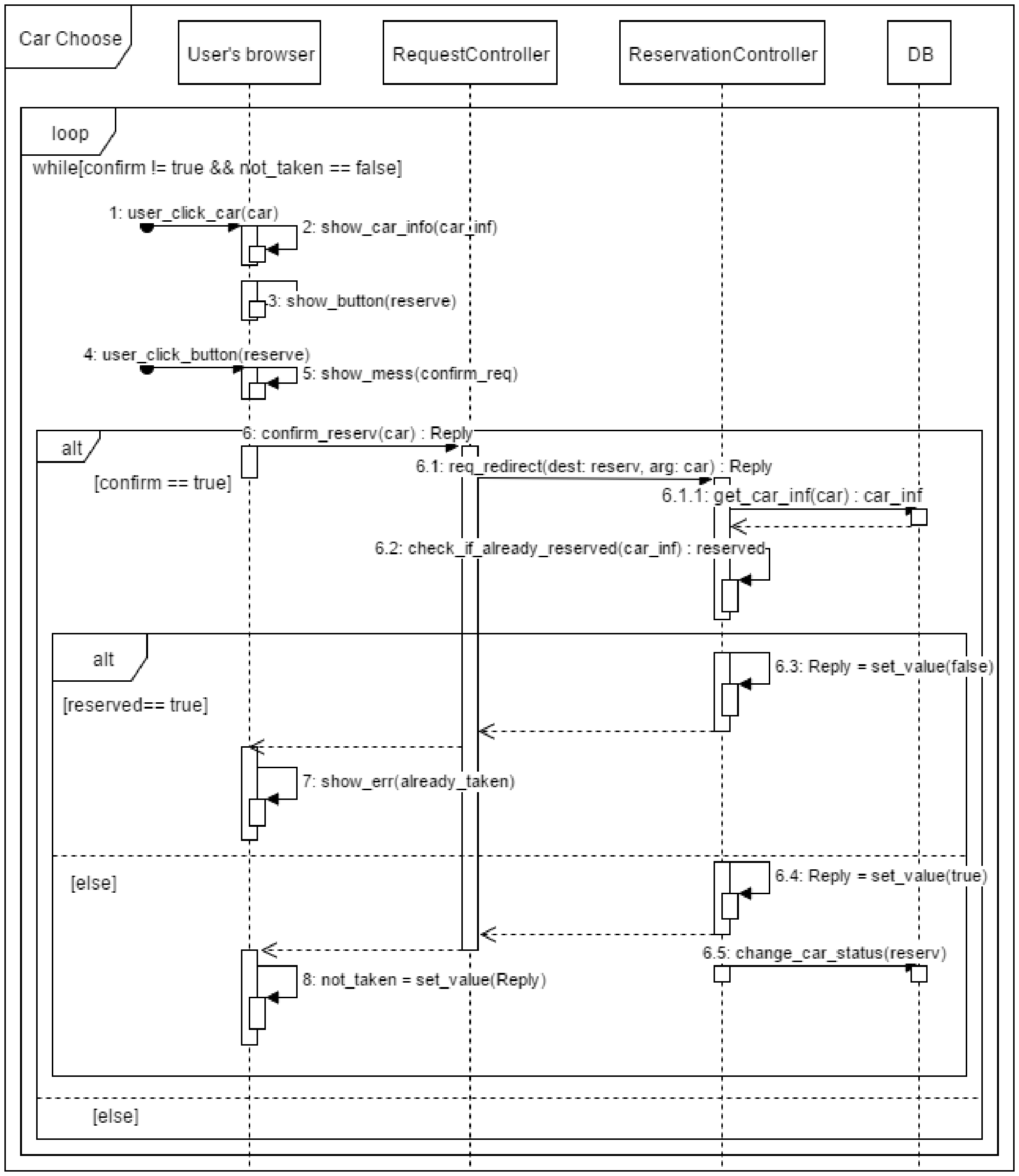
2.4 Runtime view

2.4.1 Defining area of car search


Car Reservation
User's browser
RequestController
ReservationController
LocationController
CarController
Cars
DB
1: User_desired_area_of_car_search(coord)
2: locate_nearby_cars(coord) : Reply
2.1: req_redirect(dest: reserv, arg: coord) : Reply
2.1.1: def_search_area(coord) : cars_nearby
loop
while[cars_to_check != empty]
2.1.1.1: get_car_coord() : car_coord
2.1.1.1.1: acquire_coord : car_coord
2.1.1.2: close_coords(car_coord, coord) : close
2.1.1.3: get_car_inf(cars_to_check): car_inf
alt
[close == true]
2.1.1.4: add_car_to_cars_nearby_set(car_coord, car_inf): cars_nearby
[else]
Reply = external_map_cars_repres(cars_nearby(car_coord))

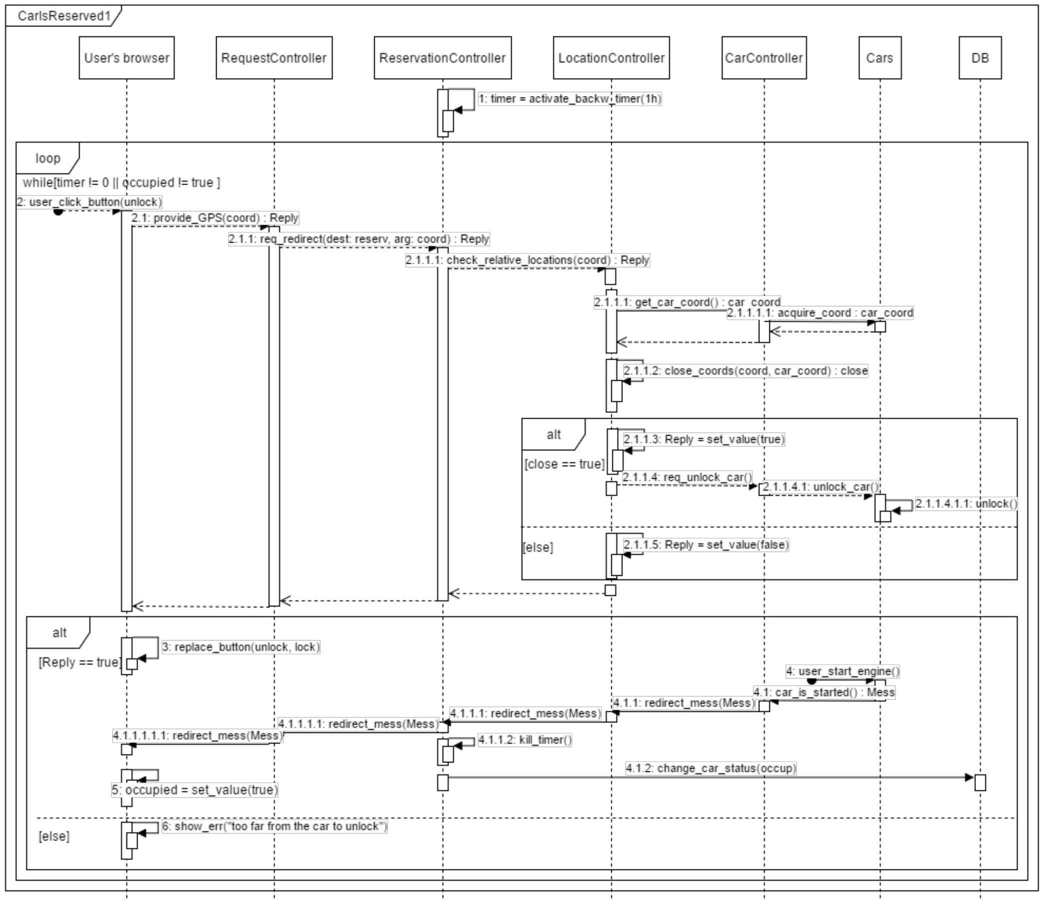
Firstly user chooses where to search cars: nearby user or nearby the specific area provided by user. In the diagram **??** at the beggining user either presses the button "Nearby" or he/she inputs address into the "Specific area" field and presses button "Submit". In both cases in the diagram these actions are treated as "User_desired_area_of_car_search(coord)" and in both cases browser sends coordinates.
Then coordinates are transferred to the CarController, which compares coordinates with each car in the database. It also collects current information about the car, including battery fulness. If car is not further than 3 ($km\hat{2}$) from the coordinates it is added to the set of cars_nearby. Proccess repeats until all cars are checked.
Then the set of cars_nearby is sent to external map web service, which is placed in the web page.

2.4.2 Choose the car


Car Choose
User's browser
RequestController
ReservationController
DB
loop
while[confirm != true && not_taken == false]
1: user_click_car(car)
2: show_car_info(car_inf)
3: show_button(reserve)
4: user_click_button(reserve)
5: show_mess(confirm_req)
alt
6: confirm_reserv(car) : Reply
[confirm == true]
6.1: req_redirect(dest: reserv, arg: car) : Reply
6.1.1: get_car_inf(car) : car_inf
6.2: check_if_already_reserved(car_inf) : reserved
alt
[reserved== true]
6.3: Reply = set_value(false)
7: show_err(already_taken)
[else]
6.4: Reply = set_value(true)
6.5: change_car_status(reserv)
8: not_taken = set_value(Reply)
[else]

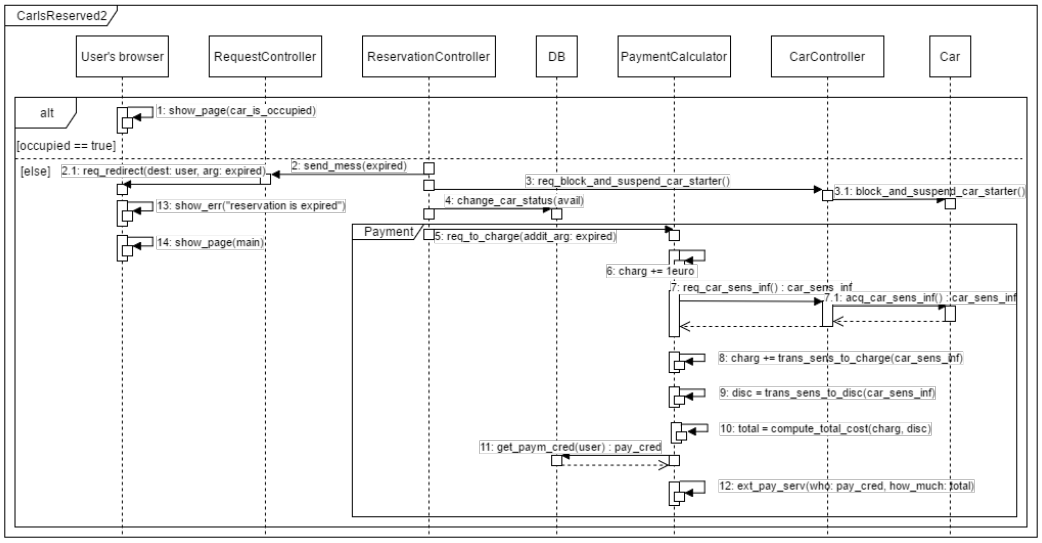
When user presses on the car on the map the information about it appears near the map window and "reserve" button popes up. When after selecting the car user presses the button "reserve" the system checks the up-to-date information about that car if it is already reserved. If it is not reserved system reserves it for the user(updates the database) and showes the Reservation page.

2.4.3 Car Is Reserved #1


CarIsReserved1
User's browser
RequestController
ReservationController
LocationController
CarController
Cars
DB
1: timer = activate_backw_timer(1h)
loop
while[timer != 0 || occupied != true]
2: user_click_button(unlock)
2.1: provide_GPS(coord) : Reply
2.1.1: req_redirect(dest: reserv, arg: coord) : Reply
2.1.1.1: check_relative_locations(coord) : Reply
2.1.1.1: get_car_coord() : car_coord
2.1.1.1.1: acquire_coord : car_coord
2.1.1.2: close_coords(coord, car_coord) : close
alt
[close == true]
2.1.1.3: Reply = set_value(true)
2.1.1.4: req_unlock_car()
2.1.1.4.1: unlock_car()
2.1.1.4.1.1: unlock()
[else]
2.1.1.5: Reply = set_value(false)
alt
[Reply == true]
3: replace_button(unlock, lock)
4: user_start_engine()
4.1: car_is_started() : Mess
4.1.1: redirect_mess(Mess)
4.1.1.1: redirect_mess(Mess)
4.1.1.1.1: redirect_mess(Mess)
4.1.1.1.1.1: redirect_mess(Mess)
4.1.1.2: kill_timer()
4.1.2: change_car_status(occup)
5: occupied = set_value(true)
[else]
6: show_err("too far from the car to unlock")

After user reserved the car BackwardTimer is set to 1 hour. Until the timer is not equals to 0 the user can still occupy the car. When user presses the "unlock button" on the web page system checkes whether user is close to the car. If not it return the message "too far from the car to unlock". When users coordinates are close to the coordinates of the car and user presses the "unlock button" the system send request to the car to unlock itself. At this point when user enters the car and starts the engine, Car sends acknowledgement to the system, ReservationController changes car status to the "occupied", kills the timer and requests the webpage to show the occupation page.
Note: If the user once unlocked the car, being close to the car, the new validated button "lock" button can be used from any distance.

2.4.4 Car Is Reserved #2 (continue)


CarIsReserved2
User's browser
RequestController
ReservationController
DB
PaymentCalculator
CarController
Car
alt
1: show_page(car_is_occupied)
[occupied == true]
[else]
2: send_mess(expired)
2.1: req_redirect(dest: user, arg: expired)
3: req_block_and_suspend_car_starter()
3.1: block_and_suspend_car_starter()
4: change_car_status(avail)
13: show_err("reservation is expired")
Payment
5: req_to_charge(addit_arg: expired)
14: show_page(main)
6: charg += 1euro
7: req_car_sens_inf() : car_sens_inf
7.1: acq_car_sens_inf() : car_sens_inf
8: charg += trans_sens_to_charge(car_sens_inf)
9: disc = trans_sens_to_disc(car_sens_inf)
10: total = compute_total_cost(charg, disc)
11: get_paym_cred(user) : pay_cred
12: ext_pay_serv(who: pay_cred, how_much: total)

If the timer is went to zero and user haven't started the engine, the system changes car status to "available" and asks the PaymentCalculator to sum up the cost, providing the charge of 1 euro. The error message "reservation is expired" appears in the browser and after few seconds main page is shown.
In the diagram **??** the frame "Payment" describes the basic algorithm of the PaymentCalculator. If the addit_arg is equals to "null", than the algorithm describes the case, when user stoppes the trip by himself and the step 6 can be ommitted.
In order to restrict the situation when user get into the car, but timer went to 0, system blocks and suspends(if user catched to start it) the car starter and locks the car(not seen in the diagram **??**). In that case user can unlock the door from inside, walk off the car and it will automatically lock it again.

2.4.5 Car Is Occupied

The runtime diagram is redundant to describe that moment of system use as it doesn't have any new specific cases that haven't been described yet and mechanisms of implementation are the same. Brief description:

- System provides the "Occupied" page, which contains 2 buttons - "lock/unlock" and "End of the Trip"
- Procedure of using buttons "lock/unlock" has been described in the runtime diagram **??** from the beggining.
- Button "End of the Trip" simply sends request to the ReservationController to change car status to "available". After that ReservationController sends request to the CarController to block the car starter(Car also locks itself) and requests the PaymentCalculator to sum up the total cost of the ride(see diagram **??** and read the description below).

3 Effort Spent

Artemiy Frolov:

```
26/11, 1h
28/11, 0.5h
30/11, 3h
1/12, 3h
2/12, 2h
3/12, 3h
4/12,
```