*School of Industrial and Information Engineering*

*Department of Computer Science and Engineering*


# *Software Engineering II*

# *PowerEnJoy - RASD*

***Presented by:***

***Bachar Senno***

***Gianluigi Iobizzi***

***Onell Saleeby***

***Under the supervision of:***

***Prof. Luca Mottola***

***Prof. Elisabetta Di Nitto***

***V1.0, 13-12-2016***

# Table of Contents

# Introduction

## Description of the Given Problem

Through this document we will analyze the detailed characteristics of a new information system dedicated to the management of a car-sharing service, together with the guidelines for its successful implementation. The system will be named “Power EnJoy” and will be designed with the main purpose of promoting environment-friendly mobility since all the cars operate on electric power.

The users will be able to use the available cars by reserving them at the desired time using a dedicated smartphone application. At the end of the rides the users can simply park the car in any regular safe area, and the fee will be seamlessly withdrawn from their credit card without any further actions needed. The cost of the rides will be based on the duration of the ride, and special conditions apply. These conditions will be detailed throughout the document.

With its introduction, the service will make possible a noticeable reduction of the air pollution in the future. Towards this, virtuous behavior such as sharing cars between multiple passengers will be awarded by decreasing the ride’s fee, this being one of the aforementioned special conditions.

# Glossary

- User: the user is the client of the service. He is registered in the service and has entered all the necessary data to use the service.
- Dispatcher: He is the person responsible to dispatch a pickup employee to retrieve the car in case it was left in an unsafe parking spot or left with a dead battery.
- Safe parking spot: Any spot where the car can be parked without incurring any kind of taxes or causing legal issues.
- Unsafe parking spot: any parking spot that isn’t considered safe according to the definition above.

- Pickup employee: an employee whose sole job is to pick up the car and park it somewhere safe or somewhere where it can be charged.
- Maps API: a maps platform that can be tweaked to provide the necessary functions (example: Google Maps).
- Reservation: The ability of the user to reserve a car for up to 1 hour, marking it as unavailable for other users.
- Ride/Trip: The ride/trip starts when the “Start Engine” button is pressed, and ends when the car is parked and the “Stop Engine” button is pressed, or when the emergency button is pressed.
- Emergency button: A button in the car which allows the user to inform the dispatcher that he had to leave the car in an unsafe location. The dispatcher can then dispatch a pickup employee to retrieve it if possible.

## Reference Documents

- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications.
- RASD sample from Oct. 20 lecture

## Identification of stakeholders

Basically, there are only two stakeholders:

- The government of the city that wants to improve the mobility conditions of the citizens and obtain a reduction of the air pollution.

- The private company that is in charge of the actual realization and management of the system, aiming to get a return for its investments.

## Identification of Actors

Two active participants take part in using this system:

- The actual user who is subscribed to the service, and thus is able to reserve and drive the cars.

- A dispatch officer, his sole job being to dispatch pick-up employees to retrieve cars with dead batteries and/or cars that are left in unsafe parking areas due to special circumstances.

## Constraints

### Regulatory Policy

The system requires to the user's permission to get his position, and it has to manage sensible data (position, phone number) taking into account the privacy laws, and this also includes for example never sending any kind of ads or spam emails to the user's email.

### Hardware Limitations

- User's app:
    - 3G/4G connection
    - GPS
    - Space for app package
    - Compatible mobile OS

- Car's device:
    - 3G/4G connection
    - GPS
    - In-car display

### Interface with Other Applications

- Interface with a maps API along with all needed functionalities.
- Interface with services to verify and withdraw money from the user's credit card.
- Interface with an external source to verify the user's driving license.

## Goals, Domain Assumptions, and Requirements

In the following section, the goals of the system will be clearly stated, starting with a small textual description of the overall the idea, and then extracting the exact goal from it. Next, the domain assumptions and the system requirements will be specified. The system should be able to, under the specified the domain assumptions, fulfill the goals in case the requirements are correctly met.

[R] and [D] | = [G]

### Goals

1. Registration: Users register through online form. Implement validators for fields, send verification email to the provided email address to make sure that it's owned by the person (with a verify button/link), and once pressed, process payment, and if successful, update the email as verified in the database and send access password to user.

   → **[G1]** Provide users with means to register and verify themselves.

2. Users who have the access password use an app on their mobile phone to access the functionalities of the service. The app prompts the user for their password before allowing access. The email is a unique identifier for an account, and the password can be edited by the user. The first password is computer-generated and sent to the user upon successful signup. Each email can be used by 1 device at most at a time.

   → **[G2]** Guarantee unique login per account.

3. Use a maps API to show the user a map, default starting location = his current location, with an option to change this to a different location. Show the user all available cars within 5 km from the chosen location on the map. Cars that have low battery (<20%) are shown in a special color. If a low-battery car is reserved, alert the user of the estimated trip duration he can go before the battery needs recharging again.

→ **[G3]** Provide users with an interface to view their location and available nearby cars (within 5 km), and allow them to estimate whether the power level is enough for his trip.

4. The user can click on a car in order to reserve it. Once the user clicks on a car, a dialog box appears for him to confirm reservation. Upon clicking yes, the car is reserved to this user for 1 hour, meaning that no other users can reserve or use that car. It becomes unavailable on the map for other users, and only shows for him and is distinguishable from other cars. The user can't reserve more than 1 car at a time. The user can cancel his reservation at any time, but will have a cooldown period before he can reserve again (to prevent misuse).

   → **[G4]** Allow the user to exclusively reserve one of the nearby cars for 1 hour.

5. When a reservation is made, the user is shown a screen to input his destination. This will allow the system to provide the user the route to the destination. This route is shown on the app, and sent to the car's system. The route can be shown on the in-car screen once the user starts driving. The user is the asked whether he wants to activate the money saving option for that trip. If the money saving option is enabled, show the user the nearest charging station with available plugs, which will allow him to plug in the car to the power grid after parking in order to get a 30% discount.

   → **[G5]** Provide user with route to the destination. Offer path to nearest charging station in case money saving is enabled.

6. If the 1-hour reservation period ends without the car being picked up, the reservation is cancelled, the car becomes available again for all users. A cooldown/penalty period of 1 hour is imposed on the user, and he can't reserve until this cooldown period ends. 1 Euro is withdrawn from the user's credit card. These rules are made available to the users when he attempts to make a reservation.

   → **[G6]** Prevent misuse of the app by continuous reservations.

7. Once the distance between the user and the reserved car becomes less than a certain threshold (i.e. 3 meters), an “Unlock” button appears on the phone’s screen along with a short vibration to alert the user of this, and once pressed, the server signals the car’s lock system to unlock.

   → **[G7]** Allow user access to the car once it’s reached.

8. Cars are equipped with a “start engine” button. Once the button is pressed, the driver can begin the trip. A timer starts, and is shown to the user through the in-car display, along with the accumulated fee to be paid (which is calculated based on time spend with the engine on). A route to the destination is also shown on the screen.

   → **[G8]** Start calculating the trip fee and show it to the user once the engine is started.

9. Once the user reaches the destination and presses the stop engine button in the car, the timer stops, and the amount to be withdrawn from his credit card is shown on the display. Once the user exits the car, a “Lock Car” button is shown on the screen. The doors lock once the button or pressed. If the button is not pressed, the doors lock automatically 10 seconds after the user(s) exits the car.

   → **[G9]** End the trip, withdraw the money, and lock the car, making it available again for reservation.

10. An emergency button is made available to the driver. If the emergency button is pressed, this is logged to the system, and the dispatch officer is notified of this (e.g. he can contact the user to check the circumstances). A pickup employee is dispatched to drive/move the car to a safe parking spot (or if needed, to the garage) if possible.

    → **[G10]** Prevent cars from being ditched randomly.

11. Encourage virtuous behavior by applying special discounts/surcharges in case special conditions are met. These conditions aim to promote sharing rides, and to prevent users from ditching cars with dead batteries. In case multiple discounts overlap, only the biggest one applies. In case discounts and surcharges overlap, discounts are discarded, and only surcharges apply.

    → **[G11]** Apply variations to the basic trip fee as follows:

    i. **[G11.1]** Apply 10% discount if there are 3+ passengers.

    ii. **[G11.2]** Remaining power level >= 50%, apply 20% discount.

iii. **[G11.3]** Car plugged into power grid at the end of the ride => apply 30% discount.
iv. **[G11.4]** Car is left more than 3KM from nearest power grid, or if remaining power level < 20% => apply 30% surcharge.

## Domain Assumptions

We suppose that these assumptions hold in the world:

- [D1] The information provided by the user during the registration is true (users are honest).
- [D2] The data provided (email address, credit card details) is valid (users are accurate).
- [D3] We have means of verifying validity of the credit card and continuously monitor the amount of money remaining before the ride.
- [D4] Maps APIs are available for implementing the map and location services and any needed licenses are available.
- [D5] The maps APIs can be tweaked to match our needs (add customized icons for cars, display changes in car statuses, etc …).
- [D6] The user has continuous and stable internet and GPS connection on his phone.
- [D7] There's always at least 1 car available for the user within a 5KM radius from his starting location. This can be guaranteed by predicting the increases of number of users from year to year, and making more cars available as needed.
- [D8] Locations of charging stations and number of available (unused) plugs in them are known to the server.
- [D9] Cars are equipped with screens, and are always connected to the server, even when they're not being used (available for reservation).
- [D10] The server can determine both the user's and car's locations at all times.

- [D11] Cars are equipped with a “Start Engine” and a “Stop Engine” buttons, both triggering the needed server-side actions when pressed.
- [D12] The driver can communicate with the dispatch officer in case of emergency.
- [D13] Cars are equipped with means to determine the number of passengers and accurately send this information to the server.
- [D14] The user can be a passenger in the car, or the driver.
- [D15] The user is held responsible for any damage or broken laws even if he’s not the one driving.
- [D16] The server is able to communicate with the car’s system to determine the remaining power level.
- [D17] Unit of time is 1 hour
- [D18] Number of seats in a car is 5 (simplification of alloy model).
- [D19] The user will always press the emergency button in case of emergency.

## Requirements

### Functional Requirements

Assuming that the domain properties stipulated in the previous paragraph hold, the following requirements can be derived in order to fulfill the goals listed. The list is organized as such: the goal is stated, and then the pertaining requirements are specified.

[G1] Provide users with means to register and verify themselves:

1. Generate the password dynamically, making sure it’s unique.
2. Validate the driving license.
3. Access credentials can be used once at a time (can’t be shared across users).

[G2] Guarantee unique login per account:

1. Provide interface for users to change password.
2. Generated password is varied enough to guarantee uniqueness.
3. Allow only one account per email address.
4. Allow only one login per email address.

[G3] Provide users with interface to view their location and available nearby cars (within 5 km), and show which have low power level:

1. Determine accurately using GPS and/or internet connection the user's current position.
2. Allow user to select different a starting position from the map, or stick with his current location.
3. Show nearby available cars (with reference to the specified starting location) on the map, and separate cars with low power level using some criteria (icon color etc…)
4. Update the map in real time in case cars statuses change during this timeframe.

[G4] Allow the user to exclusively reserve one of the nearby cars for 1 hour:

1. Allow user to click on a car to reserve it.
2. Warn the user that there is a 1 € penalty in case he doesn't use the car within this 1 hour period, and allow him to confirm his choice.
3. Reserve the car and make it unavailable for other users.
4. Allow the user to cancel his reservation, however, this will impose a 30-minutes cooldown period, and warn him about this condition.
5. Hide all other cars from the map and show the user only the reserved car.
6. Provide user with directions in order to reach the reserved car.

[G5] Provide user with route to the destination. Offer path to the nearest charging station in case a money saving option is enabled:

1. Show the user the path to the destination/chosen charging station on the phone, and on the in-car screen once the trip begins.
2. Only show the charging stations if they have at least one free plug.

3. The charging stations should be within reasonable distance of the final destination (max 2 KM away from the destination).

[G6] Prevent misuse of the app by continuous reservations:

1. A server-side timer starts when the reservation is made.
2. Prevent the user from reserving in case the timer reaches 1 hour without the user actually starting the engine (apply cooldown period of 1 hour). In case the user enters this cooldown period, the reservation is cancelled, apply the 1 € penalty, and make the car available again for other users.

[G7] Allow the user to access to the car once it's reached:

1. The user can unlock the car through his phone when he's at most 3 meters away from the car.
2. The phone vibrates when the unlock button becomes available to alert the user of this.
3. Login the user on the in-car display once the unlock button is pressed.
4. The user is still considered in reservation mode until the "Start Engine" button is pressed (to prevent the users from spending an indefinite amount of time in the car without actually starting the engine).

[G8] Start calculating the trip fee and show it to the user once the trip starts:

1. Start a server-side timer once the "Start Engine" button is pressed.
2. Calculate fee based on elapsed time.
3. Show the user the elapsed time as well as the fee in real-time on the in-car display.

[G9] End the trip, withdraw the money, and lock the car, making it available again for reservation.

1. When the user reaches his destination and stops the engine using the "Stop Engine" button, show the trip summary on the screen.
2. Withdraw from the user's credit card the trip's fee.
3. The user can lock the car when he gets out using the "Lock" button on his phone.
4. If the car is empty and left unlocked, it's locked automatically after 30 seconds.
5. The car is immediately made available for reservation again.

[G10] Prevent the car from being ditched randomly:

1. Provide user with an emergency button, and when pressed, the dispatch officer is informed and the car is picked up from its current location to a safe one.
2. In case of repeated behavior, the user is banned from using the service.

[G11] Apply variations to the basic trip fee as follows:

1. Detect how many passengers are occupying the car and transmit the information to the server
   → Apply 10 % discount if there are more than two passengers (included the driver) in the same ride
2. The server queries the car for the remaining power level at the end of the ride.
   → If the remaining power at the end of the ride is more than 50 %, apply 20 % discount on the basic fee
3. Apply 30% discount when the car is plugged at the end of the ride
4. If multiple discounts apply, only apply the highest one. Alert the user of this during the registration phase.
5. At the end of the ride, if the remaining power is less than 20% or the distance from the nearest charging station >= 3KM, apply 30% surcharge on the basic fee and discard any applicable discounts.

## Non-Functional Requirements

In the following section the main non-functional system's requirements will be presented, divided according to their thematic area:

PERFORMANCE REQUIREMENTS

- ✓ The system's central server shall be able to manage up to 10,000 simultaneous TCP connections.

DATA QUALITY REQUIREMENTS

- ✓ All parts of the system shall be synchronized in real-time with the central server and database.

EFFICIENCY & USABILITY REQUIREMENTS

- ✓ A maximum latency of 3 seconds is accepted for any given activity.
- ✓ Every functionality has to be reachable in no more than 5 steps from the login screen.
- ✓ The GUI of the mobile app should be intuitive and user-friendly.

ROBUSTNESS & RELIABILITY REQUIREMENTS

- ✓ The ratio $\frac{satisfied\ requests}{total\ requests}$ must be higher than 0.99999 for every component and functionality of the system.
- ✓ The maximum tolerated down time for extraordinary maintenance of the system components is 6 hours per year.

AVAILABILITY REQUIREMENTS

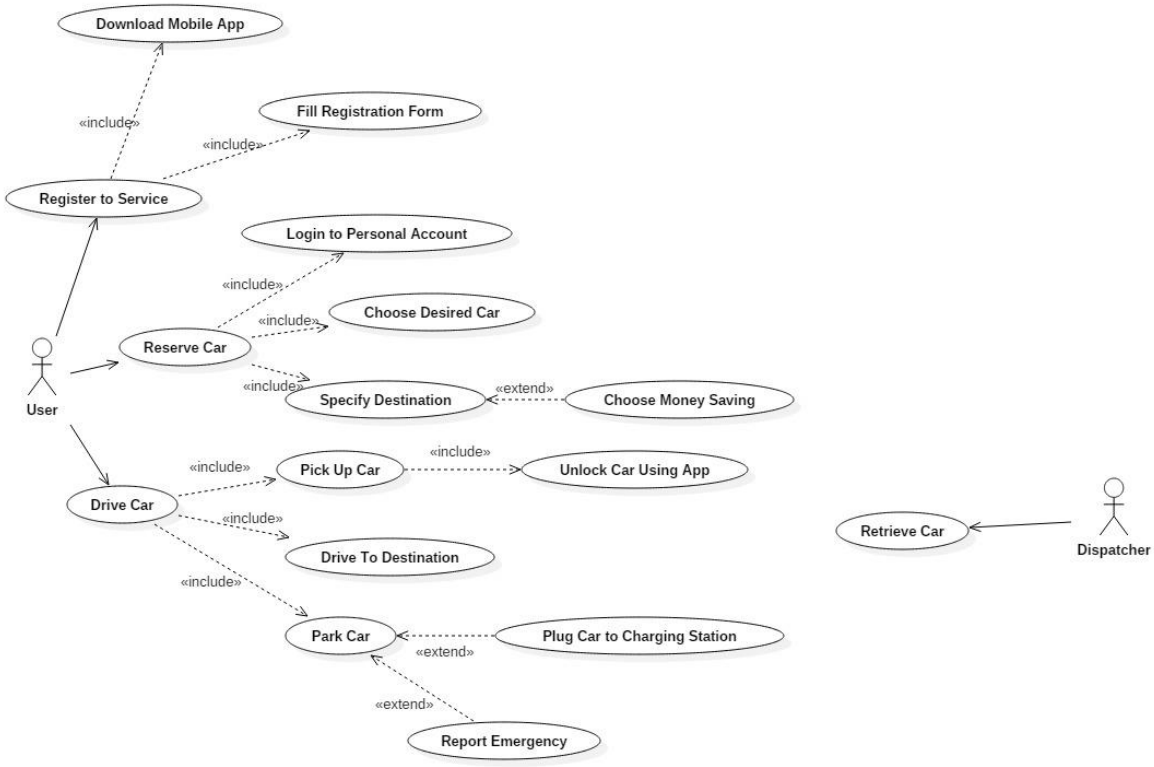
- ✓ The system shall be accessible at all time (24 hours a day, 7 days a week) apart from the aforementioned maintenance downtimes.

SECURITY REQUIREMENTS

- ✓ Personal login with email and unique password with a minimum length of 8 alphanumeric characters to be changed at least every 3 months.
- ✓ Client server connections encrypted according with HTTPS/SSL connection.
- ✓ Physical redundancy of the database and the central application servers.
- ✓ Presence of firewalls at every interface with public network.

# UML Diagrams

## Use Case Diagram




## Use Case Diagram Description

**Use Case Description**

**User Register to Service:**

Name: Register to Service.

Actor: User.

Entry Conditions: Be of legal driving age, have a valid driving license and credit card.

Flow of events:

1. Download mobile app from the appstore.
2. Choose “Create New Account” upon launching the app, and fill the registration form.
3. If the data is filled correctly, the user receives an email containing his login password which is unique and server-generated.
4. The user can use the obtained password along with his email address to login.

Exit Conditions: The user can successfully login and use the app.

Exceptions: Incorrectly submitted data will result in a suitable error message.

**User Reserve Car:**

Name: Reserve Car.

Actor: User.

Entry Conditions:

1. The user must be logged in the app.
2. The user must have GPS and internet connection on his phone.
3. The user must have at least 1 euro in his credit card.

Flow of events:

1. The system detects the user's location, and suggests available cars within a radius of 5 km. The suggested cars must have at least 50% power remaining.
2. The user clicks on a car to reserve it.
3. The user is shown a message box to confirm his choice, and a warning of the 1 euro penalty in case he fails to reach the car in 1 hour.
4. The car becomes reserved to this user, unavailable to all other users.
5. The user specifies the destination of his trip, and has the option to activate money-saving mode.
6. If money saving is activated, the system suggests the nearest available charging station to the user's chosen destination.

Exit Conditions: The user has successfully reserved a car a specified his destination.

Exceptions: If the user fails to reach the reserved car within 1 hour, the reservation is cancelled and he's charged 1 euro. The user enters a cooldown period of 1 hour (incremental, factor of 2) during which he can't reserve new cars. The user can cancel his reservation, which imposes on him a 30-minutes cooldown period, and makes the car available again.

**User Drive Car:**

Name: Drive Car.

Actor: User.

Entry Conditions:

1. The car is reserved by the user.
2. The user reached the car within the 1-hour time limit.

Flow of events:

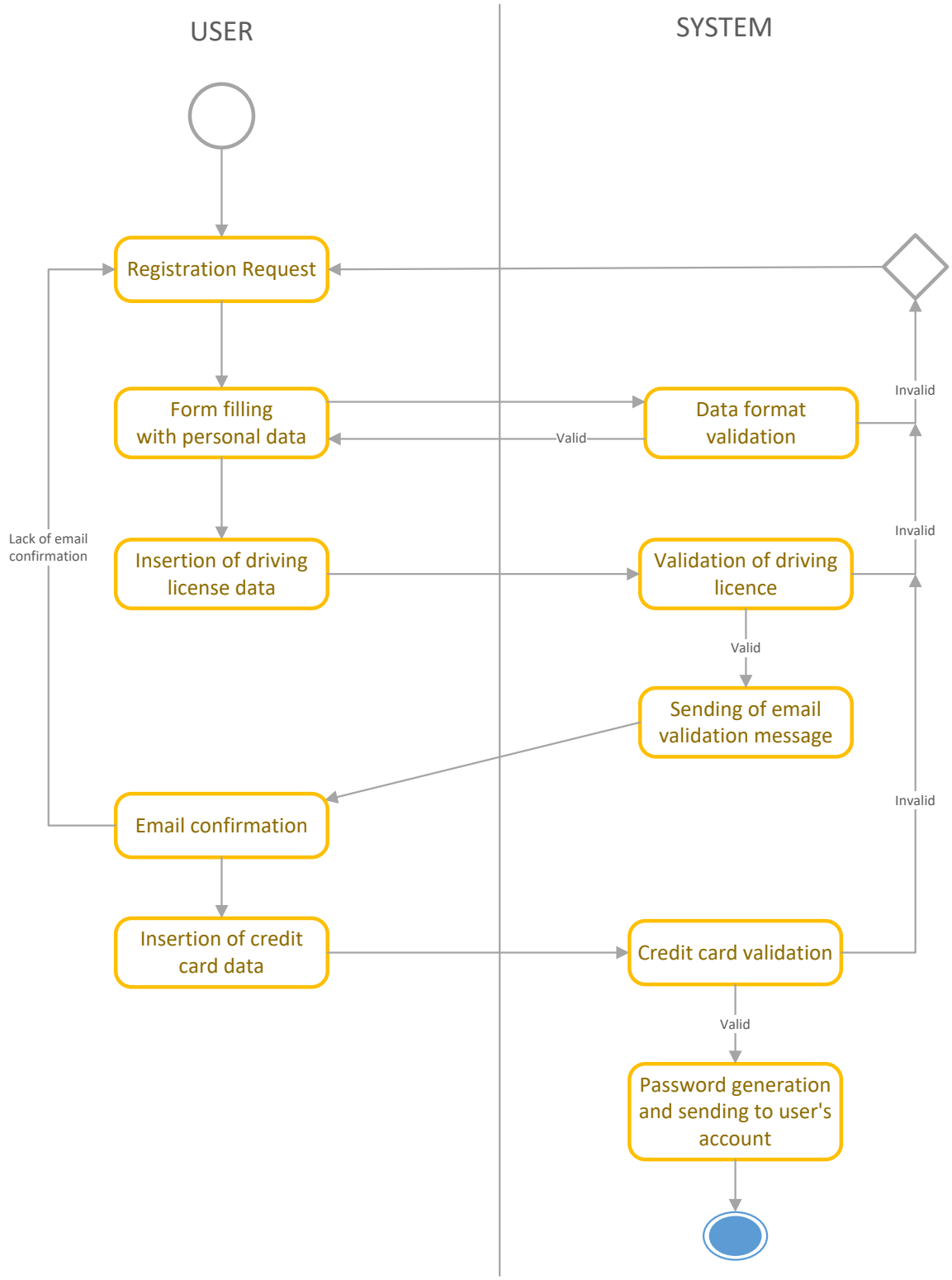
1. When the user reaches the reserved car, an “Unlock” button becomes available on the mobile app.
2. The user can unlock the car and drive to his destination. The route to the destination as well as the trip’s data (time elapsed, accumulated fee, power remaining) are shown on the in-car display.
3. When the user reaches his destination, he parks the car, gets out, and locks it through the “Lock” now available on the mobile app.
4. If the car is parked close enough to a charging station, the user can plug the car to receive a discount on the last trip.
5. The trip’s fee is withdrawn from the user’s credit card, taking into account any applicable discounts/surcharges.

Exit Conditions: The user reached his destination, exited, and locked the car. The car becomes available again for other users to use after a 10-minutes window (in case the same user wants to use it again).

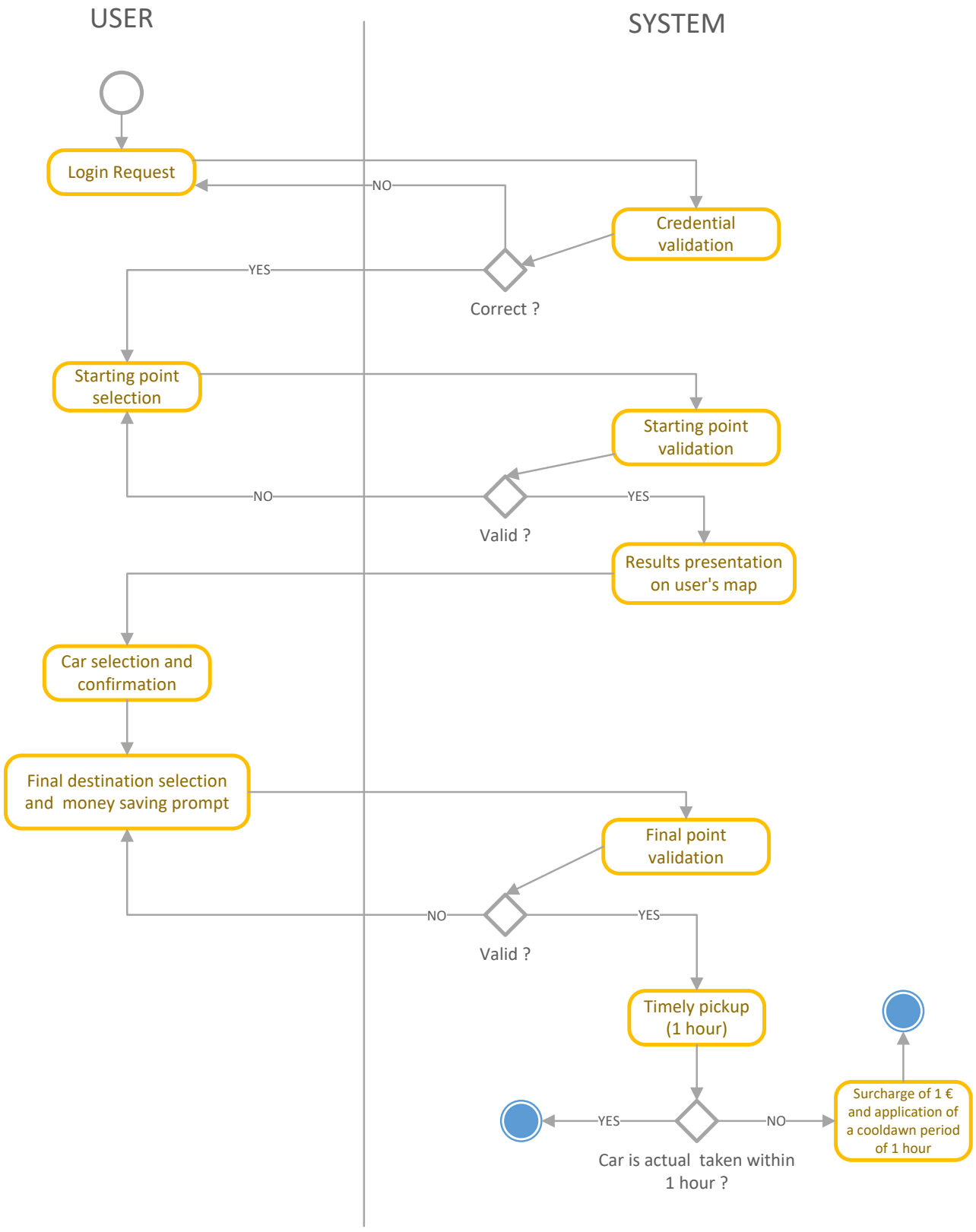
Exceptions: In case the user had to abandon the car/park it in an unsafe location, the in-car emergency button is pressed. The system is notified of this, and a dispatcher dispatches someone to pick up the car. The same happens in case the remaining power is less than 20%.

## Activity Diagrams

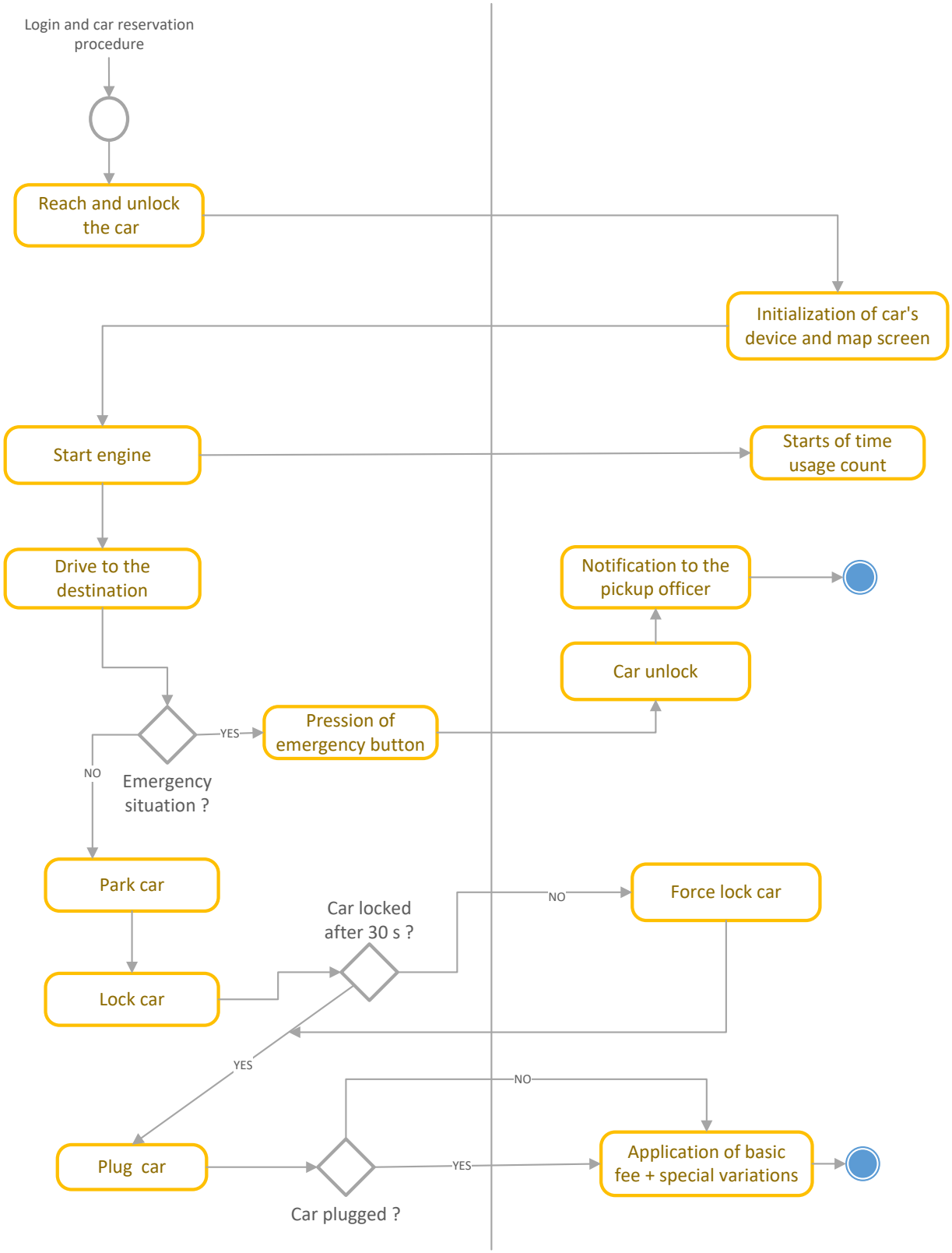
### Registration Procedure



## Car Reservation Procedure

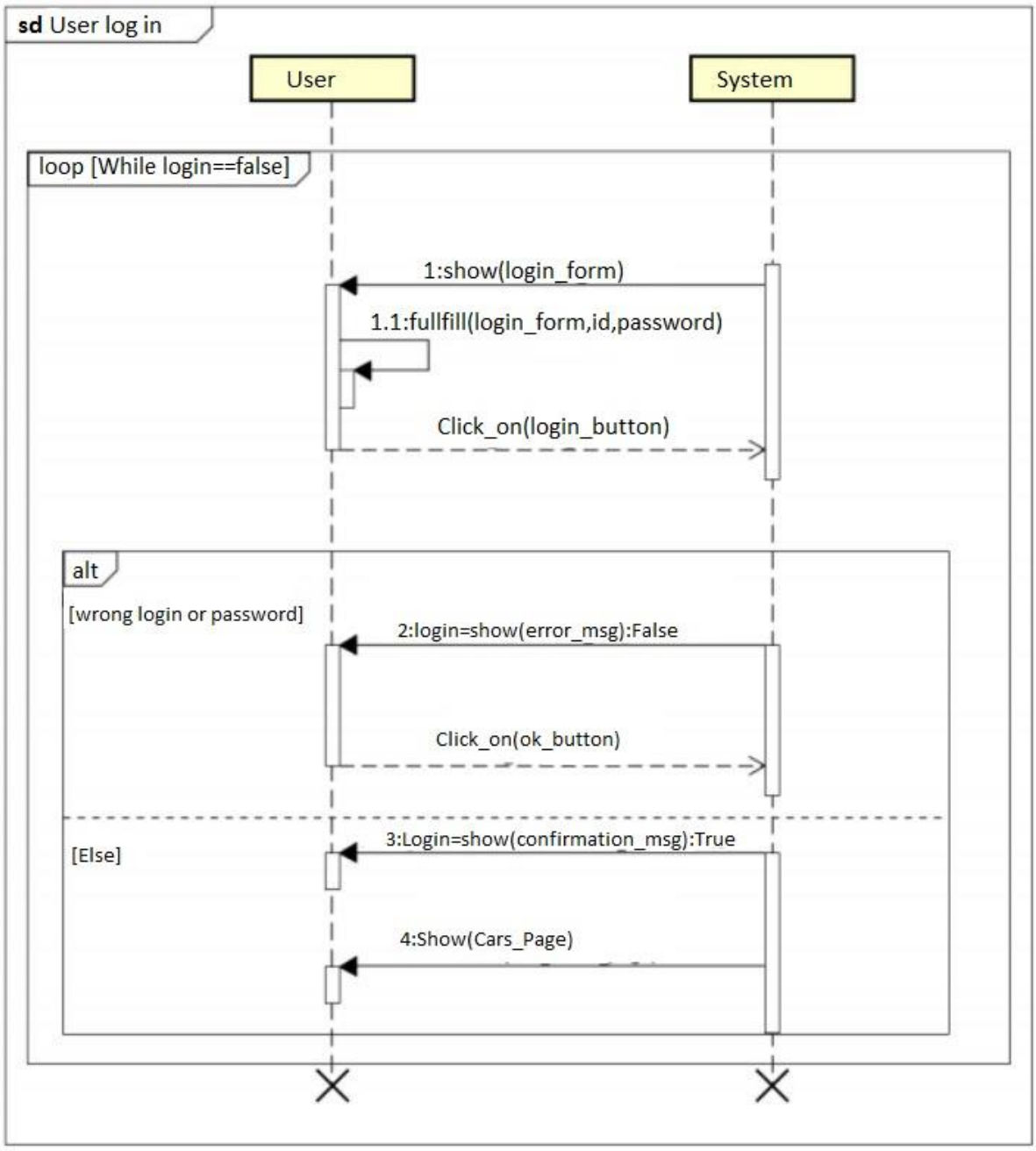


## Ride Execution

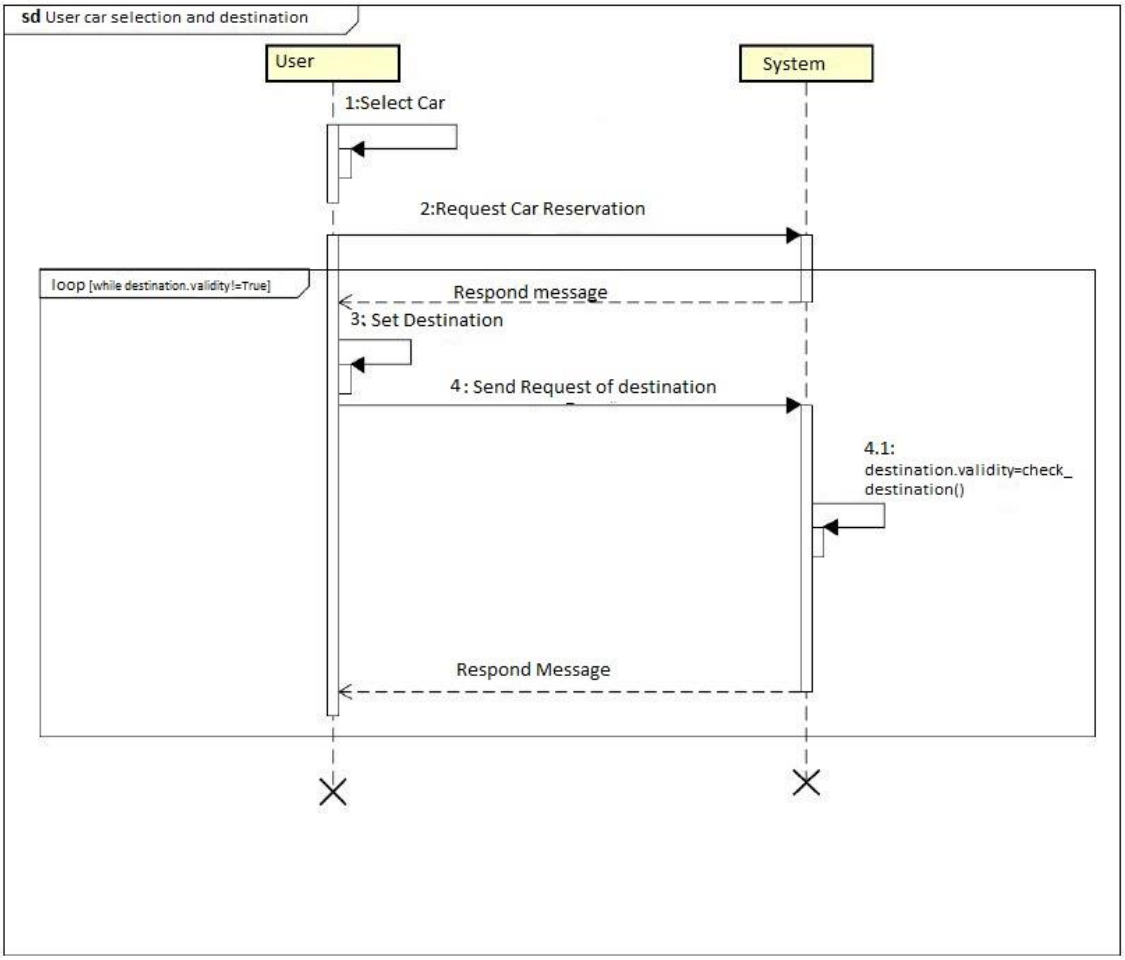


## Sequence Diagrams

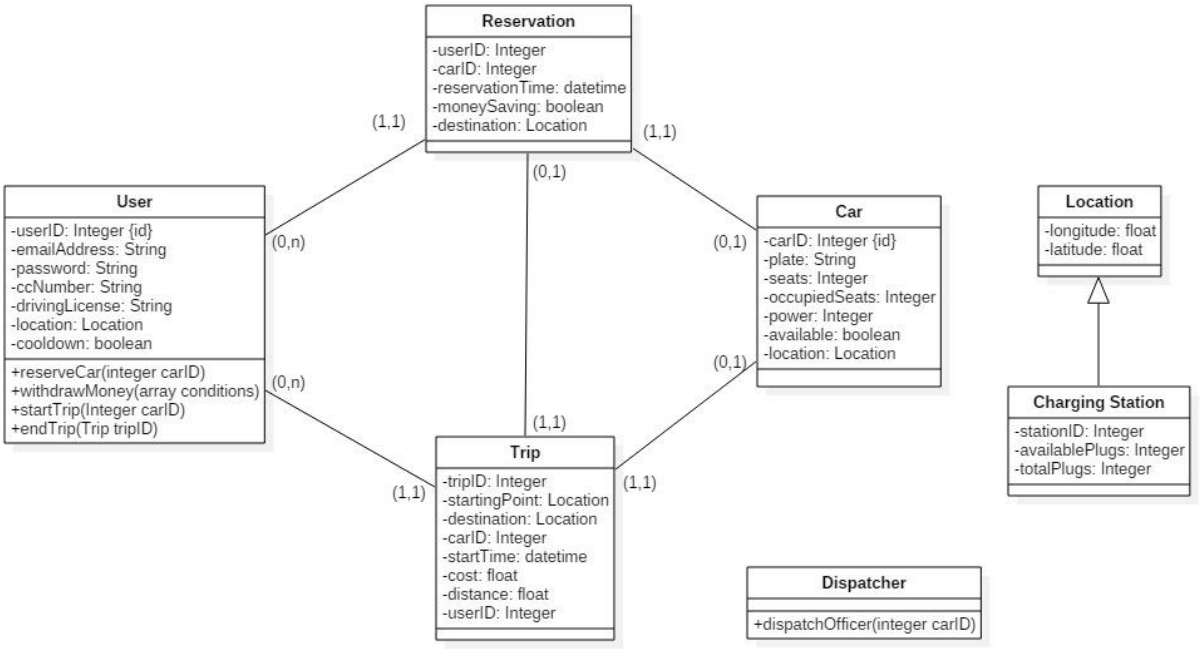
### User Login



## Car Selection and Destination



## Class Diagram



## Alloy Code

```
open util/boolean

sig time
{
   hour: Int
}
{
  hour > 0
}

sig emailAddress
{
}

sig password
{
}

sig location
{
   latitude: Int,
   longitude: Int
}

sig plug
{
}
```

```
sig car
{
    carID : Int,
    seats: Int,
    occupiedSeats: Int,
    powerLevel:  Int ,
    carLocation: location
}
{
   carID > 0
    seats = 5
    occupiedSeats =< seats
    occupiedSeats >= 0
   powerLevel >= 0
    powerLevel =< 5
}

sig chargingStation extends location
{
            stationID: Int,
            plugSet: set plug,
            chargingCars: set car
}
{
    #chargingCars <= #plugSet
    #plugSet >= 1
    stationID > 0
}
```

```
sig user
{
   userID: Int,
   userEmail: emailAddress,
   userPassword: password,
   userLocation: location,
   cooldown: Bool,
   userReservation: lone reservation,
   userTrip: lone trip
}
{
        userID > 0
}

fact noSimultaneousReservationAndTripUser
{
   no u: user | (u.userReservation != none) and (u.userTrip != none)
}

sig reservation
{
   reservedCar: lone car,
   reservationTime: lone time,
   moneySaving: lone Bool,
   destination: lone location
}
```

```
fact noTwoReservationsForSameCar
{
   all disjoint r1, r2: reservation |  (r1.reservedCar != none and  r2.reservedCar != none) => ( r1.reservedCar != r2.reservedCar)
}

fact noTwoUsersHaveSameReservation
{
all disjoint u1, u2: user | (u1.userReservation  != none and u2.userReservation != none) => ( u1.userReservation != u2.userReservation)
}
fact realReservation
{
      all r: reservation |( some u: user| u.userReservation = r)
        =>
          (
                    r.reservedCar != none
                   and
                   r.reservationTime != none
                   and
                   r.moneySaving != none
                   and
                   r.destination != none
          )
}
```

```
fact freeReservation
{
   all r: reservation | (no u: user  | u.userReservation = r )
    =>
          (
      r.reservedCar = none
      and
      r.reservationTime = none
      and
      r.moneySaving = none
      and
      r.destination = none
          )
}

sig trip
{
   tripID: lone Int,
   startingPoint: lone  location,
   destination: lone  location,
   usedCar: lone car,
   tripStartTime: lone  time,
   tripEndTime:lone  time
}
{
   (tripEndTime != none and tripStartTime != none)  implies (tripEndTime.hour >
tripStartTime.hour)
    (tripID != none) implies (tripID > 0)
}
```

```
fact noTwoUsersHaveSameTrip {
   all disjoint u1, u2: user |(u1.userTrip  != none and u2.userTrip!= none)  => (u1.userTrip != u2.userTrip)
}

fact realTrip
{
      all t: trip |( some u: user | u.userTrip = t)
         =>
          (
                  t.tripID != none
                  and
                  t.startingPoint != none
                   and
                   t.destination  != none
                  and
                  t.usedCar != none
                  and
                  t.tripStartTime != none
                  and
                  t.tripEndTime != none
          )
}

fact freeTrip
{
  all t: trip | (no u: user |  u.userTrip = t )
             =>
          (
          t.tripID = none
```

```
                and
        t.startingPoint = none
                 and
                 t.destination  = none
                and
            t.usedCar = none
          and
          t.tripStartTime = none
         and
          t.tripEndTime = none
        )
}

fact allIdsAreUnique
{
   (all u1,u2: user | (u1 != u2) => (u1.userID != u2.userID
                          and u1.userEmail != u2.userEmail
                          and u1.userPassword != u2.userPassword))
   and
  (all c1,c2: car | (c1 != c2) => c1.carID != c2.carID)
   and
   (all t1,t2: trip | (t1 != t2) => t1.tripID != t2.tripID)
}

fact noSimultaneousReservationAndTripCar
{
  all r:reservation, t: trip | (r.reservedCar !=none or t.usedCar != none) => ( r.reservedCar !=
t.usedCar )
}
```

```
fact differentStartEndPoints {
   all t:trip | (t.startingPoint != none and t.destination != none) => ( t.startingPoint != t.destination)
}

fact timeValues {
   no t1, t2: time | t1 != t2 and t1.hour = t2.hour
}

fact tripOccupiedSeats {
   all c:car | one t:trip | (t.usedCar = c  and c != none) => c.occupiedSeats > 0
}

fact noTripOccupiedSeats {
   all c:car | all t:trip |( c != t.usedCar and c != none) => c.occupiedSeats = 0
}

fact allCarsWithPowerLevel
 {
    all c:car | c.powerLevel != none
}

fact noSimultaneousCarsOnChargingAndOnTrip
{
  ( no c: car | some s: chargingStation, t : trip |( t.usedCar != none and c = t.usedCar) and (c in s.chargingCars) )
   and
  ( no t: trip | some s: chargingStation, c: car |( t.usedCar != none and c = t.usedCar) and c.carLocation = s )
}
```

```
fact noAlonePassOrEmails
{
  (  all p: password | one u: user | p = u.userPassword )
   and
    (all  e: emailAddress | one u: user | e = u.userEmail)
}

fact noAloneTimes
{
   all t: time| t = trip.tripStartTime or t= trip.tripEndTime or t = reservation.reservationTime
}

fact noPlugsSharedBetweenStations
{
  all p1: plug| no disjoint c1, c2: chargingStation | (p1 in c1.plugSet) and (p1 in c2.plugSet)
}

fact noAlonePlugs
{
 all p: plug | one c: chargingStation | p in c.plugSet
}
fact noTwoEqualLocation
{
  no disjoint l1, l2: location | (l1.latitude = l2.latitude) and (l1.longitude = l2.longitude)
}

fact noActiveUserInCooldown
{
    all u: user | (u.cooldown = True) implies (u.userReservation = none and u.userTrip = none)
}
```

```
fact chargingCarsAreAtChargingStation
{
  all c: car | one s: chargingStation | (c in s.chargingCars) =>  ( c.carLocation = s)
}

pred freeCar[c: car]  {
    all t: trip,  r: reservation | not  (c in t.usedCar) and not (c in r.reservedCar)
}

pred userCanReserve [u: user] {
   (u.userReservation = none)  and (u.cooldown = False)
}

pred userCanMakeATrip [u : user]
{
  (u.userTrip = none) and (u.cooldown = False)
}

pred freeReservation[r: reservation]
{
  no u1:user | u1.userReservation = r
}


pred freeTrip  [t: trip]
{
 no u1:user | u1.userTrip = t
}
```

```
pred reservationPreconditions [u: user, c: car, r: reservation]
{
  freeCar[c]
  userCanReserve[u]
  freeReservation [r]
}

pred tripPreconditions [u: user, c: car, t: trip]
{
  freeCar[c]
  userCanMakeATrip[u]
  freeTrip [t]
}

assert noRealReservationWithoutUser
{
     all r: reservation |
        (r.reservedCar != none)
         =>
       (one u: user | u.userReservation = r)
}

assert noRealTripWithoutUser
{
   all t:trip |
  (t.usedCar != none)
   =>
    (one u: user | u.userTrip = t)
}
```

pred show()

{

}

check noRealTripWithoutUser

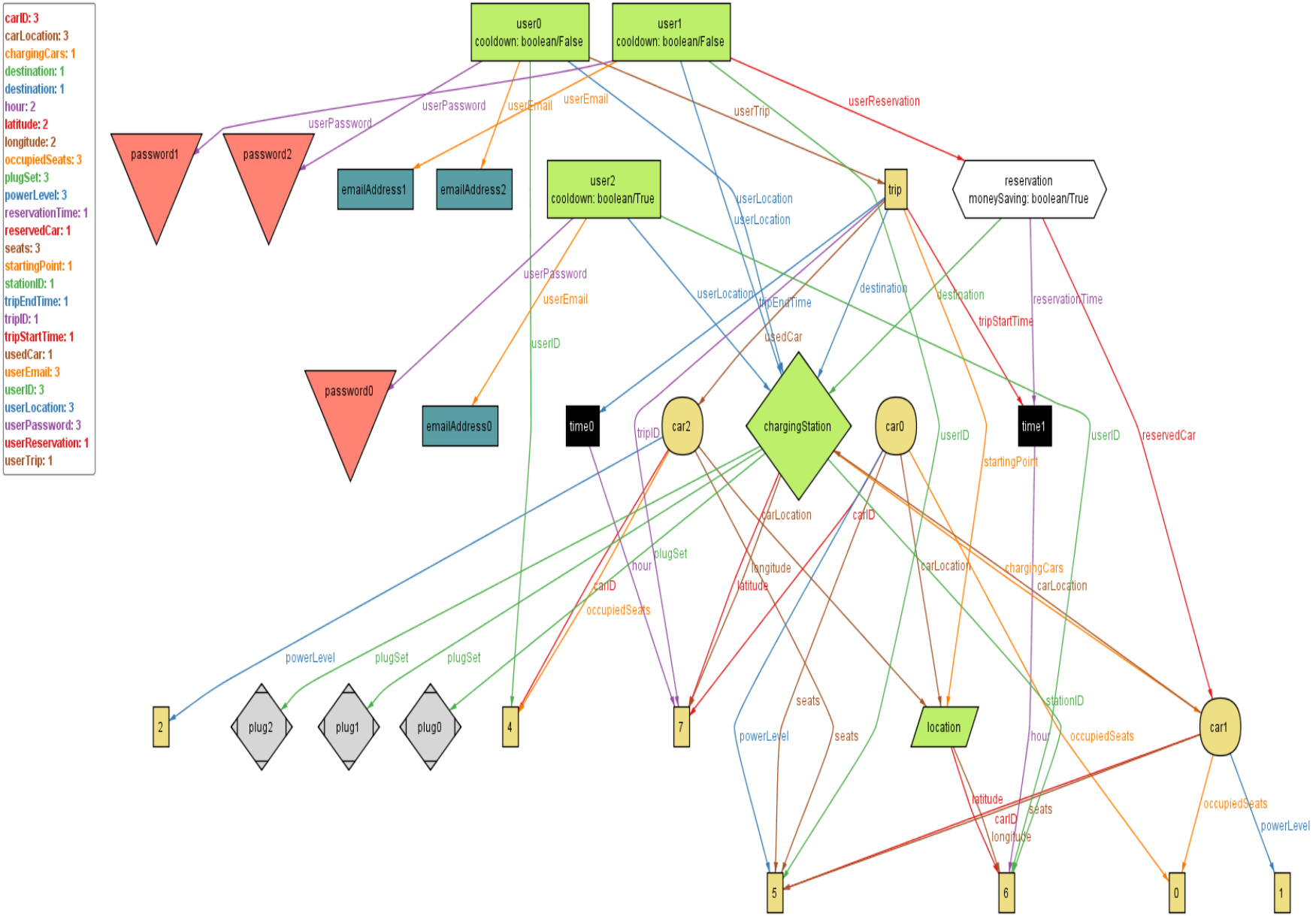
check noRealReservationWithoutUser

run reservationPreconditions

run tripPreconditions

run show for 3 but 1 reservation, 1 trip

**5 commands were executed. The results are:**
#1: No counterexample found. noRealTripWithoutUser may be valid.
#2: No counterexample found. noRealReservationWithoutUser may be valid.
#3: **Instance found.** reservationPreconditions is consistent.
#4: **Instance found.** tripPreconditions is consistent.
#5: **Instance found.** show is consistent.


carID: 3
carLocation: 3
chargingCars: 1
destination: 1
destination: 1
hour: 2
latitude: 2
longitude: 2
occupiedSeats: 3
plugSet: 3
powerLevel: 3
reservationTime: 1
reservedCar: 1
seats: 3
startingPoint: 1
stationID: 1
tripEndTime: 1
tripID: 1
tripStartTime: 1
usedCar: 1
userEmail: 3
userID: 3
userLocation: 3
userPassword: 3
userReservation: 1
userTrip: 1
user0
cooldown: boolean/False
user1
cooldown: boolean/False
user2
cooldown: boolean/True
password1
password2
password0
emailAddress1
emailAddress2
emailAddress0
trip
reservation
moneySaving: boolean/True
time0
time1
car2
car0
car1
chargingStation
location
plug2
plug1
plug0
2
4
7
5
6
0
1
userPassword
userEmail
userID
userLocation
userTrip
userReservation
tripEndTime
tripStartTime
destination
reservationTime
usedCar
startingPoint
reservedCar
tripID
carID
carLocation
chargingCars
plugSet
hour
longitude
latitude
occupiedSeats
powerLevel
seats
stationID

## Hours of Work

Bachar Senno

- 13/11 9 hours
- 12/11 5 hours
- 6/11 4 hours
- 5/11 5 hours
- 2/11 4 hours
- 1/11 8 hours
- 23/10 5 hours
- 22/10 4 hours

Gianluigi Iobizzi

- 13/11 5 hours
- 12/11 5 hours
- 10/11 3 hours
- 09/11 3 hours
- 06/11 2 hours
- 05/11 4 hours
- 03/11 2 hours
- 25/10 2 hours
- 22/10 4 hours
- 15/10 2 hours

Onell Saliby

- 10/11 1 hours
- 9/11 1 hours
- 6/11 2 hours
- 5/11 2 hours
- 2/11 1 hours
- 30/10 2 hours
- 29/10 2 hours
- 27/10 1 hours
- 25/10 1 hours

Team Meetings

- 10/11: 4 hours
- 9/11: 4 hours
- 7/11: 1 hour
- 3/11: 2 hours
- 25/10: 2 hours
- 20/10: 1 hour
- 11/10: 30 minutes

## Tools Used

- Google Docs for shared document editing
- StarUML and Microsoft Visio for UML creation
- Alloy Analyzer 4.2 to create and visualize alloy models

## Revision History

V1.0 – 12/13/2016 : Initial RASD