PowerEnJoy

REQUIREMENT ANALYSIS AND SPECIFICATION DOCUMENT

Flavio Primo, Hootan Haji Manoochehri

POLITECNICO DI MILANO | SOFTWARE ENGINEERING 2

Index

Introduction

Purpose

The aim of this document is to describe functional and nonfunctional requirements of a system-to-be.

The most important considered aspects are: stakeholder needs, constraints of the system and operative scenarios.

This document is addressed to the customer, software engineers and developers that will implement the system hereby described.

Scope

This system, from now on called PowerEnJoy, is about a digital management system for a car-sharing service that exclusively employs electric cars. PowerEnJoy provides classical functionality found in similar services such as: user registration, search for an available car and renting a car.

Since PowerEnJoy is about electric cars, it will manage facilities to park and recharge the cars and special discounts for users with virtuous behavior in respect to the environment and other users.

Stakeholders

On the supply side, there is the **PowerEnJoy company**. This is a new business that aims at creating a new competitive electric car sharing service balancing revenues, a quality service for the users' needs and care for the environment.

On the demand side, there are the **users**. They are interested in moving freely around the city by using a hassle free, environmentally safe and rewarding car sharing service such as PowerEnJoy.

Glossary

- **PowerEnJoy company:** the company behind the PowerEnJoy system.
- **PowerEnJoy:** the system described in this document, an electric car sharing system.
- **Agents:** people that interact with PowerEnJoy.
    - **Guest:** person not yet registered or logged to PowerEnJoy. He may be able to register or login if already a user of PowerEnJoy.
    - **User:** a registered person to PowerEnJoy. He may be able to find, reserve and unlock a PowerEnJoy car.
    - **Driver:** a registered user that has successfully unlocked a car. He may drive and park a PowerEnJoy car.
- **Safe area:** predefined parking spot areas in PowerEnJoy.
    - **Normal:** a city zone in which a PowerEnJoy car can be parked in. Parking spots in such areas corresponds to normal car spots.
    - **Special:** a delimited parking area owned by PowerEnJoy company. Each parking spot of such a delimited area provides a PowerEnJoy car charger.
- **Payment information:** information needed to conclude positively a virtual money transaction with a payment provider such as: bank's credit card, PayPal.
- **PowerEnJoy car:** electric cars that constitutes the PowerEnJoy car fleet. It can only be charged in a special safe are.
- **PowerEnJoy car status:** statuses that PowerEnJoy car send to PowerEnJoy.

- o **Available:** PowerEnJoy car is ready to be reserved. A PowerEnJoy car is ready to be reserved when it is parked in a safe zone, battery is over 25%.
  - o **Reserved:** PowerEnJoy car is ready to be unlocked. A PowerEnJoy car has been reserved by a user.
  - o **In-use:** PowerEnJoy car is unlocked. A PowerEnJoy car has been unlocked by a user.
  - o **Out-of-power:** PowerEnJoy car battery charge is below 20%. A PowerEnJoy car has been parked by a user with less than 20% of battery.
- **PowerEnJoy Box:** box installed on PowerEnJoy cars that provides navigation and functionality inherent to rent the car (such as lock/unlock doors, …). It communicates with PowerEnJoy about the status of the car and authorization information.
- **MaintenanceService:** internal service in PowerEnJoy company that oversees relocating out-of-power PowerEnJoy Car in special safe area to properly plug and recharge the cars.

Product perspective

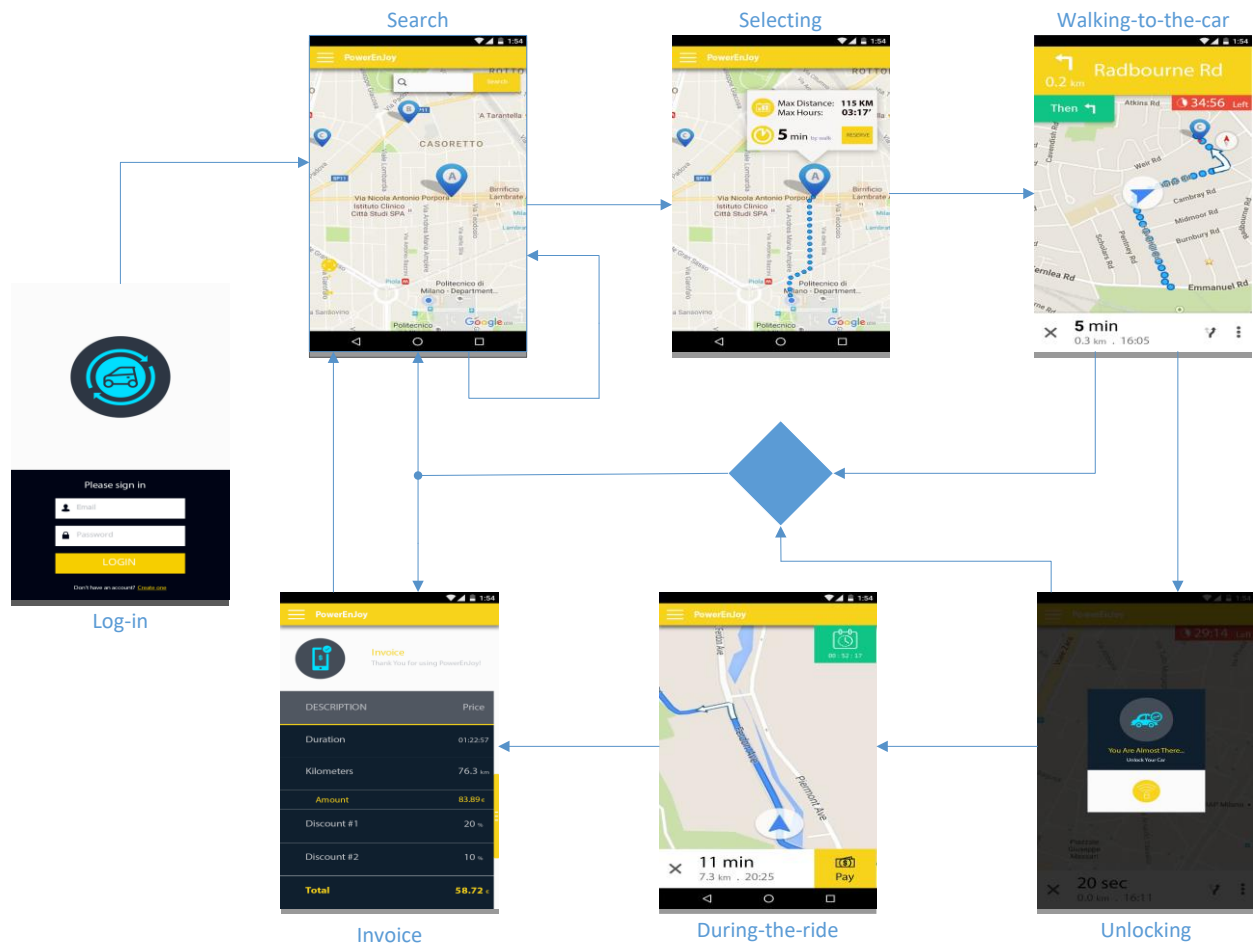The system will be composed of several parts:

- **App:** front-end mobile application used by guest and users to interact PowerEnJoy.
- **Server application:** server application that serves as a centralized service that manages PowerEnJoy resources such as PowerEnJoy cars, special safe areas and PowerEnJoy users. The app interacts solely with the server application.
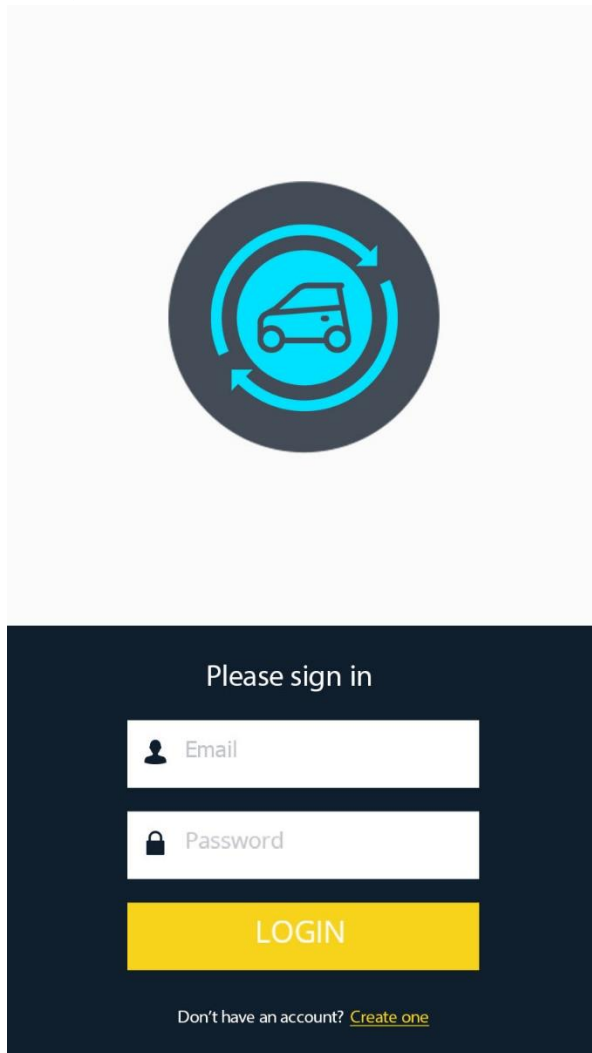
Hardware interfaces

Every car is equipped with a PowerEnJoy Box, a low-cost and low-powered single-board computer connected to a touchscreen positioned on the dashboard of the car. This single board computer should have an internet connection through 3/4G, a Bluetooth LE and a GPS.

The functionality of the box divides in 3 categories:

- **UI:** Interacting with the user via a touch screen and providing information like navigation and messages from the system (like: amount of the payment, how many minutes/kilometers the user drove, etc.).
- **Network:** it has two network cards, 1st Bluetooth to authenticate the user to unlock the car when he/she is nearby, and 3g/4g connection for communication with the server.
- **Control Unit:** it has control on the actuators of the car, and providing functionalities such as lock/unlock of the doors, starting engine, reading the status of the batteries, and other sensors like GPS, accelerometer, etc.

Software Interfaces

Storyboard

Search

Selecting

Walking-to-the-car

Log-in

Invoice

During-the-ride

Unlocking

Mockups

Mockup 1: at first launch the app presents the login screen. If the guest is registered then he does the login, otherwise he registers for a new account.
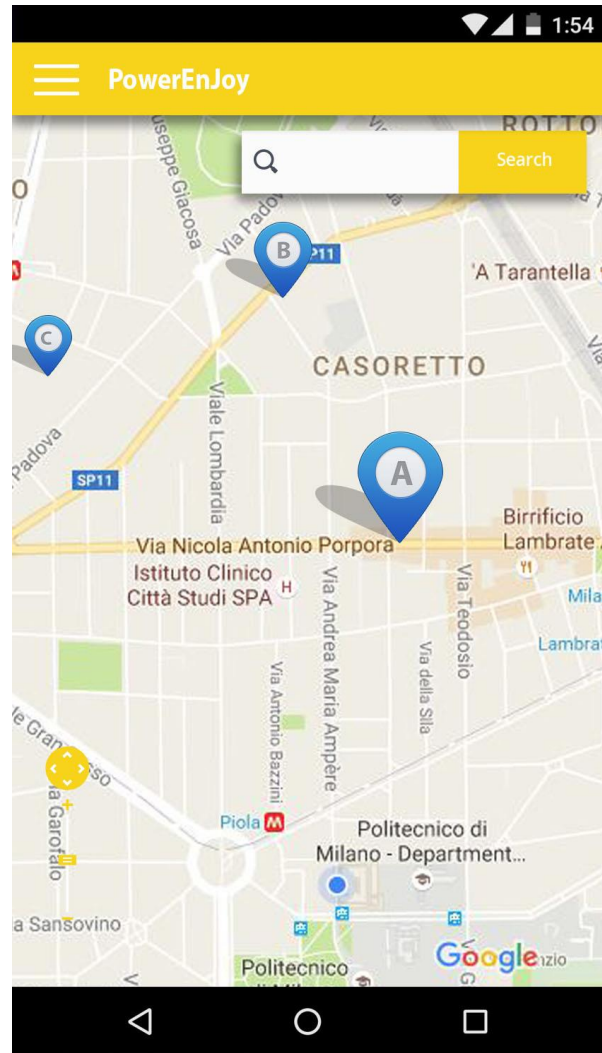
Mockup 2: choose a car to have a drive with either by clicking on the car marker or by searching for an address
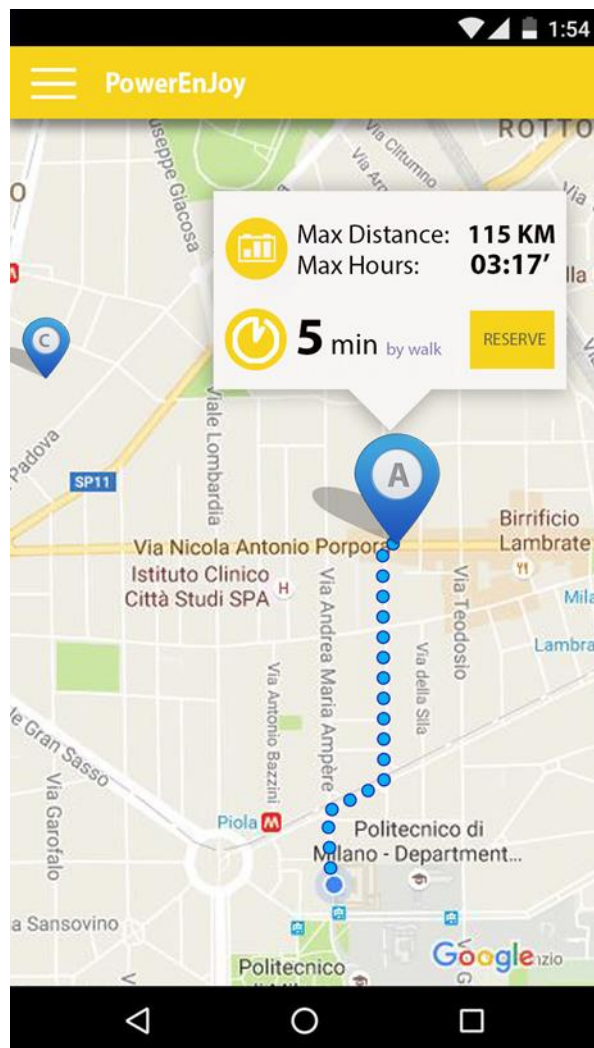
Mockup 3: once clicked on a car the application shows the route and time necessary to get to the car. It also shows the level of battery of the car in terms of maximum distance and time that the car can achieve.
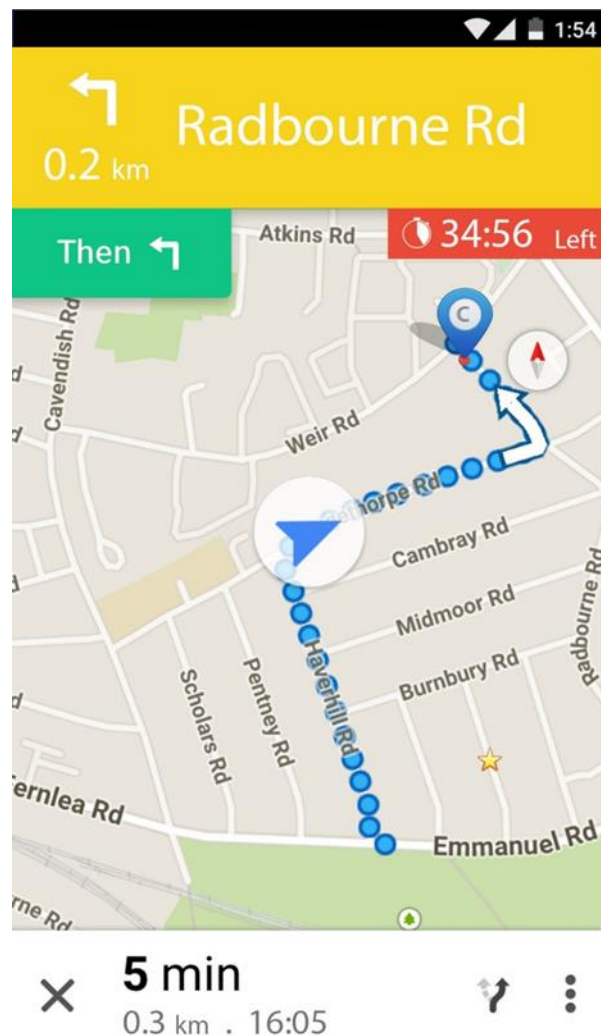
Mockup 4: once the car has been reserved, the application starts the navigation to get to the reserved car. It also shows the remaining time before the expiration of the reservation.
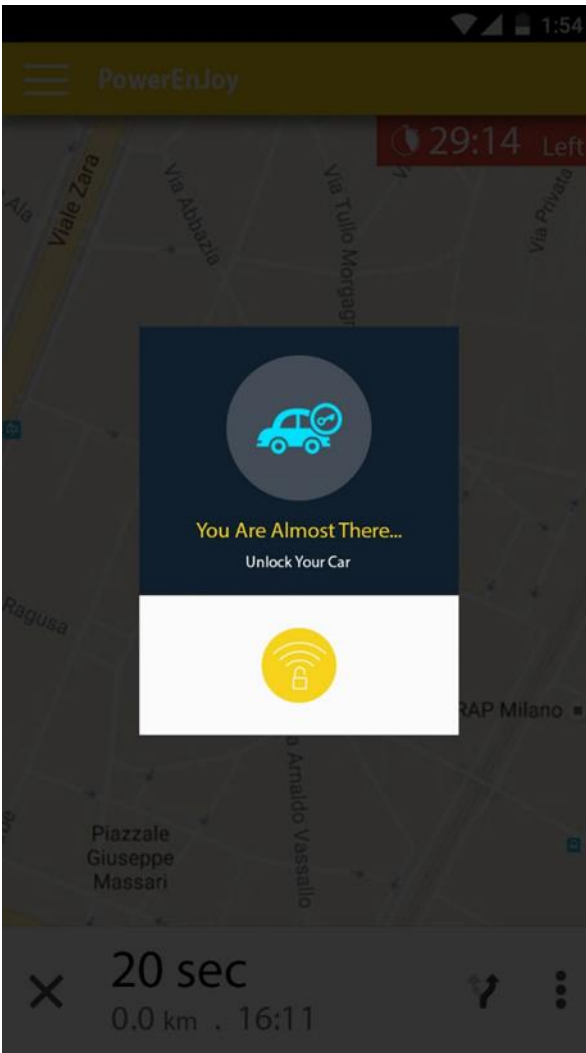
Mockup 5: when the car is near the application offer the user to unlock the car and start the ride.

Mockup 6: once the car has been unlocked the app shows the user position on the app and the pay button to start the payment and terminate the ride.

Mockup 7: once the car has been parked and the user has payed, the app shows information about the ride and the applied discounts.

Backend

- **DBMS:**
  MariaDB 10.1.14 from http://mariadb.com/
- **Programming Language:**
  Java with JavaEE 7 from http://www.oracle.com/technetwork/java/javaee/overview/index.html
- **OS:**
  Ubuntu Linux 16.04 LTS from https://www.ubuntu.com/
- **Map API:**
  Google Maps from https://developers.google.com/maps/

Frontend

- **OS:**
  iOS (client), Android (client), Windows Phone (client)

Assumptions

A list of domain assumptions the writers of this document assume to hold in the real world:

- A PowerEnJoy car cannot be parked outside of a safe area.
- A PowerEnJoy car cannot malfunction.
- The PowerEnJoy Box is provided as a third-party product.
- Every user has a smartphone equipped with Bluetooth LE, internet connection, GPS and the PowerEnJoy application installed.
- Given a place, the system can always detect the zone where it is.
- Each PowerEnJoy car is equipped with a PowerEnJoy Box.
- Only the user that has reserved for the PowerEnJoy car can drive it.
- User keeps the PowerEnJoy app open during the reservation time interval frame.
- Payment takes place when no one is inside the car and the car is parked with closed doors.
- PowerEnJoy Box can detect how many people are present in the car.
- User pays if and only if he is outside the car, the engine car is stopped, the car is parked in a safe area and all the doors and windows are closed.

Specific Requirements

Functional Requirements

Goal 1: Allow a guest to register to the platform.

- **Requirement 1**: the system prevents guest from accessing services before being registered or logged in.
- **Requirement 2**: guest can register only if he provides: identity card number, fiscal code number, driver license number and a payment information.
- **Requirement 3**: after registration, the guest is logged in as a user and the system gives a personal password to the guest to access the system.
- **Requirement 4**: the system validates input format given by the guest.

Goal 2: Allow a user to access the platform by logging in.

- **Requirement 1**: the system validates the input format of all the inputs given by the guest.
- **Requirement 2**: guest can login as user if and only if he provides a correct couple of username and password.
- **Requirement 3**: the system prevents anyone from logging more than once at a time.

Goal 3: Allow a user to find an available car.

- **Requirement 1**: the system needs the location of the user or an address.
- **Requirement 2**: the user should be able to specify a certain distance, in advance search option.
- **Domain assumption 1**: we assume that all the available cars are in the safe areas which are pre-defined.
- **Domain assumption 2**: given a place, the system can always detect the zone where it is.

Goal 4: Allow a user to reserve an available car.

- **Requirement 1**: the car should be still available.

- **Requirement 2**: a user cannot have more than one active reservation.

Goal 5: Allow a user to cancel his reservation.

- **Requirement 1**: user should be able to cancel his reservation only within the reservation time interval of one hour.

Goal 6: Allow a user to unlock a reserved car.

- **Requirement 1**: the time passed from the reservation should not exceed 1 hour, if not he will be charged 1 euro.
- **Requirement 2**: the user should be in a distance equal or less than 50 meters from the PowerEnJoy car.
- **Requirement 3**: the user should have reserved the car with PowerEnJoy.
- **Requirement 4:** the user should use the PowerEnJoy application to unlock the car.

Goal 7: Allow a user to pick the car up.

- **Requirement 1**: the user should have successfully unlocked the car.
- **Domain assumption 1**: a user only picks up the car he reserved for.

Goal 8: Allow a user to pay for the ride.

- **Requirement 1**: if more than 3 passengers in car then 10% discount on the last ride.
- **Requirement 2**: if more than 50% of battery when parked then 20% discount on the last ride.
- **Requirement 3**: if car is parked in special area and the user plug the car to the power grid then 30% discount on the last ride.
- **Requirement 4**: if car is parked in a safe area distant more than 3Km from a special station or the battery is less than 20% then charge 30% more on the last ride.
- **Requirement 5**: after payment, the car locks itself and informs the system that it has been locked.
- **Requirement 6**: user can pay only with the app with the payment information provided.

Goal 9: Allow a user to park in a safe area.

- **Requirement 1**: car can be parked if and only if it is in a safe area (normal or special) and the safe area has free parking spots.

Goal 10: Allow the user to plug the car into the grid.

- **Requirement 1**: if and only if the car is parked in a special area then the user can plug the car to the grid.
- **Domain Assumption 1**: for every parking spot in a special area there is an available plug to the grid.

Scenarios

Scenario 1: Find and reserve a PowerEnJoy car

George wants to get to his favorite restaurant located on the other side of the city, so he decides to go there by using the PowerEnJoy car sharing service. George since is already registered to the service, he launches the PowerEnJoy app on his smartphone and login.

George selects a car near him and the application then shows a route to get to that car based on George current position. George agrees with the solution presented and clicks the button "Reserve". The system updates the car status to "reserved" (from the status "available") and now the car is reserved for George. George has up to 1 hour to either unlock and pick-up the car or cancel the reservation.

Scenario 2: Cancel reservation

Freddie has an appointment with a dentist and he reserved a car with his PowerEnJoy App. Suddenly, he remembers that his appointment is tomorrow and not today. So, he opens the app up and go on his reservation and simply push the "Cancel reservation" button. The system changes the status of the car from "reserved" to "available", and other users can reserve the car immediately.

Scenario 3: Unlock and pick up the car

Peter has reserved a PowerEnJoy car, so he walks guided by the PowerEnJoy app to the car. When he sees the car (about 15 meters' distance), he clicks the button "Unlock". The phone app authenticates securely with the PowerEnJoy Box via Bluetooth. Now the cars unlock the doors and starts up the car touchscreen. The touchscreen displays the navigation system, the elapsed time from the doors unlocking.

Peter opens the car doors and sits on the driver seat. To start the engine, he presses the button "start engine" on the car dashboard.

Scenario 4: Pay for the ride

Simon, an Erasmus student from Belgium, used PowerEnJoy to go to the supermarket to buy a lot of groceries for his birthday party. He drives back home and parks in a safe area near his home. He gets off the car with the groceries and then he pushes the "I'm done" button. The system shows him a bill and after applying a discount (if is there any) George gets charged. He then gets a report of the status of the payment on his phone (the system uses his payment information that he entered during the registration to the system). The car then locks itself and the car is now available to the other users.

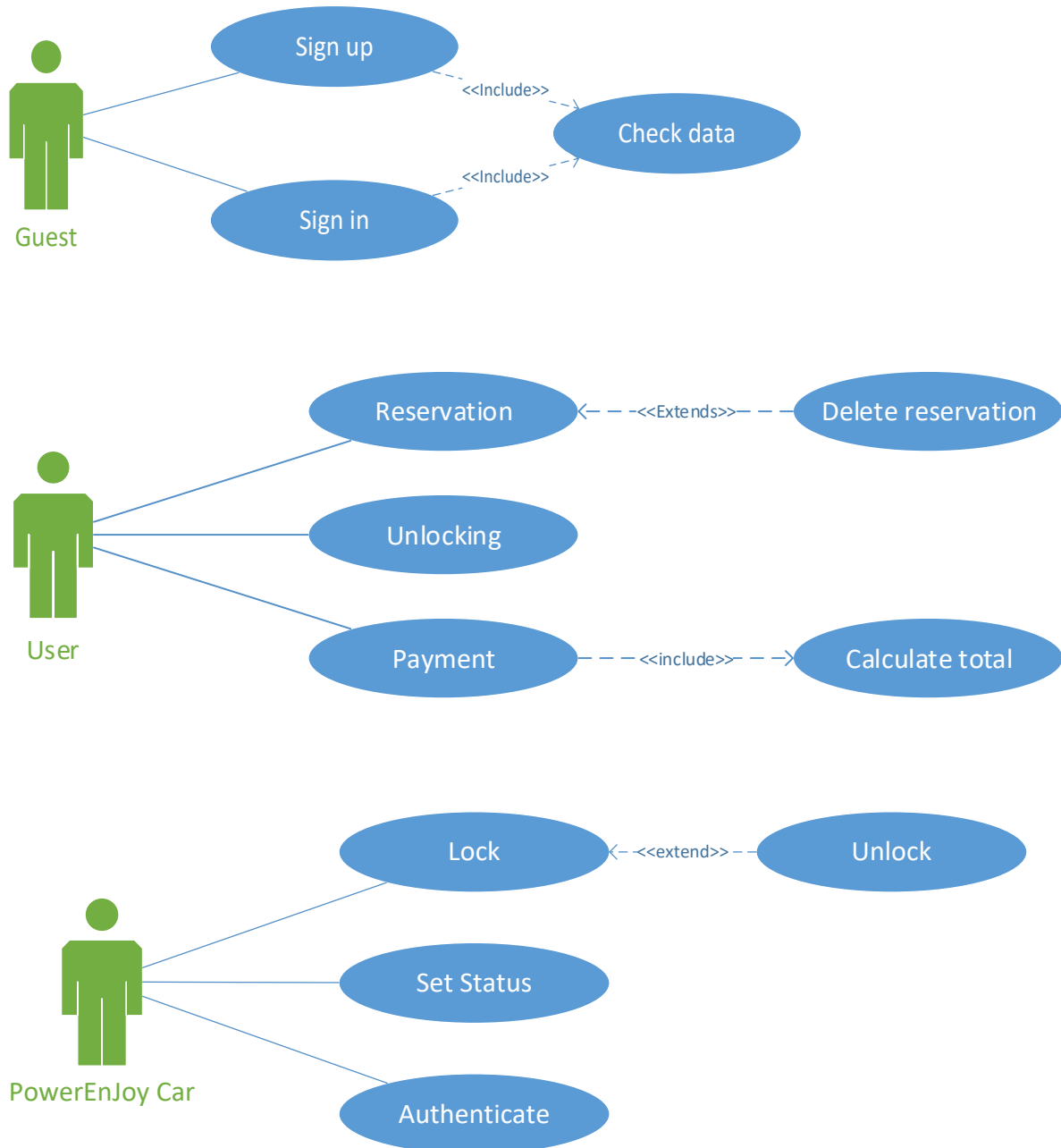### Scenario 5: Park car in a safe normal area

Tommy used PowerEnJoy to go to a very important business meeting with customers of his company in a restaurant near city center because it is the fastest solution for him. He drives to his destination by using the navigation system on the car. When he arrives there he can see a safe area near the restaurant on the screen and since the car has 56% battery full, he decides to park the car there and get 20% discount.

Scenario 6: Park car in a safe special area

Alice is driving a PowerEnJoy car and she is near her destination. She looks on the map that there is a special safe area nearby, so she decides to park the car there to use a discount if she plugs the car to the grid once parked. Alice notices that there is an available parking spot in the special safe area, so she

parks the car there. She stops and exits the car and she plugs the car to the grid. Alice then clicks on the "Pay" button on the PowerEnJoy app. The apps calculate the trip cost and applies a 30% discount.
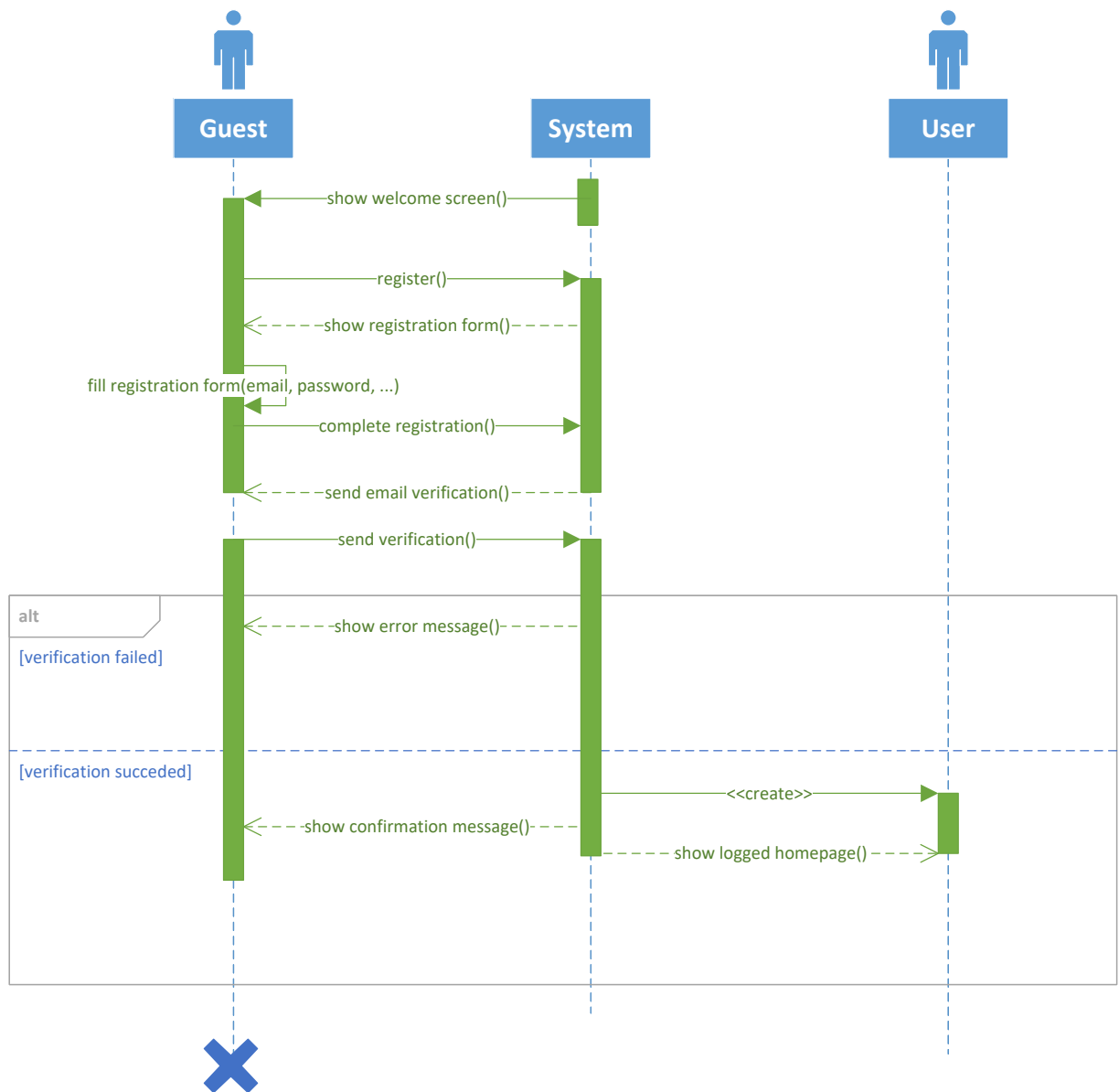
Use cases and UML diagrams

Use Case 1

Name	Sign Up
Description	Registration to the system by a user-to-be.
Primary Actor	Guest

Basic Flow	1. The guest opens PowerEnJoy app and he is greeted by a welcome screen. 2. The guest clicks on the button "Register". 3. The system outputs the form on the screen. 4. The guest inserts his: name, email, password, phone number, identity card number, fiscal code number, driver license number and a payment information and the clicks the button "Complete registration". 5. The system checks the syntax of the inputs and, with a third-party server, the payment information correctness. If the checks are passed the system sends a confirmation email to the user. 6. The guest verifies his account by clicking on the verification link sent by email. 7. The guest is now a registered user to PowerEnJoy.
Alternate Flow	/
Exception	• The guest is already registered. • One or more fields are not well-formed. • Email is already in use. • Payment information is not correct. • Validation link is no more valid (after 24 hours).

Sequence Diagram 1: Sign Up

Use Case 2

Name	Sign In
Description	User sign into the PowerEnJoy app
Primary Actor	Guest, User
Basic Flow	1. The guest opens PowerEnJoy app and he is greeted by a welcome screen. 2. Guest will sign-in with credentials (email and password) that he previously provided in the sign-up page 3. System will check if the credentials are correct then system will recognize him as a user from now on

	4. System redirect the user to the main page of the app and shows a map with nearby available PowerEnJoy cars.
Alternate Flow	/
Exception	• Email or password are incorrect • A wrong format input • Email not found (the registration is not done yet)
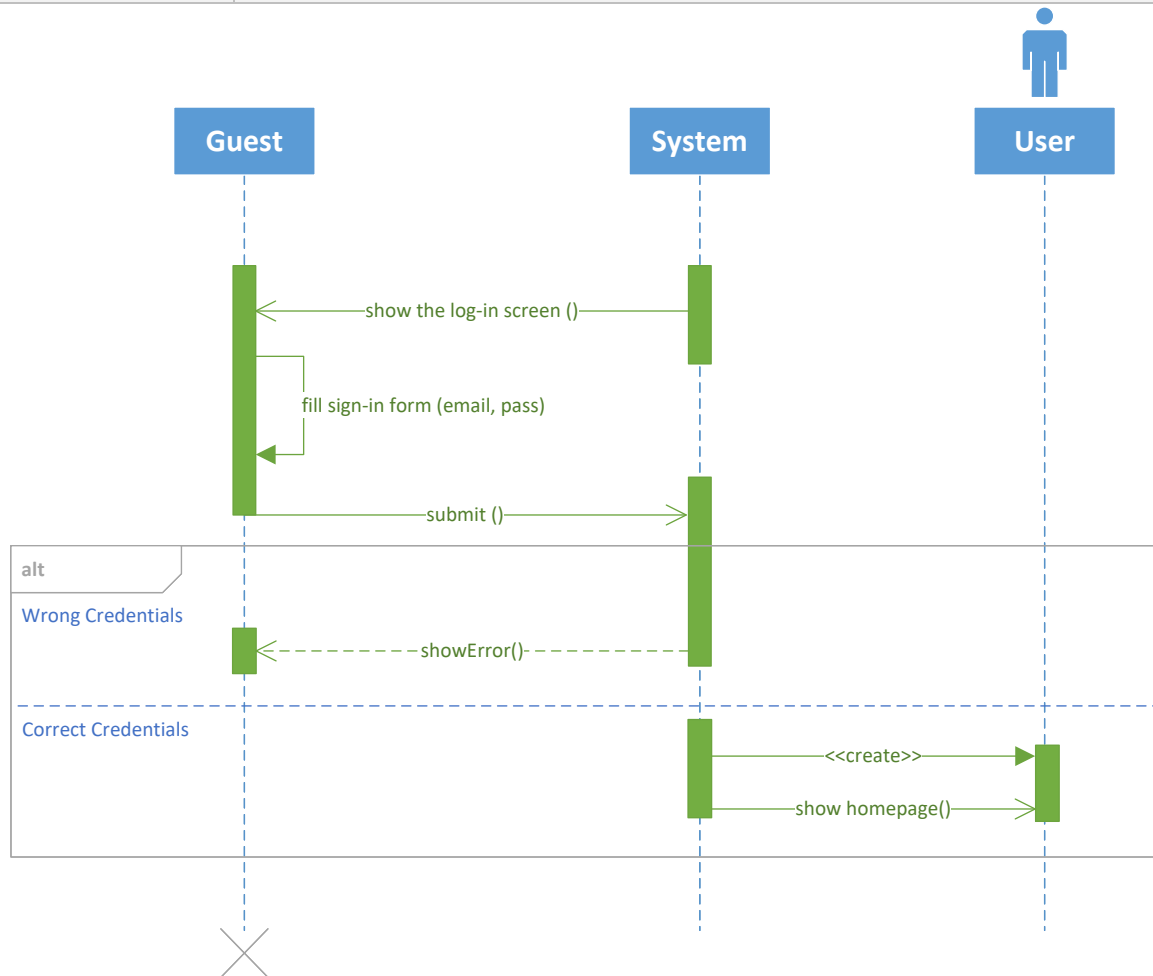
Sequence Diagram 1: Login

Use case 3

Name	Reservation
Description	Reservation of a PowerEnJoy car.
Primary Actor	User, PowerEnJoy car
Basic Flow	1. The user launches the PowerEnJoy app (in which he already signed in). 2. The system shows a map centered on the user with a range of 300 meters and a search box with the button "Find". In the map are showed only the cars with the "Available" status. 3. The user clicks the car he wants to reserve.

	4. The system calculates the position of the user and consequently the position of the selected car. The system draws the route from the user to the selected car.
	5. The user accepts the car and clicks on the button "Reserve".
	6. The system creates a unique authentication key and shares it with the PowerEnJoy car and the PowerEnJoy app of the user.
	7. The car is now reserved to the user and the car's status changes to "Reserved".
Alternate Flow	**Search for a car**
	1. Same operations as before point 3.
	2. The user inserts an address on the search box and clicks the button "Find".
	3. The system checks if it's a valid address; if so the system search and select for the address and shows the user a map centered on the given address with the available PowerEnJoy cars.
	4. The alternate flow continues with point 3 with the selected car.
Exception	• The address is not valid.
	• A car is not available at the specified address.
	• The user abort the operation.
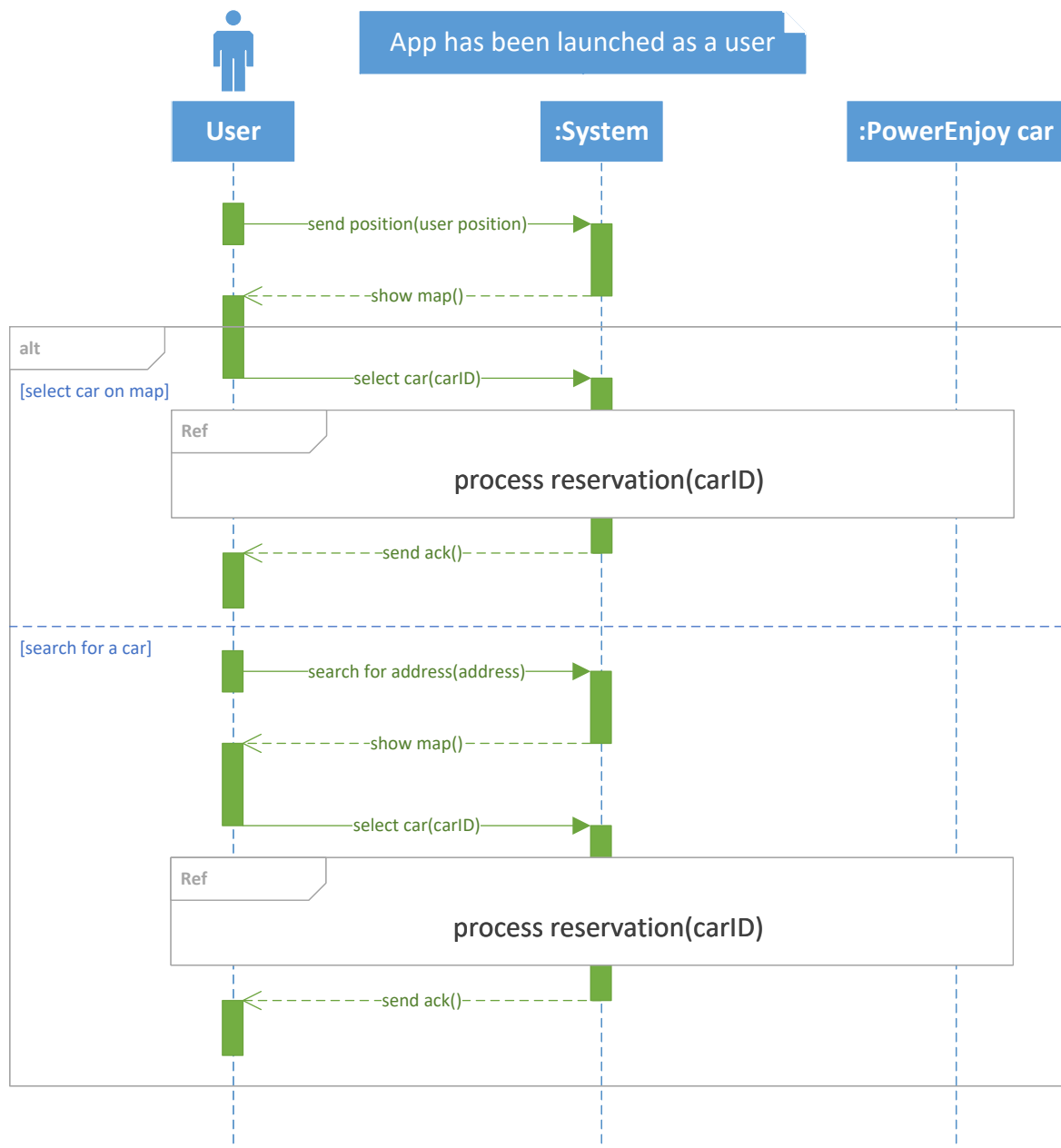
Sequence Diagram 3: Reservation
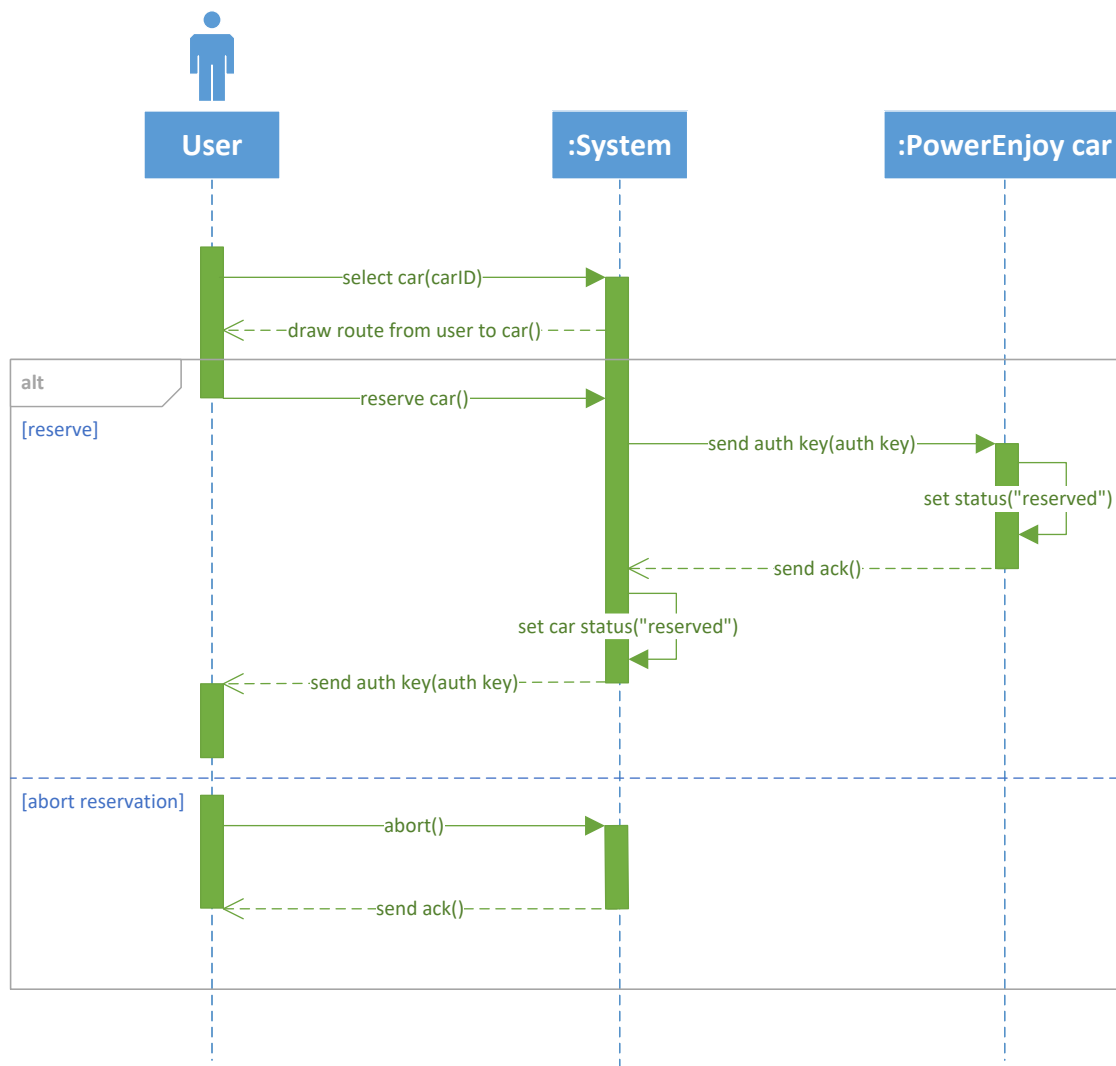
Sequence Diagram 4: Process Reservation
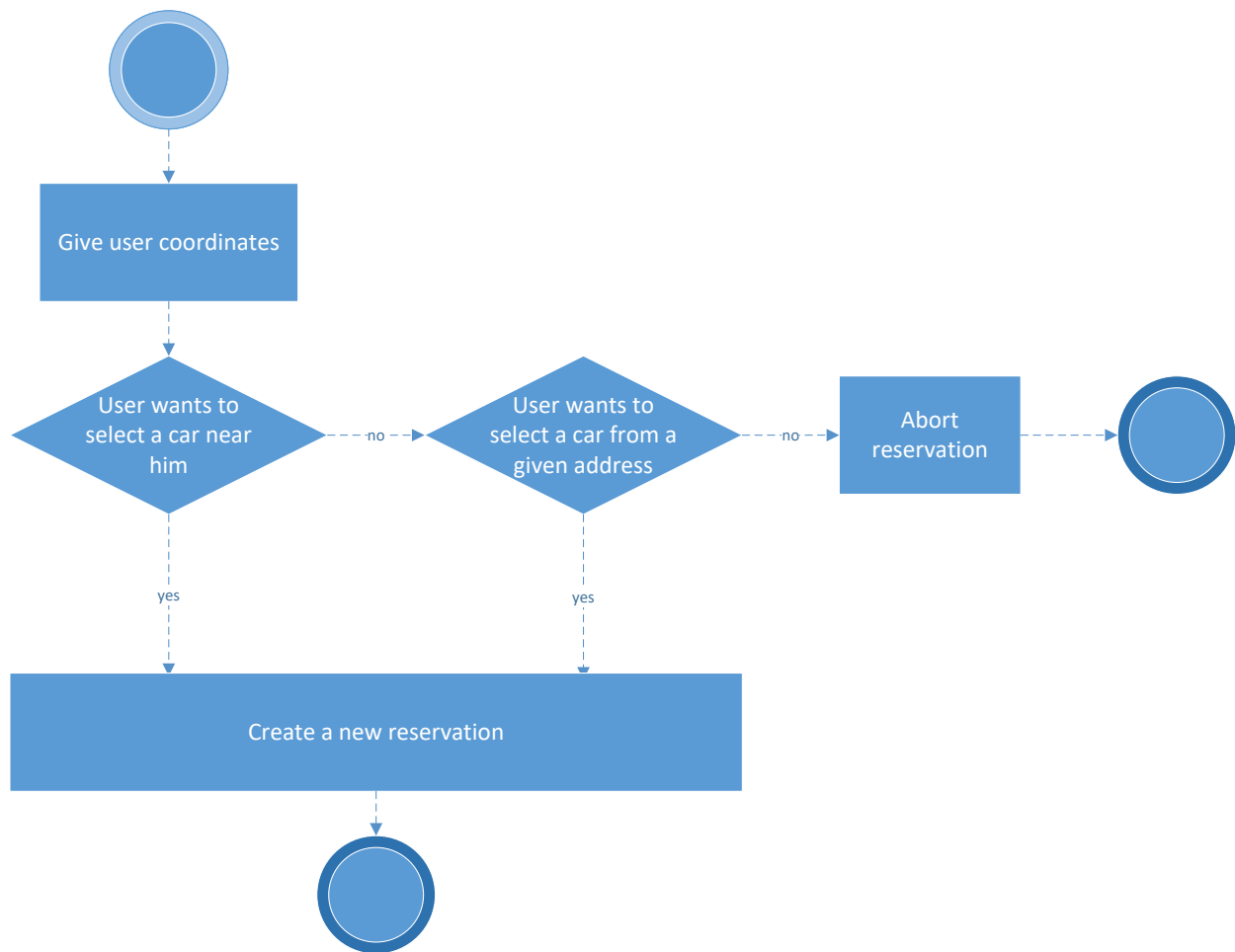
FSM Diagram 1: Reservation

Use case 4

Name	Cancel Reservation
Description	The registration of the user cancelled
Primary Actor	User, PowerEnJoy car
Basic Flow	1. User clicks the "Cancel reservation" button on the PowerEnJoy app. 2. The system sends a confirmation message to the user about cancelling the reservation. 3. The user clicks on the confirmation button. 4. The system cancel the reservation. The systems notify the user and the PowerEnJoy car about the cancel of the reservation. 5. The PowerEnJoy car set his status to "Available" and notifies the system. 6. The system set car status to "Available"
Alternate Flow	1. After 1 hour, the system automatically cancels the reservation. 2. System charges the user 1 euro.

	3. System Notifies the user of the cancellation and his payment.
Exception	• The user cancels his request when he got prompt.
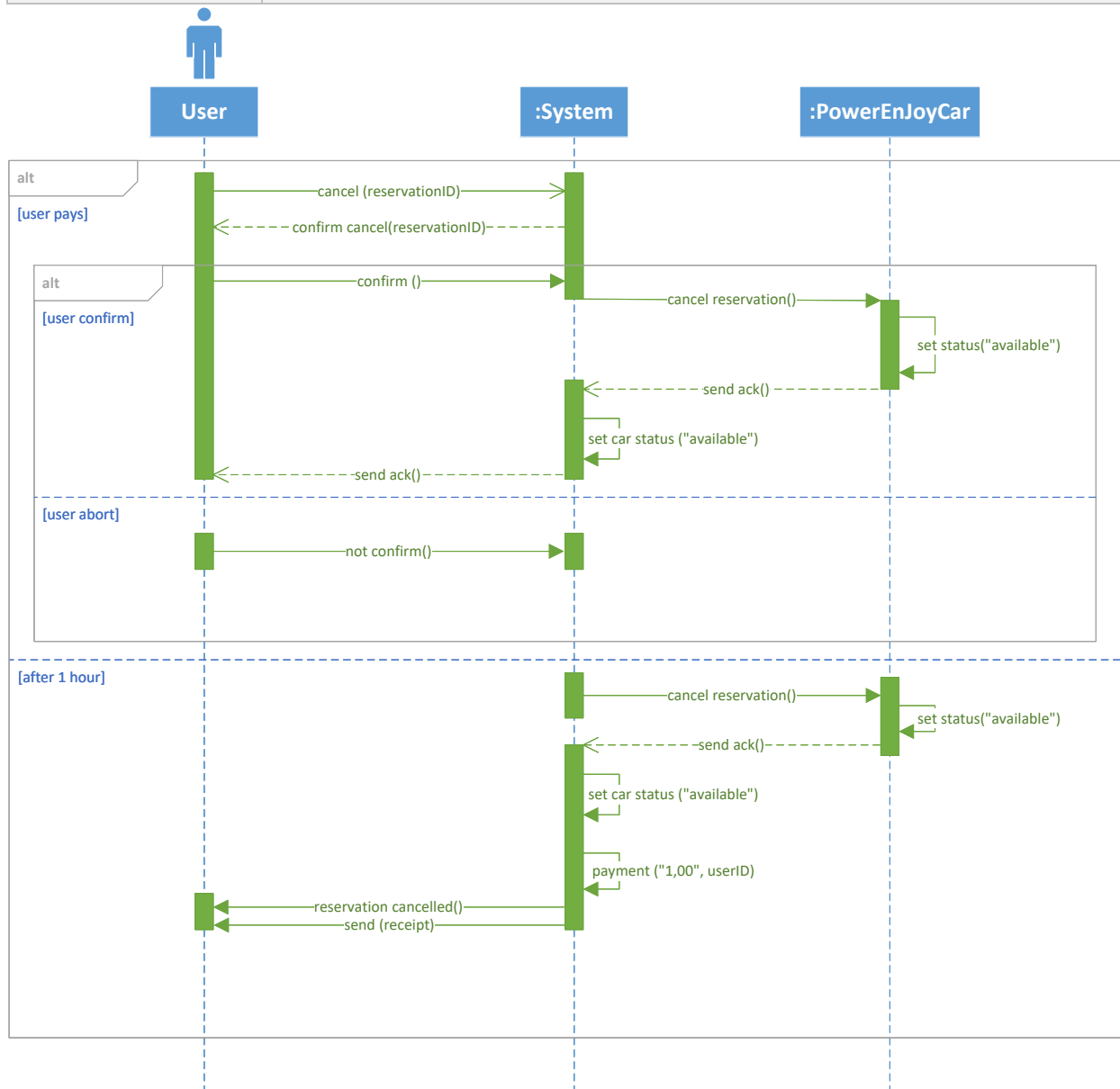
Sequence Diagram 5: Cancel Reservation

Use case 5

Name	Unlocking
Description	User unlocks the car.
Primary Actor	User, PowerEnJoy car
Basic Flow	1. The PowerEnJoy car detects that the user (with the PowerEnJoy app open) is within 50 meters from the car. The PowerEnJoy car asks the PowerEnJoy app of the user for the authentication code.

	2. The PowerEnJoy app shows to the user an "Unlock" button and the user clicks it. The PowerEnJoy app sends back to the PowerEnJoy car the authentication code. 3. The PowerEnJoy car validates the authentication code and if correct: • unlocks the car doors. • powers on the touchscreen of the PowerEnJoy car with the navigation application. • starts the count for the car usage. • changes the car status to "In-use" and notifies the system and the PowerEnJoy app that it has been unlocked.
Alternate Flow	/
Exception	/

Sequence Diagram 6: Car unlocking

Use case 6

Name	Payment
Description	User pay for his ride.
Primary Actor	User, PowerEnJoy car
Basic Flow	1. user ask for payment via app

	2. System asks the PowerEnJoy car to report his status (battery percentage, location, is it plugged or not, etc.).
	3. PowerEnJoy car send its state to the system.
	4. System calculates the total cost based on the information provided (considering the applicable discounts), and charges the user accordingly.
	5. System sends to the PowerEnJoy car the new state.
	6. System notifies the user, and send him the receipt.
Alternate Flow	/
Exception	• The car is not parked. • The engine is still working. • Car has less than 20% battery to be available.
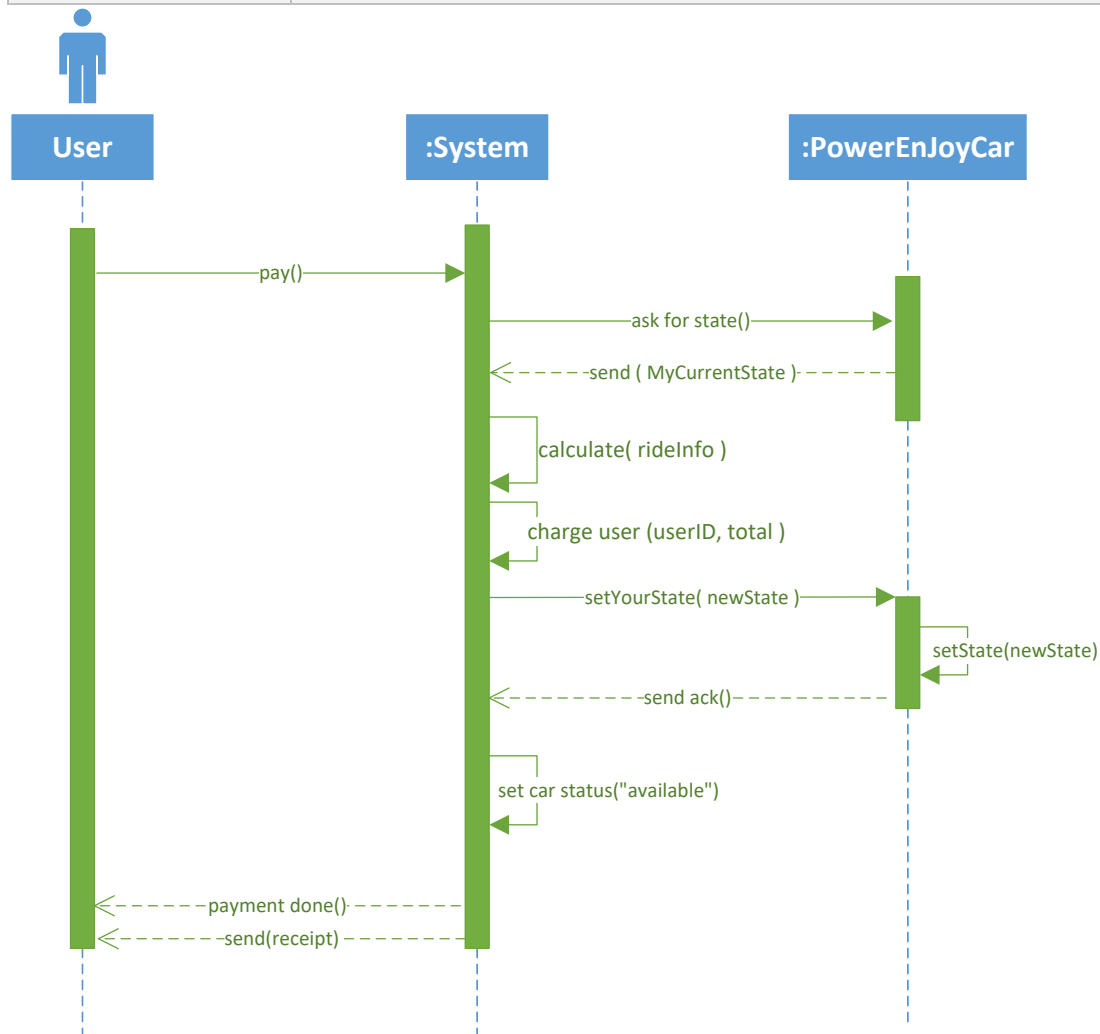
Sequence Diagram 7: Car payment
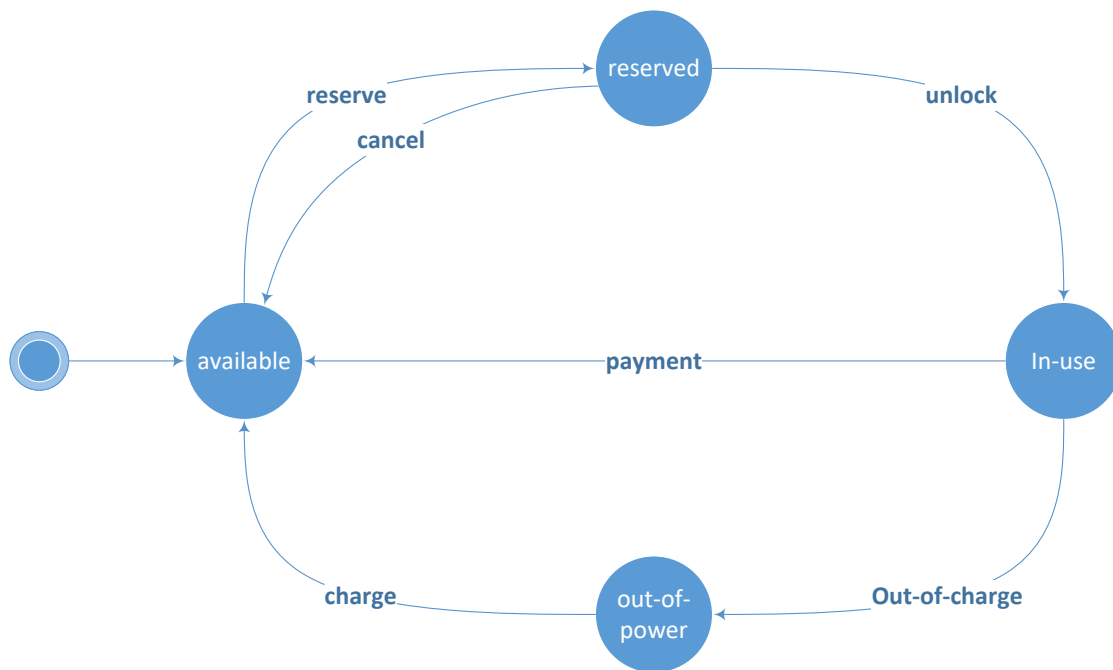
FSM Diagram 2: Car status

Class Diagram
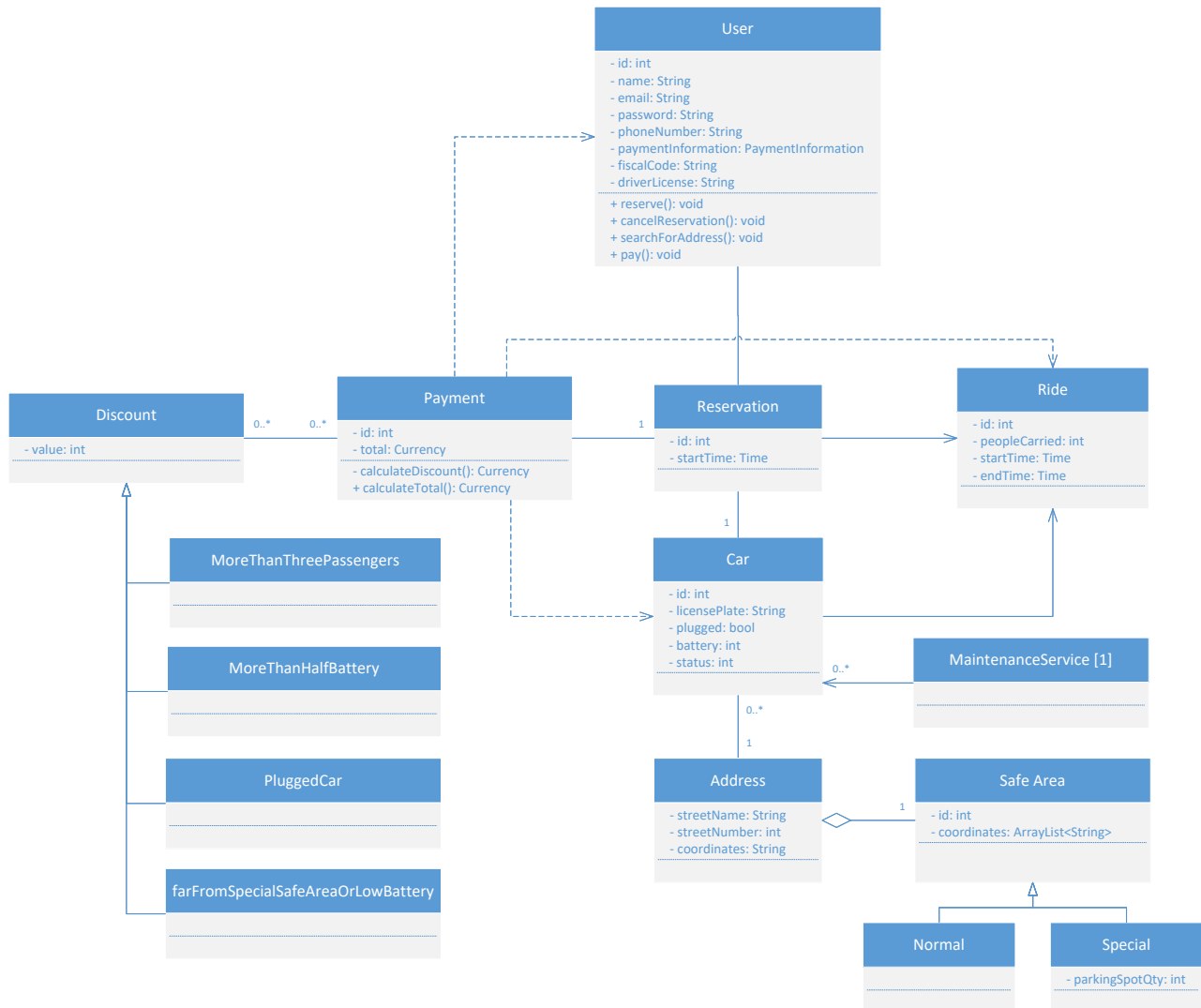
Alloy

Alloy modeling

Alloy modeling code with signatures, facts, predicates and run commands.

```
/** SIGNATURES **/
/* Default Classes */
sig Stringa {}

sig Currency {}

sig Time {}

/* Custom Classes */
sig PaymentInformation {}

sig User {
```

```
        id: one Int,
        name: one Stringa,
        email: one Stringa,
        password: one Stringa,
        phoneNumber: one Stringa,
        paymentInformation: one PaymentInformation,
        fiscalCode: one Stringa,
        driverLicense: one Stringa,

        currentReservation: lone Reservation
} {
        id > 0
        email != password
}

sig Payment {
        id: Int,
        total: lone Currency,

        reservation: one Reservation
} {
        id > 0
}

sig Reservation {
        id: Int,
        startTime: one Time,

        user: one User,
        payment: one Payment,
        car: one Car,
        ride: lone Ride
} {
        id > 0
        car.status = INUSE or car.status = RESERVED
}

sig Ride {
        id: one Int,
        peopleCarried: one Int,
        startTime: one Time,
        endTime: lone Time, // zero if Ride isn't finished yet
} {
        id > 0
        peopleCarried > 0 and peopleCarried < 6
        startTime != endTime
}

sig Car {
        id: one Int,
        licensePlate: one Stringa,
```

```
        status: one CarStatus,
        plugged: Bool,
        battery: Int,

        currentRide: lone Ride,
        currentReservation: lone Reservation,
        currentAddress: one Address
} {
        id > 0
        // battery range from 0 (0% of battery charge) to 6 (100% of battery charge)
        battery >= 0 and battery <= 6
        // 1 is considered 20% of battery charge
        (battery < 1 and status = OUTOFPOWER) or
                (battery > 0 and (status = AVAILABLE or status = RESERVED or status =
INUSE))
}

sig Address {
        streetName: one Stringa,
        streetNumber: one Int,
        coordinates: one Stringa,

        car: set Car
} {
        streetNumber > 0
}

abstract sig SafeArea {
        id: one Int,
        Coordinates: set Stringa,

        address: set Address
} {
        id > 0
        #address > 0
}

sig Normal extends SafeArea {}

sig Special extends SafeArea {
        parkingSpotQty: one Int
} {
        id > 0
        parkingSpotQty > 0
}

one sig MaintenanceService {
        car: set Car
}

abstract sig Discount {
```

```
        payment: set Payment
}

lone sig moreThanThreePassengers extends Discount {}

lone sig moreThanHalfBattery extends Discount {}

lone sig carPluggedInSpecialSafeArea extends Discount {}

// more than 3Km constraint is not modelled
lone sig moreThanThreeKmToSpecialSafeAreaOrBatteryLessThanTwenty extends Discount {}

/* Enums */
enum CarStatus {
        AVAILABLE,
        RESERVED,
        INUSE,
        OUTOFPOWER
}

enum Sign {
        POSITIVE,
        NEGATIVE
}

enum Bool {
        TRUE,
        FALSE
}

/** FACTS **/
/* Duplicated instances */
// each User is not duplicated
fact noDuplicatedUsers {
        no u1, u2 : User |
                u1 != u2 and
                (u1.id = u2.id or // unique id
                u1.fiscalCode = u2.fiscalCode or // unique fiscalCode
                u1.email = u2.email or // unique email
                u1.driverLicense = u2.driverLicense) // unique driverLicense
}

// each Payment is not duplicated
fact noDuplicatedPayment {
        no p1, p2: Payment | p1 != p2 and p1.id = p2.id
}

// each Reservation is not duplicated
fact noDuplicatedReservation {
        no r1, r2: Reservation | r1 != r2 and r1.id = r2.id
}
```

```
// each Ride is not duplicated
fact noDuplicatedRide {
      no r1, r2: Ride | r1 != r2 and r1.id = r2.id
}

// each Car is not duplicated
fact noDuplicatedCar {
      no c1, c2: Car | c1 != c2 and
            (c1.id = c2.id or // unique id
            c1.licensePlate = c2.licensePlate) // unique licensePlate
}

// each SafeArea is not duplicated
fact noDuplicatedSafeArea {
      no s1, s2: SafeArea | s1 != s2 and s1.id = s2.id
}

// each Address is not duplicated
fact noDuplicateAddress {
      no a1, a2: Address | a1 != a2 and a1.coordinates = a2.coordinates
}

/* Relations */
// oneToOne biijective relation between Payment and Reservation
fact samePaymentAsReservation {
      all p: Payment | p.reservation.payment = p
}

// oneToOne biijective relation between Reservation and Car
fact sameReservationAsCar {
      all r: Reservation | r.car.currentReservation = r
      all c: Car | #{c.currentReservation} > 0 implies c.currentReservation.car = c
}

// oneToMany biijective relation between Car and Address
fact sameCarAsAddress {
      all c: Car | c in c.currentAddress.car

      all a: Address |
            let carWithThisAddress = {c: Car | c.currentAddress = a} |
            a.car = carWithThisAddress
}

// oneToOne biijective relation between Reservation and User
fact sameReservationAsUser {
      all r: Reservation | r.user.currentReservation = r
}

/* Domain Rules */
// PaymentInformation are present in the same number as the Users
```

```
fact nameNumberPaymentInformationAndUsers {
    #PaymentInformation = #User
}

// Users don't share PaymentInformation
fact noSharedPaymentInformationByUsers {
    no u1, u2: User | u1 != u2 and u1.paymentInformation = u2.paymentInformation
}

// Users don't share Reservations
fact noSharedCurrentReservationByUsers {
    no u1, u2: User | u1 != u2 and u1.currentReservation = u2.currentReservation
}

// Reservations don't share Cars
fact noSharedCarByReservations {
    no r1, r2: Reservation | r1 != r2 and r1.car = r2.car
}

// Reservations don't share Rides
fact noSharedRideByReservations {
    no r1, r2: Reservation | r1 != r2 and r1.ride = r2.ride
}

// SafeAreas don't share Addresses
fact noShareAddressBySafeArea {
    no s1, s2: SafeArea | s1 != s2 and #(s1.address & s2.address) > 0
}

// MaintenanceService is connected with all Car that are OUTOFPOWER and are not
plugged to the grid
fact maintenanceServiceConnectedOnlyWithOutOfPowerCars {
    all m: MaintenanceService, c: Car | (c.status = OUTOFPOWER and c.plugged =
FALSE) implies
            c in m.car else !(c in m.car)
}

// Only Car that are INUSE can be connected to a Reservation with a Ride
fact onlyInUseCarHasARide {
    no c: Car | (c.status = OUTOFPOWER or c.status = AVAILABLE or c.status =
RESERVED) and
    #c.currentRide > 0
}

// RESERVED Car can't have a Ride
fact reservationWithRideHasInUseCar {
    all r: Reservation | r.car.status = RESERVED implies
            #r.ride = 0 else #r.ride > 0
}

// every Ride belongs to a Reservation
```

```
fact rideShouldBeReferenced {
      #{c: Car | c.status = INUSE} = #Ride
}

// every parked Car (so with status AVAILABLE or OUTOFPOWER) is parked in a SafeArea
fact carsAreParkedInASafeArea {
      all c: Car | (c.status = AVAILABLE or c.status = OUTOFPOWER) implies
            one s: SafeArea | c.currentAddress in  s.address
}

// there can't be more Car than available ParkingSpotQty in a Special SafeArea
fact numberOfCarLessThanSpecialSafeAreaParkingSpotQty {
      all s: Special, c: Car | c.status != INUSE implies
            #((c.currentAddress)->(s.address)) <= s.parkingSpotQty
}

// a Car can be plugged only in a Special SafeArea
fact pluggedCarOnlyInSpecialSafeArea {
      all c: Car, s: Special | c.currentAddress in s.address implies
            (c.plugged = TRUE or c.plugged = FALSE) else c.plugged = FALSE
}

/* Existential */
// for PowerEnJoy to work it is needed at least one PowerEnJoy Car
fact atLeastACar {
      #Car > 0
}

// for PowerEnJoy to work it is needed at least one Special SafeArea
fact atLeastASpecialSafeArea {
      #Special > 0
}

/* Discounts */
// apply a discount of -10% if there are more than 3 people in a Car
fact moreThanThreePassengersDiscount {
      no d: moreThanThreePassengers, p: Payment |
            p.reservation.ride.peopleCarried <= 3 and p in d.payment
}

// apply a discount of -20% if Car's battery is greater than 50%
fact moreThanHalfBatteryDiscount {
      no d: moreThanHalfBattery, p: Payment |
            p.reservation.car.battery <= 3 and p in d.payment // range of battery is
0 to 6 where 3 is half
}

// apply a discount of -30% if Car is plugged in a Special SafeArea
fact  carPluggedInSpecialSafeAreaDiscount {
      no d: carPluggedInSpecialSafeArea, p: Payment |
            p.reservation.car.plugged = FALSE and p in d.payment
```

```
        }

        // apply a discount of +30% if a Car is parked with less than 20% of battery charge
        fact moreThanThreeKmToSpecialSafeAreaOrBatteryLessThanTwentyDiscount {
                no d: moreThanThreeKmToSpecialSafeAreaOrBatteryLessThanTwenty, p: Payment, s:
        Special |
                        p.reservation.car.battery > 1 and p.reservation.car.currentAddress in
        s.address and
                        p in d.payment // range of battery is 0 to 6 where 1 is considered 20%
        }

        /** ASSERTIONS **/
        // all INUSE Car should be connected with a Reservation with a Ride
        assert inUseCarsHaveARide {
                no c: Car | c.status = RESERVED and #{c.currentReservation.ride} > 0
        }

        // AVAILABLE or OUTOFPOWER should not be Reserved
        assert noOutOfPowerOrAvailableCarReserved {
                no c: Car, u: User | (c.status = AVAILABLE or c.status = OUTOFPOWER) and
                        u.currentReservation.car = c
        }

        // the number of the Reservations should be less or equal to the number of the Users
        assert reservationNumberLessOrEqualThanUsers {
                #Reservation <= #User
        }

        // the number of the Payments should be less or equal to the number of the
        Reservations
        assert paymentNumberLessOrEqualThanReservation {
                #Payment <= #Reservation
        }

        /** PREDICATES **/
        pred maintenanceServiceServedCar {
                #{c: Car | c.status = OUTOFPOWER} = 1
                #{MaintenanceService.car} = 1
        }

        pred show {
                #{c: Car | c.status = AVAILABLE} > 0
                #{c: Car | c.status = RESERVED} > 0
                #{c: Car | c.status = INUSE} > 0
                #{c: Car | c.status = OUTOFPOWER} > 0

                #{c: Car | c.plugged = TRUE} > 0
                #{c: Car | c.plugged = FALSE} > 0

                #User > 2
                #Reservation > 0
```

```
        #MaintenanceService.car > 0
        #{d: Discount | #{d.payment} > 0} > 2
        #SafeArea > 3
}

pred generalRasdExample {
        #Car = 3
        #User = 2
        #Reservation > 0
        #MaintenanceService.car = 0
        #{d: Discount | #{d.payment} > 0} > 0
        #SafeArea = 2
        #Address = 3
}

pred discountsAndMaintenanceServiceRasdExample {
        #{d: Discount | #{d.payment} > 0} = 2
        #MaintenanceService.car = 2
        #Address = 2
        #Car = 3
}

/** RUN AND CHECKS **/
// OK
check inUseCarsHaveARide for 15

// OK
check noOutOfPowerOrAvailableCarReserved for 15

// OK
check reservationNumberLessOrEqualThanUsers for 15

// OK
check paymentNumberLessOrEqualThanReservation for 15

// OK
run maintenanceServiceServedCar for 15

// OK
run show for 20

// OK
run generalRasdExample for 20

// OK
run discountsAndMaintenanceServiceRasdExample for 20
```

Assertion checks

Alloy Analyzer 4.2_2015-02-22 (build date: 2015-02-22 18:21 EST)

Executing "Check inUseCarsHaveARide for 15"
Solver=minisat(jni) Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
600838 vars. 8626 primary vars. 2102130 clauses. 2474ms.
No counterexample found. Assertion may be valid. 752ms.

Executing "Check noOutOfPowerOrAvailableCarReserved for 15"
Solver=minisat(jni) Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
600901 vars. 8641 primary vars. 2102110 clauses. 2148ms.
No counterexample found. Assertion may be valid. 942ms.

Executing "Check reservationNumberLessOrEqualThanUsers for 15"
Solver=minisat(jni) Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
600267 vars. 8611 primary vars. 2100353 clauses. 1601ms.
No counterexample found. Assertion may be valid. 140ms.

Executing "Check paymentNumberLessOrEqualThanReservation for 15"
Solver=minisat(jni) Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
600267 vars. 8611 primary vars. 2100353 clauses. 1507ms.
No counterexample found. Assertion may be valid. 323ms.

Executing "Run maintenanceServiceServedCar for 15"
Solver=minisat(jni) Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
600269 vars. 8611 primary vars. 2100361 clauses. 1519ms.
Instance found. Predicate is consistent. 2606ms.

Executing "Run show for 20"
Solver=minisat(jni) Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
1687499 vars. 14076 primary vars. 6219738 clauses. 6665ms.
Instance found. Predicate is consistent. 11961ms.

Executing "Run generalRasdExample for 20"
Solver=minisat(jni) Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
1686836 vars. 14076 primary vars. 6217255 clauses. 4278ms.
Instance found. Predicate is consistent. 3265ms.

Executing "Run discountsAndMaintenanceServiceRasdExample for 20"
Solver=minisat(jni) Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
1686440 vars. 14076 primary vars. 6215655 clauses. 3868ms.
Instance found. Predicate is consistent. 3523ms.

Assertions and predicates run in Alloy Analyzer
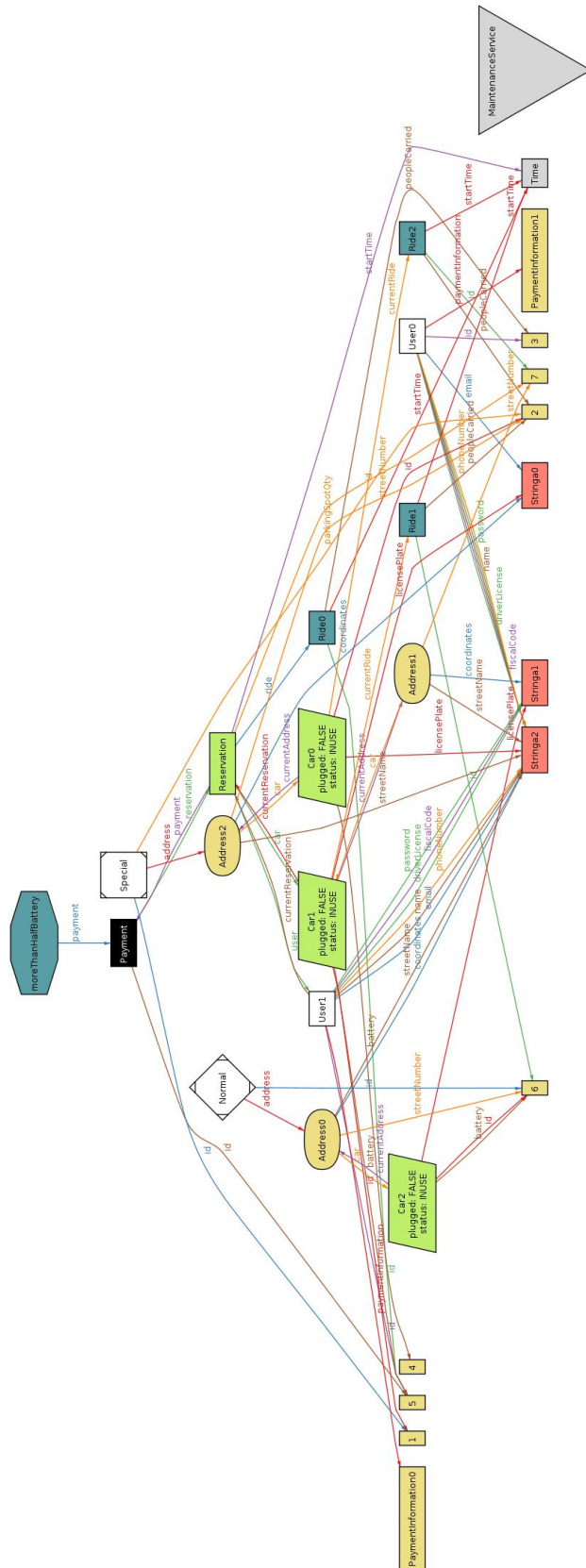
Generated worlds

Generated World 1: world with a general case
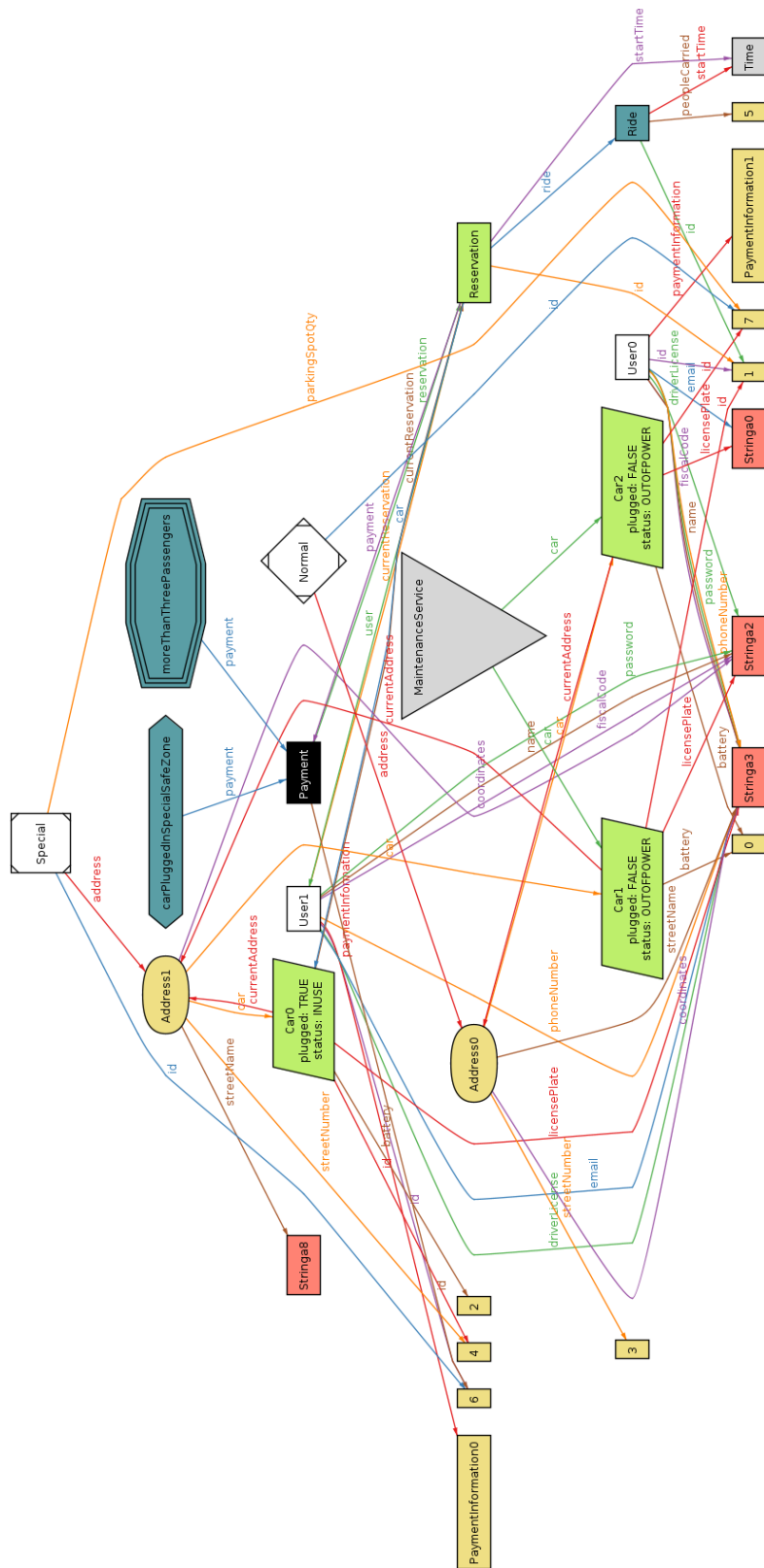
Generated World 2: world with an emphasis on Discounts and MaintenanceService

Appendices

Software used

The following software were used to produce this document:

- **Microsoft Word 2016:** as the main word processor application
- **Microsoft Visio 2016:** for UML modelling and all the diagrams
- **Alloy Analyzer 4.2:** for Alloy code writing and model verification
- **Adobe Photoshop 2015.5:** to draw the mockups

Team time management

Table describing the time management for the team.

Team member	Hours
Flavio Primo	50
Hootan Haji Manoochehri	50
	100 total