School of Industrial and Information Engineering

Department of Computer Science and Engineering

Software Engineering II

PowerEnJoy -Design Document

Presented by:

Bachar Senno

Gianluigi Iobizzi

Onell Saleeby

Under the supervision of:

Prof. Luca Mottola

Prof. Elisabetta Di Nitto

V1.1, 10-01-2017

Table Of Contents:

Introduction

Purpose:

The purpose of this document is to give more technical details than the RASD about PowerEnjoy system.

This document is addressed to developers and aims to identify:

- The high level architecture
- The design patterns
- The main components and their interfaces provided one for another
- The Runtime behavior

Scope:

The project Powerenjoy, which is a service based on mobile application , has a main target which are the clients.

The system allows clients to reserve a car via a mobile using his GPS position or inserting an address manually to find a car within 5 km of distance from the chosen point.

The user should register in order to login, and be able to reserve the car and to start a trip.

The system includes extra services and functionalities such as discounts in cases like riding with more than three person or leaving a power level equal or more than 50% ,and extra charges in cases of leaving the car more than 3 km away from the charging station or leaving a power level equal or less than 20%

The main purpose of the system is to make possible a noticeable reduction of the air pollution in the future and to offer an excellent service to the clients.

Definitions, acronyms, abbreviations:

- RASD: requirements analysis and specifications document
- DD: design document
- API: application programming interface; it is a common way to communicate with another system.

- MVC: model view controller

- URL: uniform resource locator

- Push notification: it is a notification sent to a smartphone using the mobile application, so it must be installed.

- Push service: it is a service that allows to send push notifications with own API

- REST: Representational State Transfer

- RESTful: REST with no session

- UX: user experience design

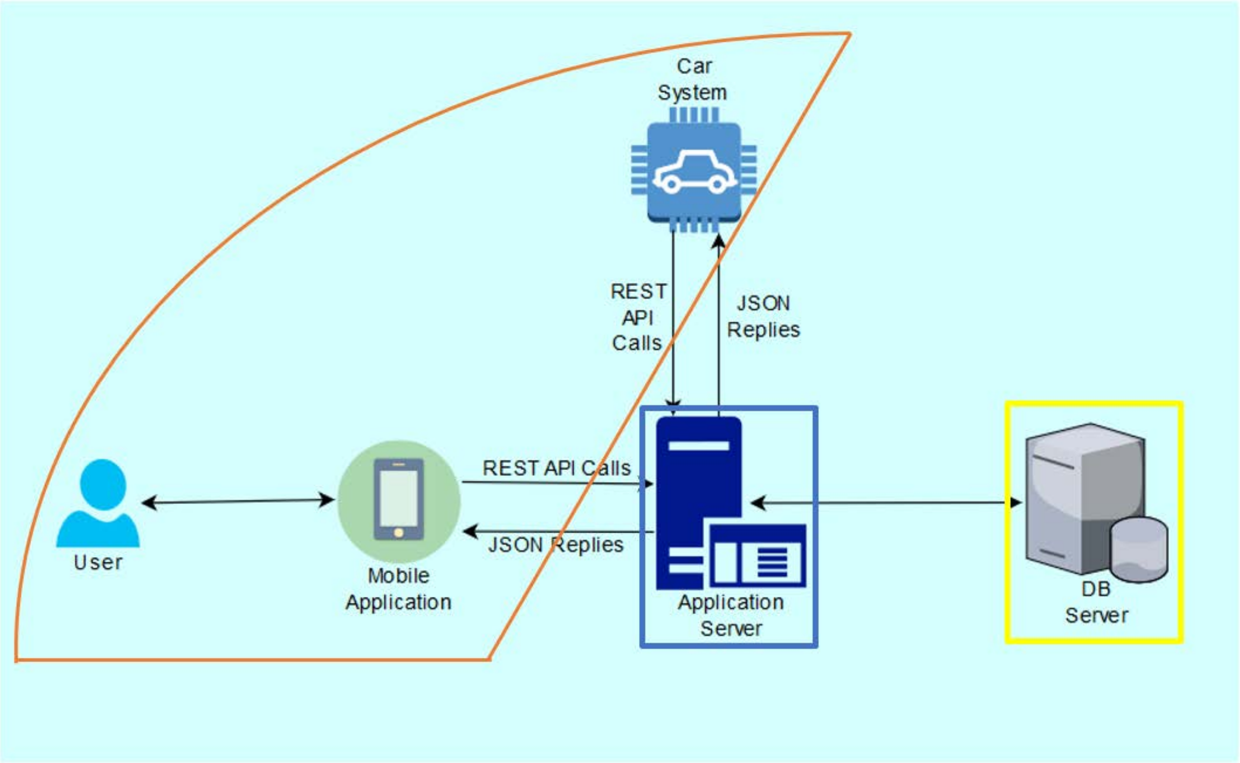
- BCE: business controller entity

Reference documents:

- RASD document
- Sample Design Deliverable Discussed on Nov.2.pdf
- Architecture And Design In Practice.pdf
- Design – part I & II.pdf

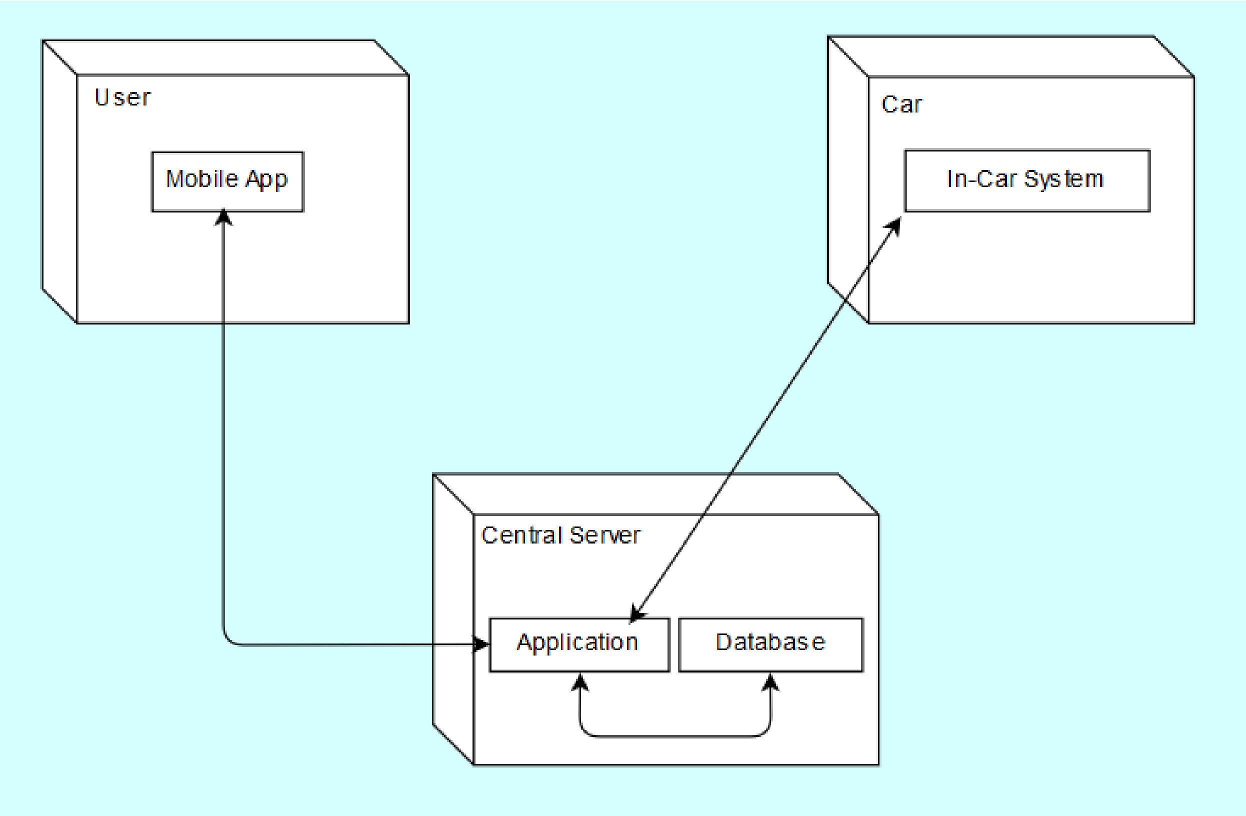
Architectural Design

Overview:



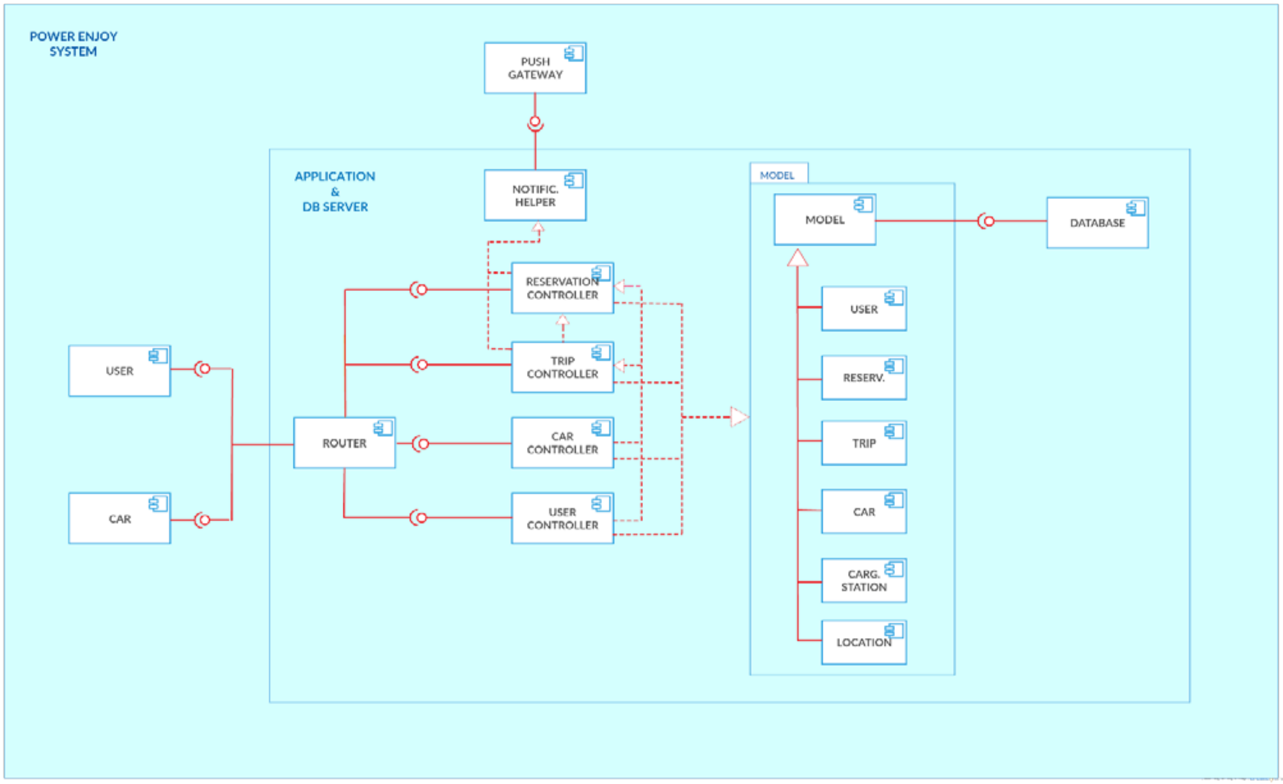
This diagram shows the high-level architectural design of our system. A shown above, our system adopts a 3-tier architecture. The database server contains the DBMS and the physical data pertaining to users, cars, and any other useful information that we need to store. The application server is the main linking and operations point. It has access to the data stored in the DB server, and it can communicate with the clients (cars and users). The clients and send GET and POST requests to the application server using REST API calls, and the application server responds with relevant JSON-encoded replies/data.

High level components and their interaction:



This diagram shows the 3 main components that are the basic components of our system. At the center, we have a central server, which contains the main API for our app, as well as the database which contains the actual data. The central server serves as a link between the users and the cars, so all interactions between these two have to go through the central server (reserving, unlocking, and using a car). The central server contains all the needed functions and methods to guarantee seamless and efficient operations between the users and the cars. Also, the central server for all computations and decision making (calculating the trips fees, withdrawing money, imposing the cooldown period after a failed reservation etc..). In order to do this, the central server can communicate with both parties in order to obtain any needed information. For example, the central server can query a car to know the remaining power level after a trip and whether it’s plugged in a charging station or not at the end of a trip, since all of these factors are taken into consideration when calculating the fee.

Component view:



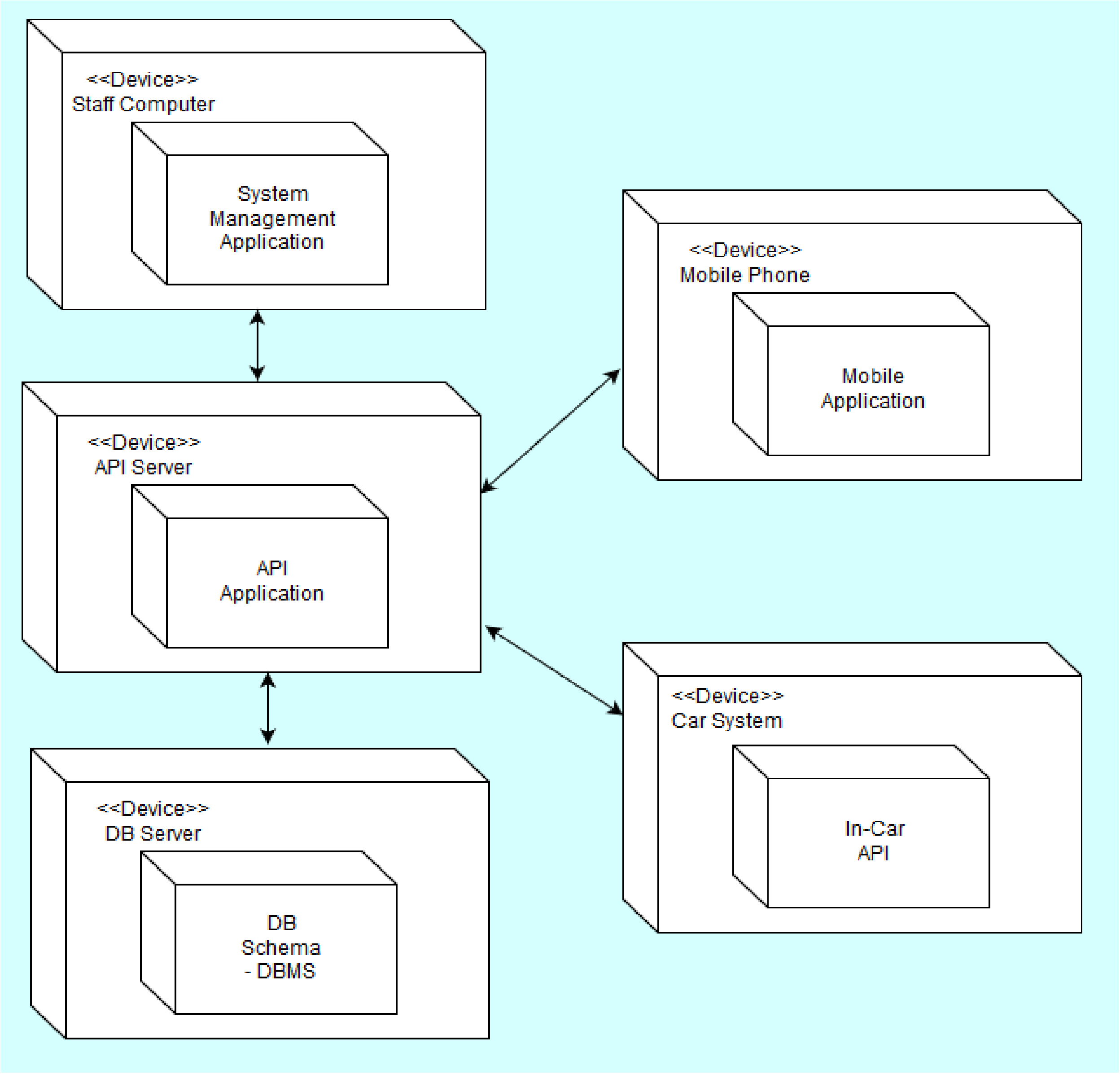
Component Diagram:

The system exposes many public resources, all of them require a proper authentication and authorization to be used. The router will dispatch the incoming request from a user's mobile app or a car's system to the appropriate Controller, it will then do the work as required by the client. In what follows, we provide a brief description of all them:

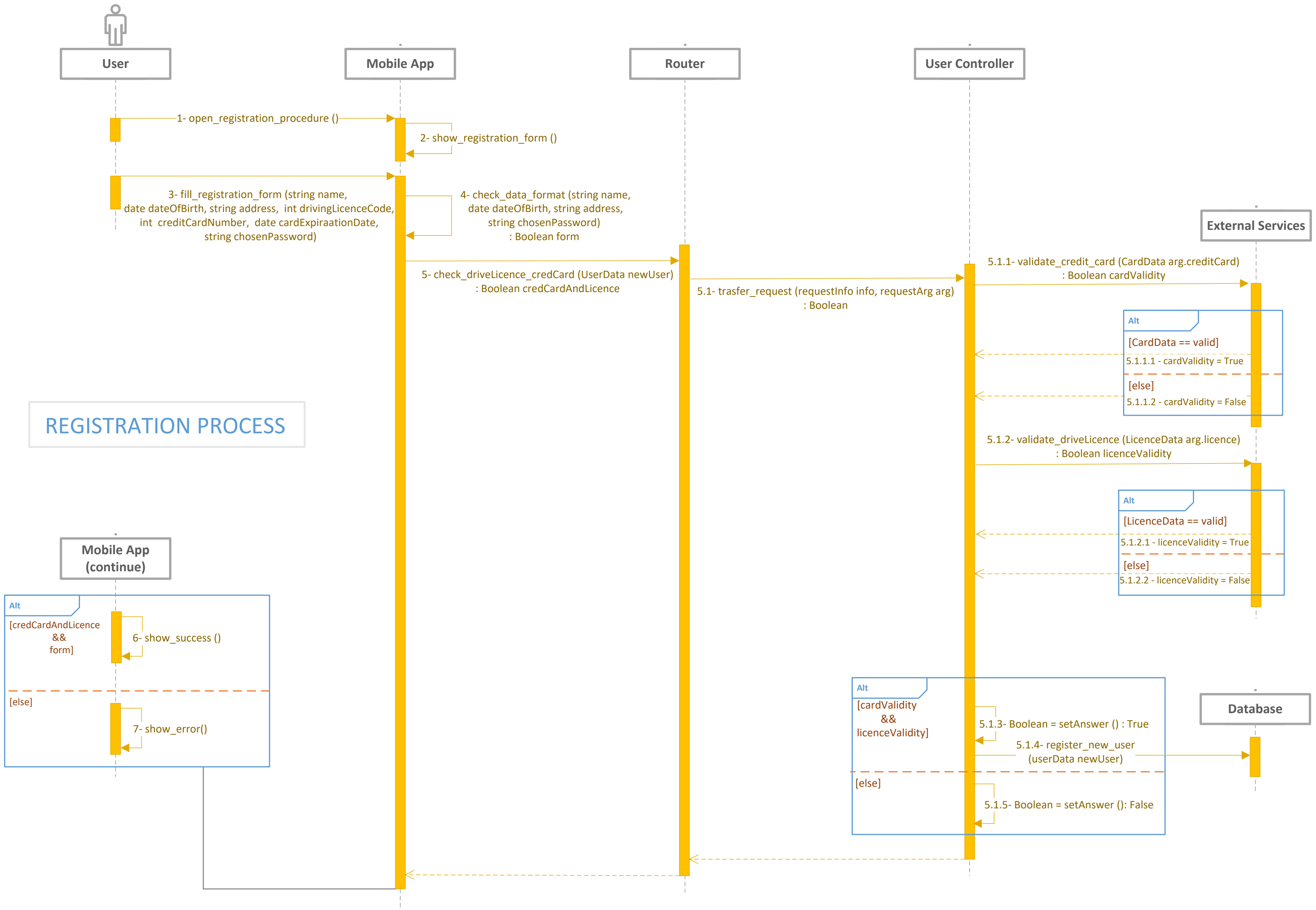
- RESERVATION CONTROLLER - Its role is the correct management of the car-search engine and the validation of the whole car-reservation request;
- TRIP CONTROLLER – Makes possible the start of the trip when confirmed by the user and during the trip it continuously updates the data and the total cost of the ride. At the end of it is responsible of sending the charge requests over the user's credit card to the payment handler;
- CAR CONTROLLER – When it questioned about the status of a car it is in charge of monitor the car data through the dedicated interface and to transmit them to the requiring controller;
- USER CONTROLLER – Dedicated to the management of the account data of every system user. It plays an important role during the login phase.

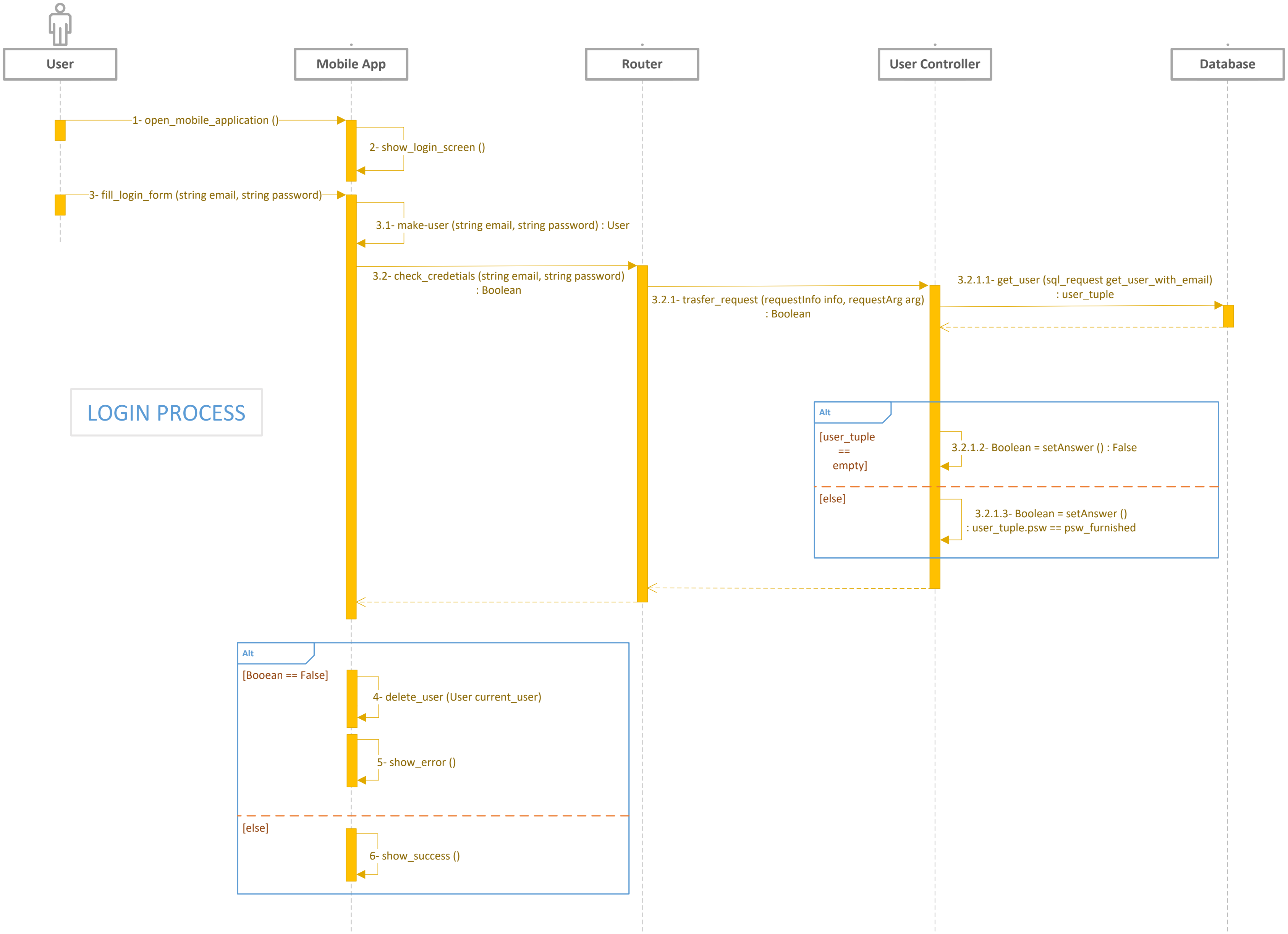
- NOTIFICATION HELPER - will take care of the kind of communication to use for the interaction with the required device.
  For instance, it will use the Push Gateway own APIs to interact with the User's app.

Deployment view:



Runtime view:

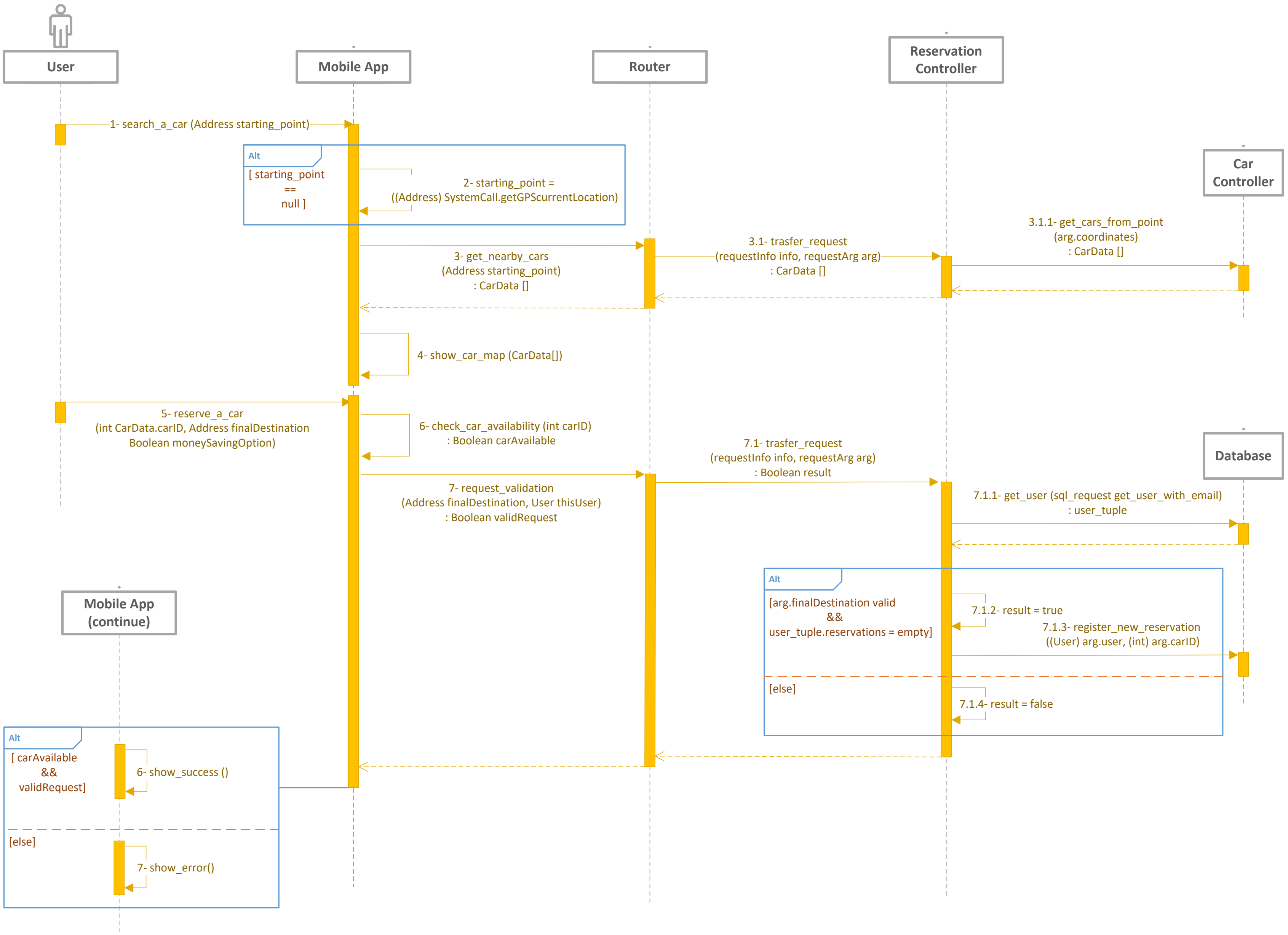

User
Mobile App
Router
User Controller
External Services
Database
1- open_registration_procedure ()
2- show_registration_form ()
3- fill_registration_form (string name,
date dateOfBirth, string address, int drivingLicenceCode,
int creditCardNumber, date cardExpiraationDate,
string chosenPassword)
4- check_data_format (string name,
date dateOfBirth, string address,
string chosenPassword)
: Boolean form
5- check_driveLicence_credCard (UserData newUser)
: Boolean credCardAndLicence
5.1- trasfer_request (requestInfo info, requestArg arg)
: Boolean
5.1.1- validate_credit_card (CardData arg.creditCard)
: Boolean cardValidity
Alt
[CardData == valid]
5.1.1.1 - cardValidity = True
[else]
5.1.1.2 - cardValidity = False
5.1.2- validate_driveLicence (LicenceData arg.licence)
: Boolean licenceValidity
Alt
[LicenceData == valid]
5.1.2.1 - licenceValidity = True
[else]
5.1.2.2 - licenceValidity = False
Alt
[cardValidity
&&
licenceValidity]
5.1.3- Boolean = setAnswer () : True
5.1.4- register_new_user
(userData newUser)
[else]
5.1.5- Boolean = setAnswer (): False
REGISTRATION PROCESS
Mobile App
(continue)
Alt
[credCardAndLicence
&&
form]
6- show_success ()
[else]
7- show_error()

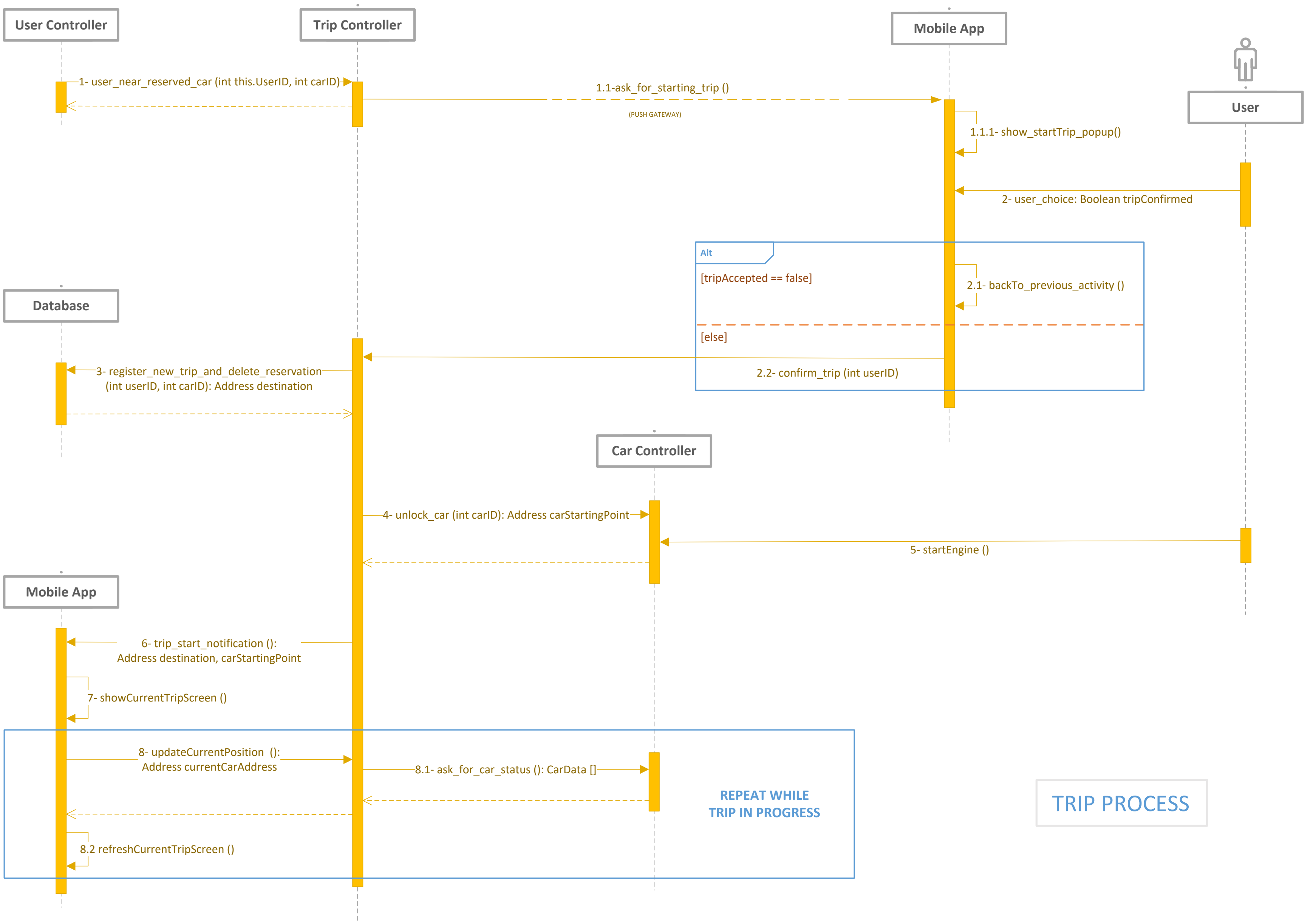


LOGIN PROCESS

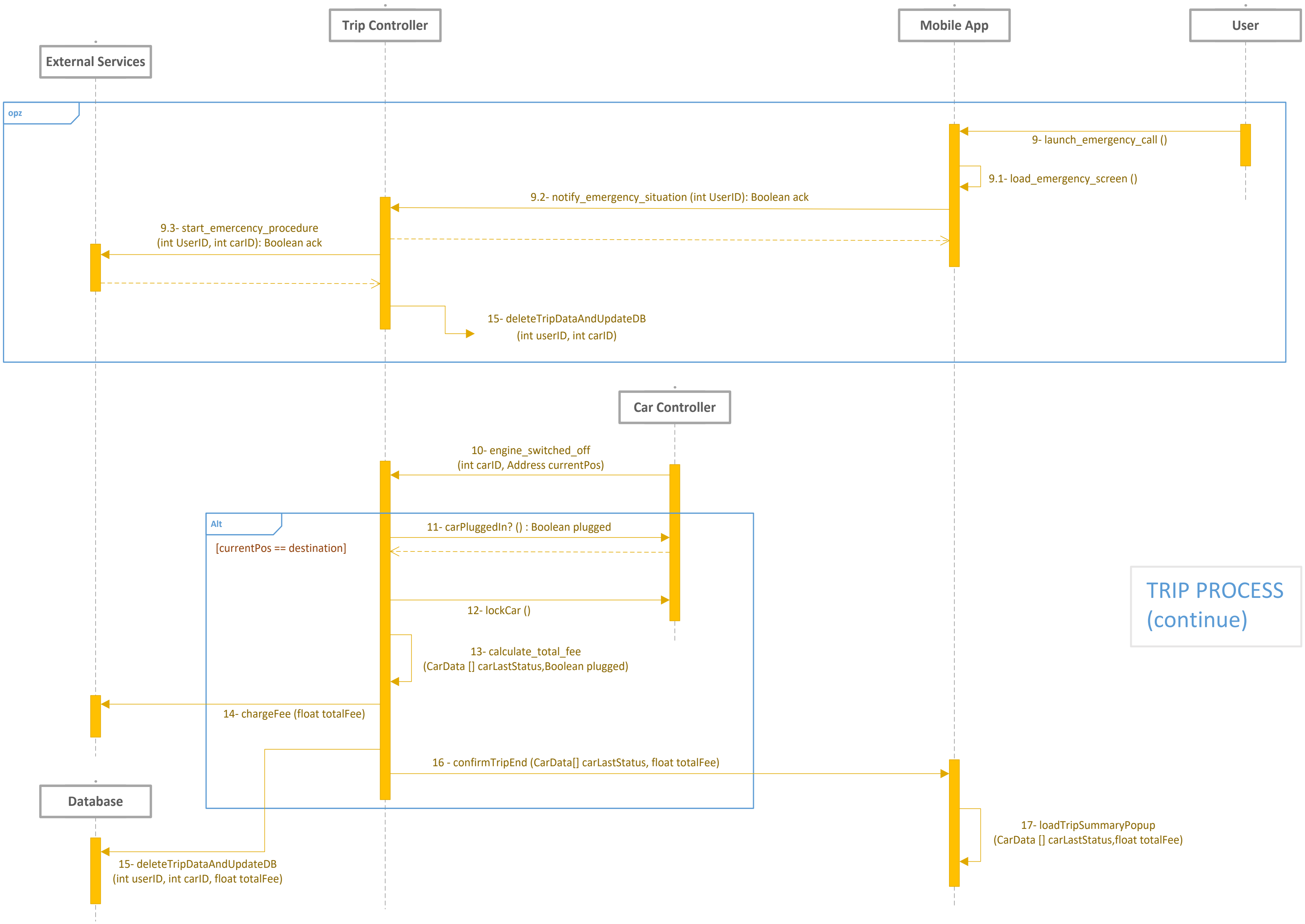
User
Mobile App
Router
User Controller
Database
1- open_mobile_application ()
2- show_login_screen ()
3- fill_login_form (string email, string password)
3.1- make-user (string email, string password) : User
3.2- check_credetials (string email, string password)
: Boolean
3.2.1- trasfer_request (requestInfo info, requestArg arg)
: Boolean
3.2.1.1- get_user (sql_request get_user_with_email)
: user_tuple
Alt
[user_tuple
==
empty]
3.2.1.2- Boolean = setAnswer () : False
[else]
3.2.1.3- Boolean = setAnswer ()
: user_tuple.psw == psw_furnished
Alt
[Booean == False]
4- delete_user (User current_user)
5- show_error ()
[else]
6- show_success ()

RESERVATION PROCESS


User
Mobile App
Router
Reservation Controller
Car Controller
Database
1- search_a_car (Address starting_point)
Alt
[starting_point == null]
2- starting_point = ((Address) SystemCall.getGPScurrentLocation)
3- get_nearby_cars (Address starting_point) : CarData []
3.1- trasfer_request (requestInfo info, requestArg arg) : CarData []
3.1.1- get_cars_from_point (arg.coordinates) : CarData []
4- show_car_map (CarData[])
5- reserve_a_car (int CarData.carID, Address finalDestination Boolean moneySavingOption)
6- check_car_availability (int carID) : Boolean carAvailable
7- request_validation (Address finalDestination, User thisUser) : Boolean validRequest
7.1- trasfer_request (requestInfo info, requestArg arg) : Boolean result
7.1.1- get_user (sql_request get_user_with_email) : user_tuple
Alt
[arg.finalDestination valid && user_tuple.reservations = empty]
7.1.2- result = true
7.1.3- register_new_reservation ((User) arg.user, (int) arg.carID)
[else]
7.1.4- result = false
Mobile App (continue)
Alt
[carAvailable && validRequest]
6- show_success ()
[else]
7- show_error()

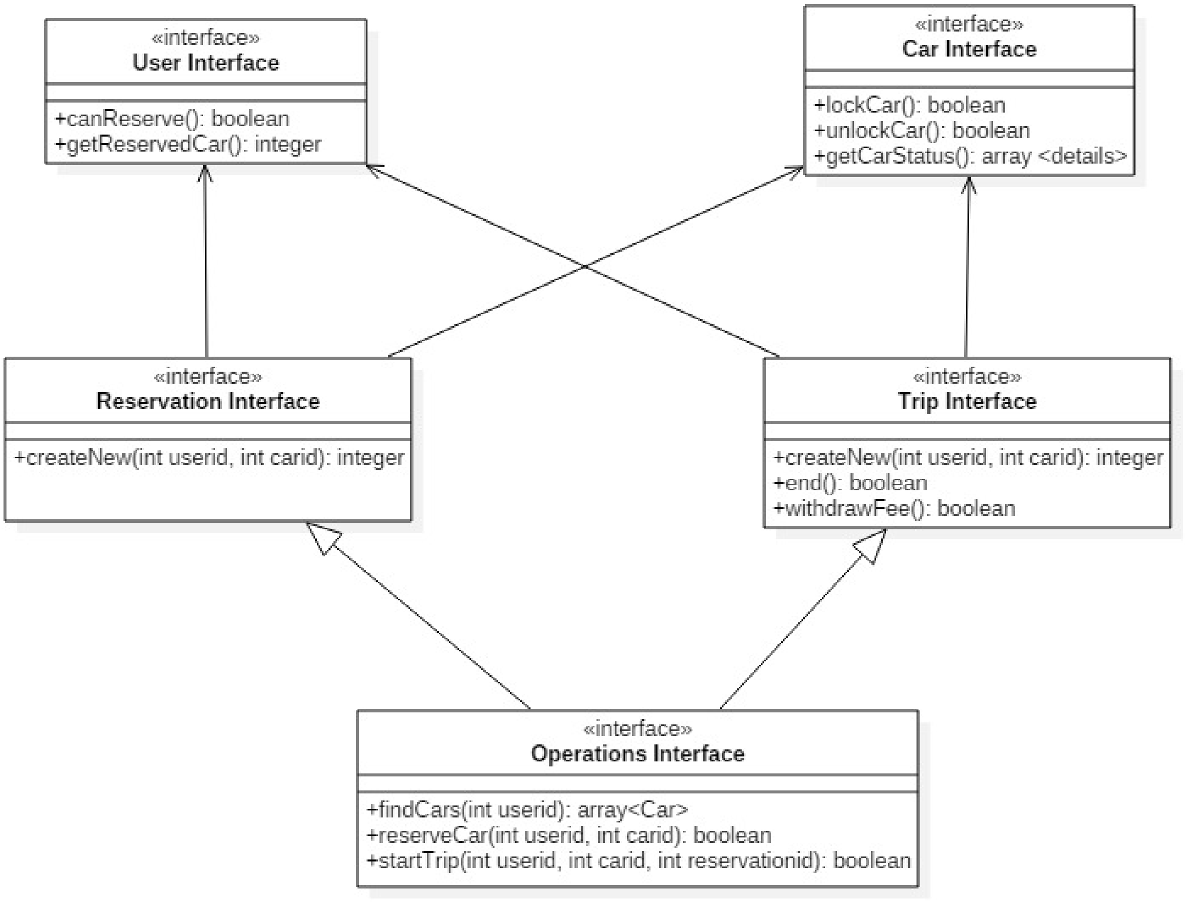

User Controller
Trip Controller
Mobile App
User
1- user_near_reserved_car (int this.UserID, int carID)
1.1-ask_for_starting_trip ()
(PUSH GATEWAY)
1.1.1- show_startTrip_popup()
2- user_choice: Boolean tripConfirmed
Alt
[tripAccepted == false]
2.1- backTo_previous_activity ()
[else]
2.2- confirm_trip (int userID)
Database
3- register_new_trip_and_delete_reservation
(int userID, int carID): Address destination
Car Controller
4- unlock_car (int carID): Address carStartingPoint
5- startEngine ()
Mobile App
6- trip_start_notification ():
Address destination, carStartingPoint
7- showCurrentTripScreen ()
8- updateCurrentPosition ():
Address currentCarAddress
8.1- ask_for_car_status (): CarData []
8.2 refreshCurrentTripScreen ()
REPEAT WHILE
TRIP IN PROGRESS

TRIP PROCESS


Trip Controller
Mobile App
User
External Services
opz
9- launch_emergency_call ()
9.1- load_emergency_screen ()
9.2- notify_emergency_situation (int UserID): Boolean ack
9.3- start_emercency_procedure
(int UserID, int carID): Boolean ack
15- deleteTripDataAndUpdateDB
(int userID, int carID)
Car Controller
10- engine_switched_off
(int carID, Address currentPos)
Alt
[currentPos == destination]
11- carPluggedIn? () : Boolean plugged
12- lockCar ()
13- calculate_total_fee
(CarData [] carLastStatus,Boolean plugged)
14- chargeFee (float totalFee)
16 - confirmTripEnd (CarData[] carLastStatus, float totalFee)
Database
15- deleteTripDataAndUpdateDB
(int userID, int carID, float totalFee)
17- loadTripSummaryPopup
(CarData [] carLastStatus,float totalFee)

TRIP PROCESS
(continue)

Component Interfaces


«interface»
User Interface
+canReserve(): boolean
+getReservedCar(): integer
«interface»
Car Interface
+lockCar(): boolean
+unlockCar(): boolean
+getCarStatus(): array <details>
«interface»
Reservation Interface
+createNew(int userid, int carid): integer
«interface»
Trip Interface
+createNew(int userid, int carid): integer
+end(): boolean
+withdrawFee(): boolean
«interface»
Operations Interface
+findCars(int userid): array<Car>
+reserveCar(int userid, int carid): boolean
+startTrip(int userid, int carid, int reservationid): boolean

Architectural Style and Patterns:

Adopting a dynamic architectural style was what we believe to provide us with the best results. Therefore, when considering the user interface for example, we adopted a top-down (high-level to low-level) architectural style and way of thinking. We tried to place ourselves in the users' shoes, consider what a complete optimal interface looks like, and then slice it down into separate smaller components and design them.

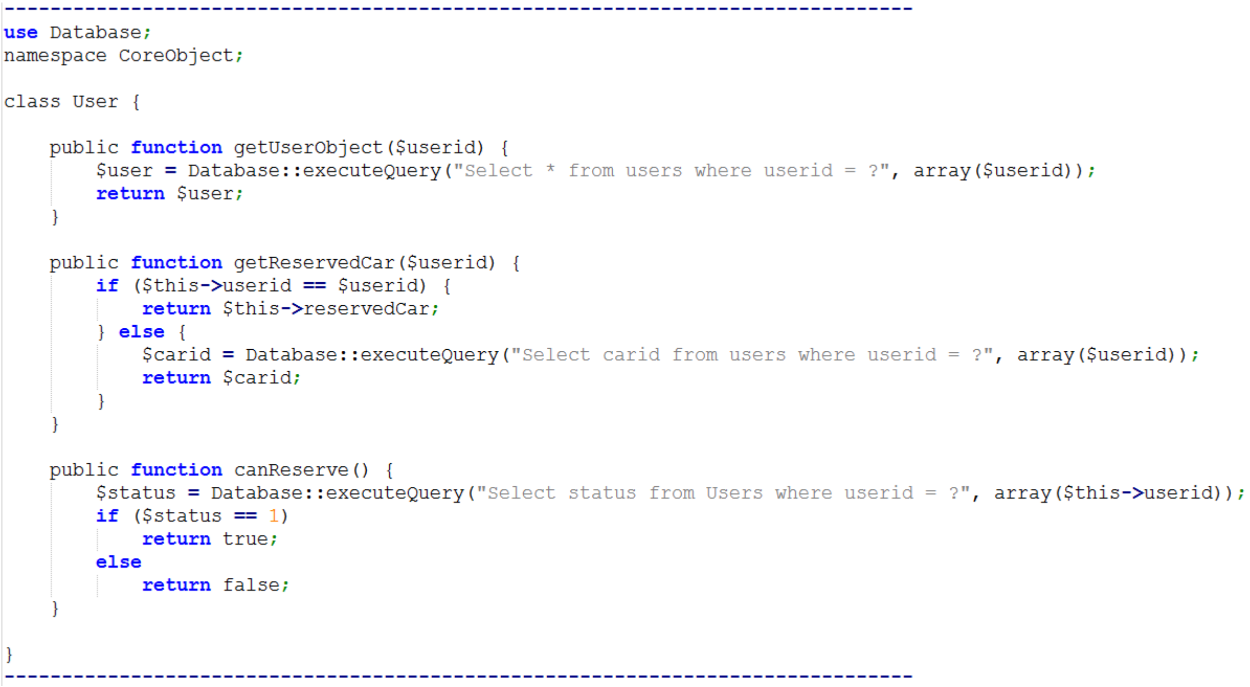
On the other hand, when we were considering the actual inner-working of the system, the components architectures, and the algorithm design, we went for a bottom-up (low-level to high-level) approach, since this allows for more reusability of the code, and also it offers more chance for unit testing, error avoidance, and increased granularity.

Adopting different architectural approaches to different aspects of the system allowed us to achieve our objectives and minimize any error chances, but also offer the user an overall satisfying experience.

Algorithm Design

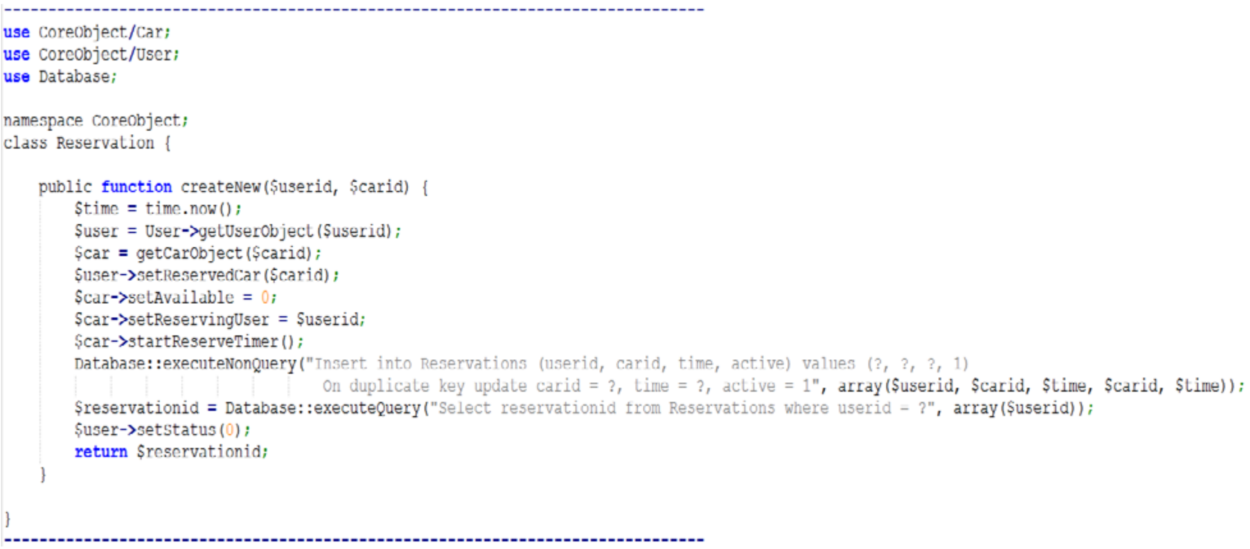
In what follows, we use (pseudo)-code to illustrate in a simplified manner the actual code implementation of the different components. The code is written in a php-like language, and shows the different classes in the project.

Class User:



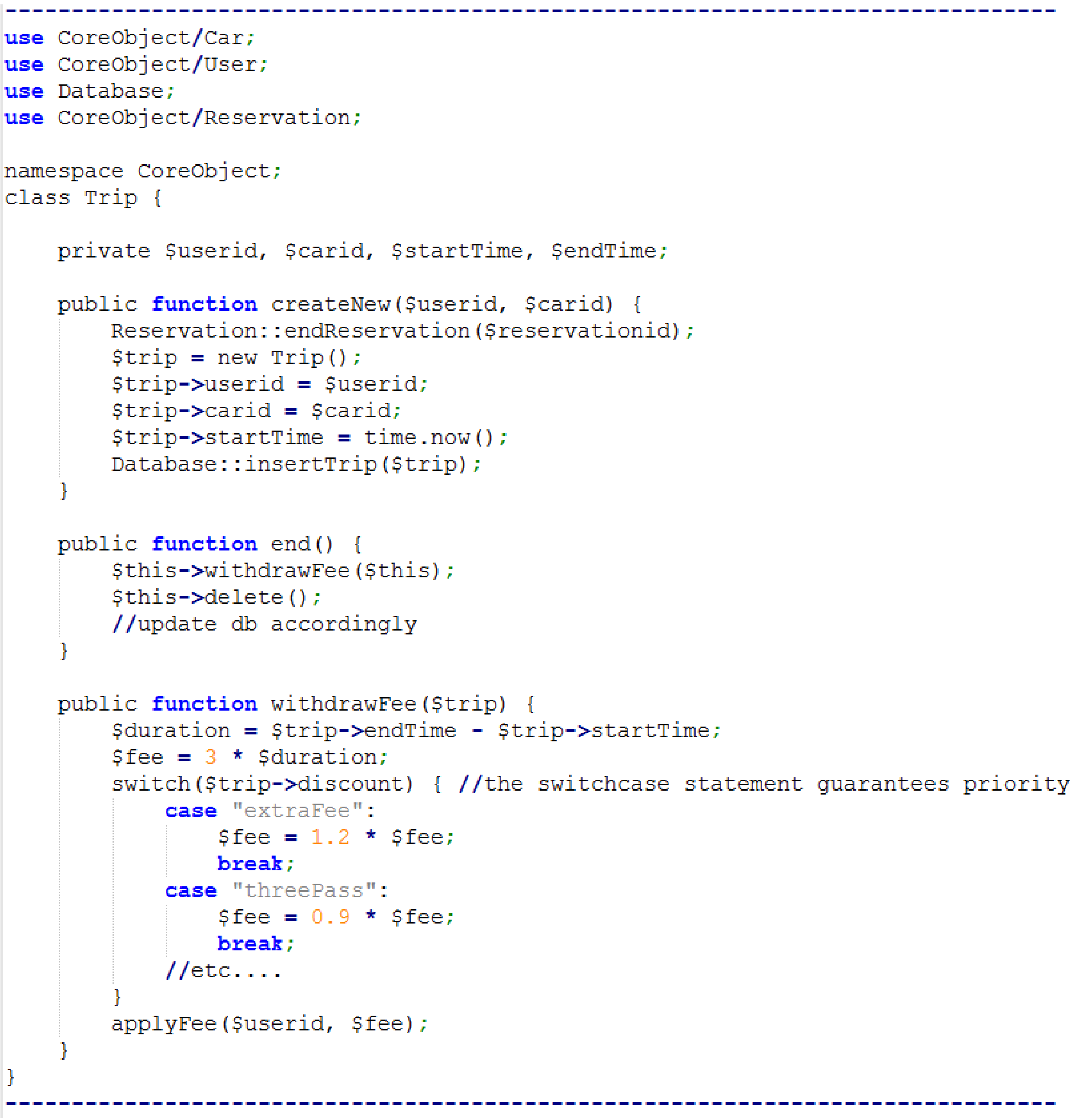
```
--------------------------------------------------------------------------------
use Database;
namespace CoreObject;

class User {

    public function getUserObject($userid) {
        $user = Database::executeQuery("Select * from users where userid = ?", array($userid));
        return $user;
    }

    public function getReservedCar($userid) {
        if ($this->userid == $userid) {
            return $this->reservedCar;
        } else {
            $carid = Database::executeQuery("Select carid from users where userid = ?", array($userid));
            return $carid;
        }
    }

    public function canReserve() {
        $status = Database::executeQuery("Select status from Users where userid = ?", array($this->userid));
        if ($status == 1)
            return true;
        else
            return false;
    }

}
--------------------------------------------------------------------------------
```

Class Reservation:



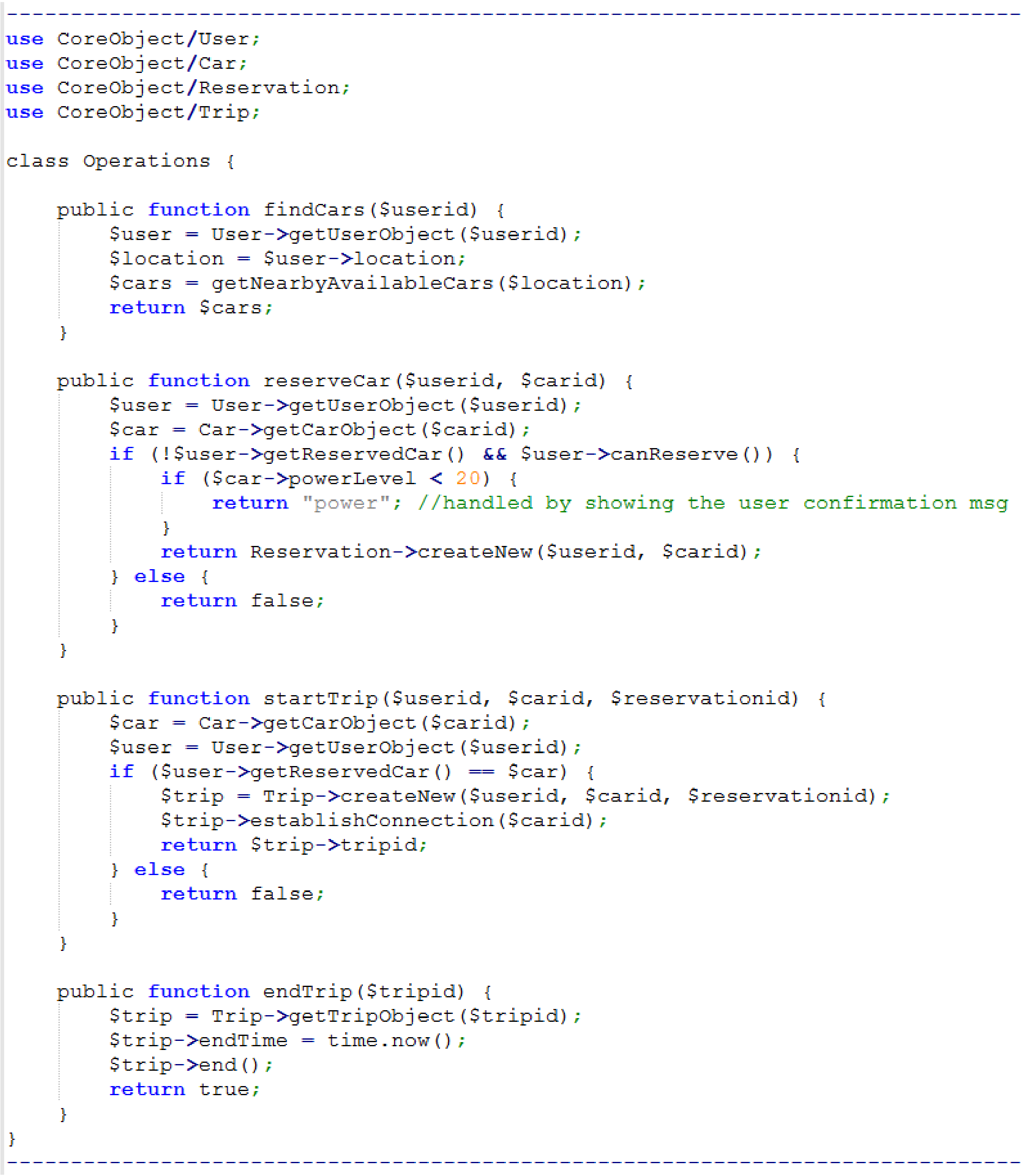
```
--------------------------------------------------------------------------------
use CoreObject/Car;
use CoreObject/User;
use Database;

namespace CoreObject;
class Reservation {

    public function createNew($userid, $carid) {
        $time = time.now();
        $user = User->getUserObject($userid);
        $car = getCarObject($carid);
        $user->setReservedCar($carid);
        $car->setAvailable = 0;
        $car->setReservingUser = $userid;
        $car->startReserveTimer();
        Database::executeNonQuery("Insert into Reservations (userid, carid, time, active) values (?, ?, ?, 1)
                                  On duplicate key update carid = ?, time = ?, active = 1", array($userid, $carid, $time, $carid, $time));
        $reservationid = Database::executeQuery("Select reservationid from Reservations where userid = ?", array($userid));
        $user->setStatus(0);
        return $reservationid;
    }

}
--------------------------------------------------------------------------------
```

Class Trip:



```
------------------------------------------------------------------------------
use CoreObject/Car;
use CoreObject/User;
use Database;
use CoreObject/Reservation;

namespace CoreObject;
class Trip {

    private $userid, $carid, $startTime, $endTime;

    public function createNew($userid, $carid) {
        Reservation::endReservation($reservationid);
        $trip = new Trip();
        $trip->userid = $userid;
        $trip->carid = $carid;
        $trip->startTime = time.now();
        Database::insertTrip($trip);
    }

    public function end() {
        $this->withdrawFee($this);
        $this->delete();
        //update db accordingly
    }

    public function withdrawFee($trip) {
        $duration = $trip->endTime - $trip->startTime;
        $fee = 3 * $duration;
        switch($trip->discount) { //the switchcase statement guarantees priority
            case "extraFee":
                $fee = 1.2 * $fee;
                break;
            case "threePass":
                $fee = 0.9 * $fee;
                break;
            //etc....
        }
        applyFee($userid, $fee);
    }
}
------------------------------------------------------------------------------
```

Class Operations:



```
--------------------------------------------------------------------------------
use CoreObject/User;
use CoreObject/Car;
use CoreObject/Reservation;
use CoreObject/Trip;

class Operations {

    public function findCars($userid) {
        $user = User->getUserObject($userid);
        $location = $user->location;
        $cars = getNearbyAvailableCars($location);
        return $cars;
    }

    public function reserveCar($userid, $carid) {
        $user = User->getUserObject($userid);
        $car = Car->getCarObject($carid);
        if (!$user->getReservedCar() && $user->canReserve()) {
            if ($car->powerLevel < 20) {
                return "power"; //handled by showing the user confirmation msg
            }
            return Reservation->createNew($userid, $carid);
        } else {
            return false;
        }
    }

    public function startTrip($userid, $carid, $reservationid) {
        $car = Car->getCarObject($carid);
        $user = User->getUserObject($userid);
        if ($user->getReservedCar() == $car) {
            $trip = Trip->createNew($userid, $carid, $reservationid);
            $trip->establishConnection($carid);
            return $trip->tripid;
        } else {
            return false;
        }
    }

    public function endTrip($tripid) {
        $trip = Trip->getTripObject($tripid);
        $trip->endTime = time.now();
        $trip->end();
        return true;
    }
}
--------------------------------------------------------------------------------
```

Requirements Traceability:

Naturally, the suggested design architecture along with the suggested components aim to fulfill the requirements we set in the RASD. That way, and taking into account the domain assumptions that we previously made, we can guarantee that the goals of the project will be realized. In the following, we concretize this by linking the previously set goals with the different components we suggest in the design document.

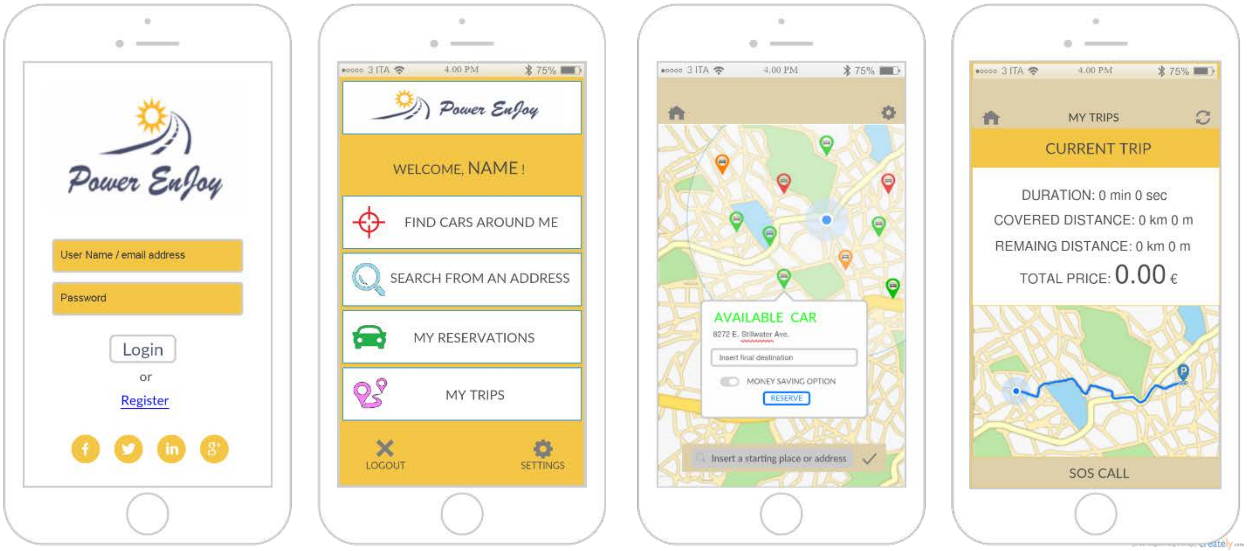
- [G1] Provide users with means to register and verify themselves:
  - User Controller + Database
  - Router
- [G2] Guarantee unique login per account:
  - User Controller
  - Router
- [G3] Provide users with interface to view their location and available nearby cars (within 5 km), and show which have low power level:
  - User Controller
  - Car Controller
  - Reservation Controller
- [G4] Allow the user to exclusively reserve one of the nearby cars for 1 hour:
  - User Controller
  - Car Controller
  - Reservation Controller
- [G5] Provide user with route to the destination. Offer path to the nearest charging station in case a money saving option is enabled:
  - Trip Controller
  - Reservation Controller
  - Car Controller
  - User Controller
- [G6] Prevent misuse of the app by continuous reservations:
  - Reservation Controller
  - User Controller

- [G7] Allow the user to access to the car once it’s reached:
  - Car Controller
  - Reservation Controller
- [G8] Start calculating the trip fee and show it to the user once the trip starts:
  - Car Controller
  - Trip Controller
- [G9] End the trip, withdraw the money, and lock the car, making it available again for reservation.
  - Car Controller
  - Reservation Controller
  - User Controller
  - Trip Controller
- [G10] Prevent the car from being ditched randomly:
  - Car Controller
  - Trip Controller
- [G11] Apply variations to the basic trip fee
  - Trip Controller
  - User Controller
  - Car Controller

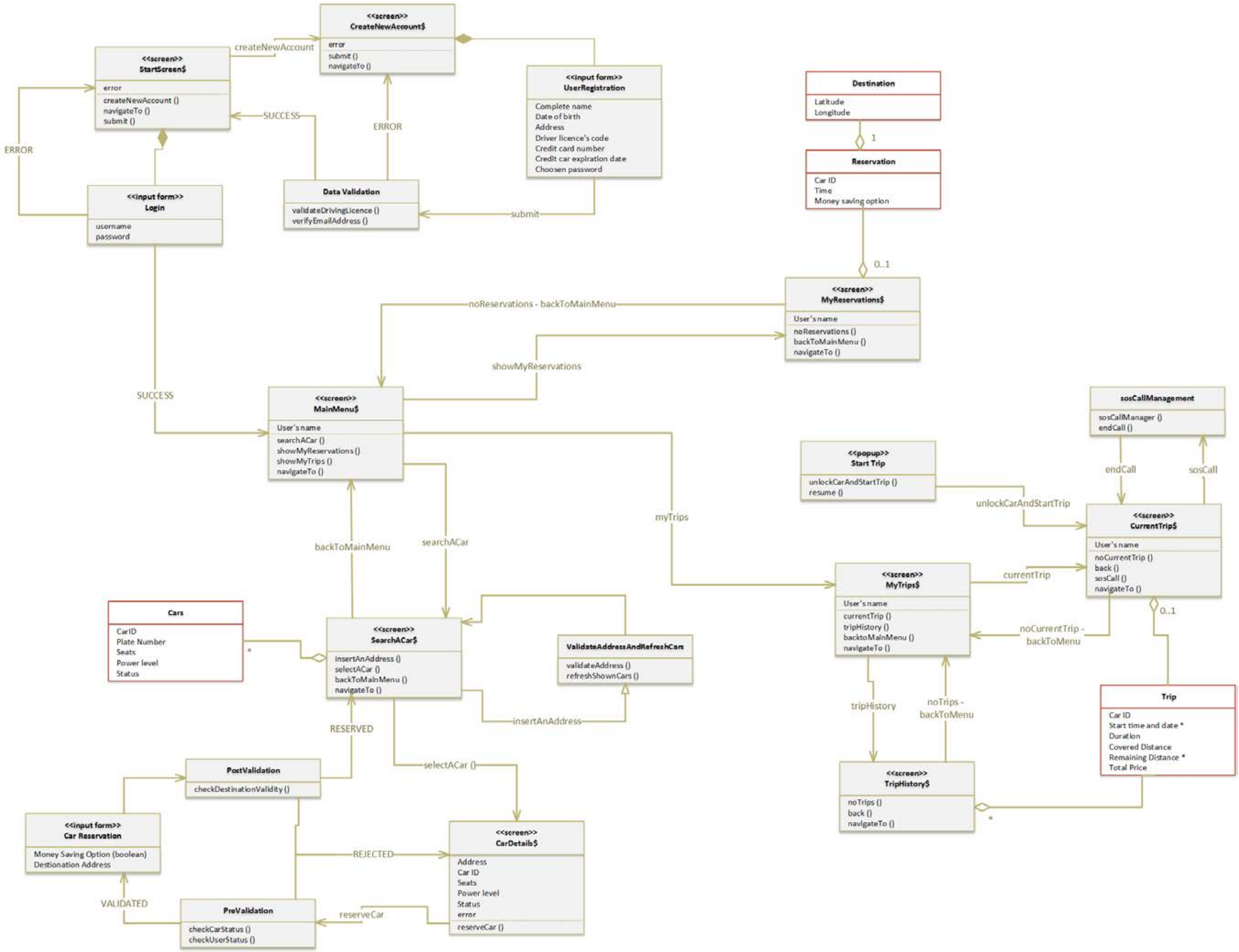
User Interface Design

This section the focus will be on the graphic interface layer of the USER's MOBILE APPLICATION. The human interface of secondary applications such the in-car system and central system's management application for his own simplicity is taken as implicit and not discussed here. The screen mockups and the UX – BCE diagrams are the conceptual tools used for the interface project. With the mockups, we give a general and not restrictive idea of the composition of the main app menus and screens then a rigorous planning of the interface is provided by the UX and BCE diagram. A UX diagram is an UML-based graphic tool useful to show the exact structure and behavior of all the interface's subcomponent and to provide a detailed idea of the whole presentation layer. With the BCE diagram instead, we define the interaction of the interface with the application and data sub-layers and, definitively, we plan how the entire application will divided over different tiers.

Mockups:

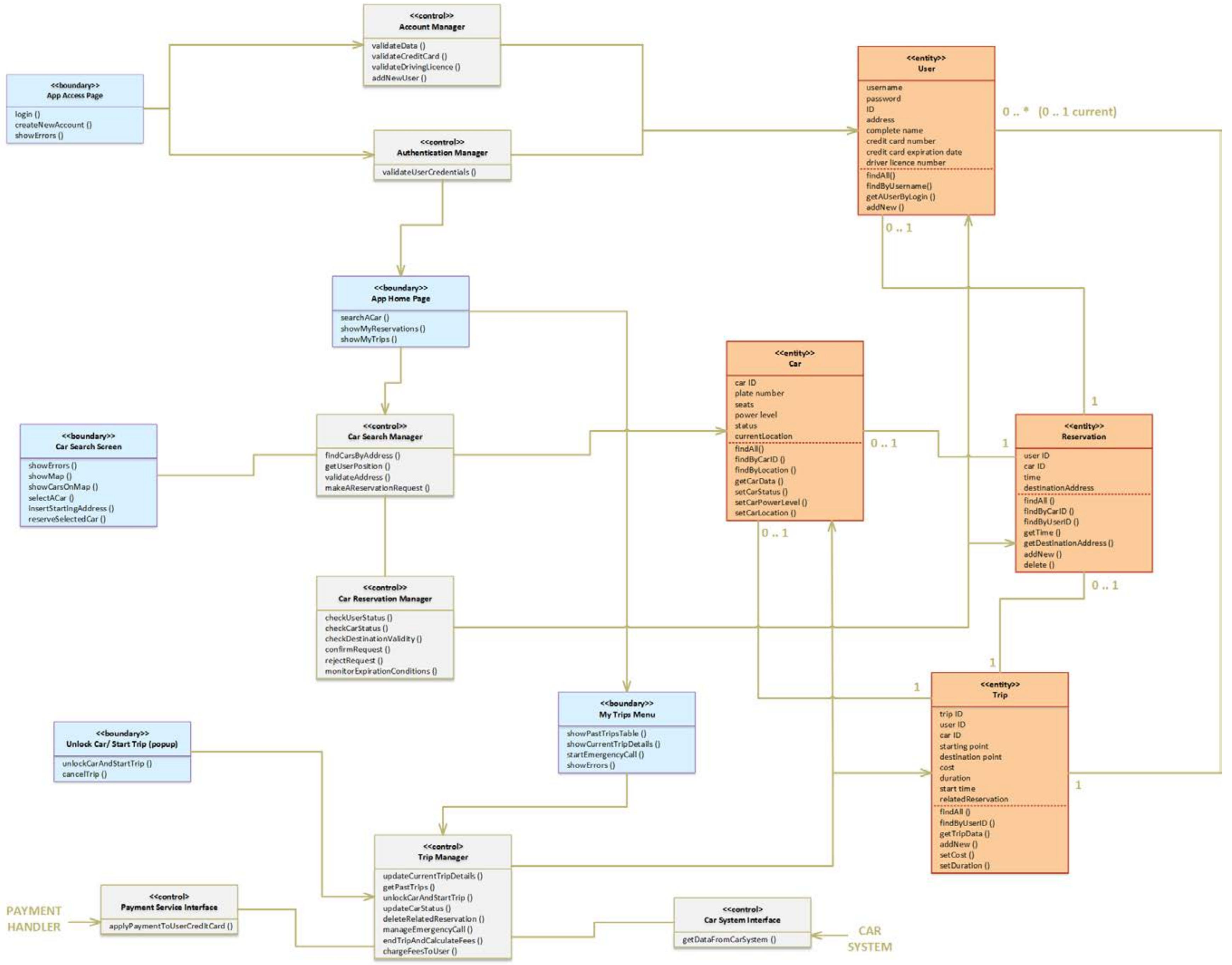


UX Diagram:



<<screen>>
StartScreen$
error
createNewAccount ()
navigateTo ()
submit ()
createNewAccount
<<screen>>
CreateNewAccount$
error
submit ()
navigateTo ()
<<input form>>
UserRegistration
Complete name
Date of birth
Address
Driver licence's code
Credit card number
Credit car expiration date
Choosen password
SUCCESS
ERROR
ERROR
<<input form>>
Login
username
password
Data Validation
validateDrivingLicence ()
verifyEmailAddress ()
submit
Destination
Latitude
Longitude
1
Reservation
Car ID
Time
Money saving option
0..1
<<screen>>
MyReservations$
User's name
noReservations ()
backToMainMenu ()
navigateTo ()
noReservations - backToMainMenu
showMyReservations
SUCCESS
<<screen>>
MainMenu$
User's name
searchACar ()
showMyReservations ()
showMyTrips ()
navigateTo ()
sosCallManagement
sosCallManager ()
endCall ()
endCall
sosCall
<<popup>>
Start Trip
unlockCarAndStartTrip ()
resume ()
unlockCarAndStartTrip
myTrips
<<screen>>
CurrentTrip$
User's name
noCurrentTrip ()
back ()
sosCall ()
navigateTo ()
backToMainMenu
searchACar
currentTrip
<<screen>>
MyTrips$
User's name
currentTrip ()
tripHistory ()
backtoMainMenu ()
navigateTo ()
0..1
noCurrentTrip - backToMenu
Cars
CarID
Plate Number
Seats
Power level
Status
*
<<screen>>
SearchACar$
insertAnAddress ()
selectACar ()
backToMainMenu ()
navigateTo ()
ValidateAddressAndRefreshCars
validateAddress ()
refreshShownCars ()
insertAnAddress
Trip
Car ID
Start time and date *
Duration
Covered Distance
Remaining Distance *
Total Price
tripHistory
noTrips - backToMenu
RESERVED
selectACar ()
PostValidation
checkDestinationValidity ()
<<screen>>
TripHistory$
noTrips ()
back ()
navigateTo ()
*
<<input form>>
Car Reservation
Money Saving Option (boolean)
Destionation Address
REJECTED
<<screen>>
CarDetails$
Address
Car ID
Seats
Power level
Status
error
reserveCar ()
VALIDATED
PreValidation
checkCarStatus ()
checkUserStatus ()
reserveCar

BCE Diagram:


<<boundary>>
App Access Page
login ()
createNewAccount ()
showErrors ()
<<control>>
Account Manager
validateData ()
validateCreditCard ()
validateDrivingLicence ()
addNewUser ()
<<control>>
Authentication Manager
validateUserCredentials ()
<<entity>>
User
username
password
ID
address
complete name
credit card number
credit card expiration date
driver licence number
findAll()
findByUsername()
getAUserByLogin ()
addNew ()
0 .. * (0 .. 1 current)
0 .. 1
<<boundary>>
App Home Page
searchACar ()
showMyReservations ()
showMyTrips ()
<<boundary>>
Car Search Screen
showErrors ()
showMap ()
showCarsOnMap ()
selectACar ()
insertStartingAddress ()
reserveSelectedCar ()
<<control>>
Car Search Manager
findCarsByAddress ()
getUserPosition ()
validateAddress ()
makeAReservationRequest ()
<<entity>>
Car
car ID
plate number
seats
power level
status
currentLocation
findAll()
findByCarID ()
findByLocation ()
getCarData ()
setCarStatus ()
setCarPowerLevel ()
setCarLocation ()
0 .. 1
1
1
<<entity>>
Reservation
user ID
car ID
time
destinationAddress
findAll ()
findByCarID ()
findByUserID ()
getTime ()
getDestinationAddress ()
addNew ()
delete ()
0 .. 1
0 .. 1
<<control>>
Car Reservation Manager
checkUserStatus ()
checkCarStatus ()
checkDestinationValidity ()
confirmRequest ()
rejectRequest ()
monitorExpirationConditions ()
1
1
<<boundary>>
Unlock Car/ Start Trip (popup)
unlockCarAndStartTrip ()
cancelTrip ()
<<boundary>>
My Trips Menu
showPastTripsTable ()
showCurrentTripDetails ()
startEmergencyCall ()
showErrors ()
<<entity>>
Trip
trip ID
user ID
car ID
starting point
destination point
cost
duration
start time
relatedReservation
findAll ()
findByUserID ()
getTripData ()
addNew ()
setCost ()
setDuration ()
1
<<control>
Trip Manager
updateCurrentTripDetails ()
getPastTrips ()
unlockCarAndStartTrip ()
updateCarStatus ()
deleteRelatedReservation ()
manageEmergencyCall ()
endTripAndCalculateFees ()
chargeFeesToUser ()
PAYMENT
HANDLER
<<control>
Payment Service Interface
applyPaymentToUserCreditCard ()
<<control>
Car System Interface
getDataFromCarSystem ()
CAR
SYSTEM

Hours of work:

Bachar Senno

- 28/11: 2 hours
- 29/11: 2 hours
- 2/12: 6 hours
- 5/12: 4 hours
- 6/12: 5 hours
- 10/12: 2 hours
- 11/12: 2 hours

Gianluigi Iobizzi

- 28/11 : 2 hours
- 29/11: 2 hours
- 2/12:   2 hours
- 3/12:   4 hours
- 4/12:   4 hours
- 5/12: 3 hours
- 8/12 : 2 hours
- 10/12: 4 hours
- 11/12: 3 hours

Onell Saleeby

- 21/11: 1:30hours
- 28/11: 1:30 hours
- 2/12: 1 hour
- 3 /12:1 hour
- 9/12: 2 hours

Revision History:

V1.0 Initial Version

V1.1 (10/01/2017) Added sequence diagram for trip process. Fixed front page.