POLITECNICO DI MILANO

SOFTWARE ENGINEERING 2 - A.Y. 2022-23

PROF. ELISABETTA DI NITTO

eMall – e-Mobility for All

Requirements Analysis and Specification Document (RASD)

Authors
Claudio ARIONE
Riccardo BEGLIOMINI
Niccolò BINDI

Version: 0.9
December 19, 2022

Contents

1 Introduction

The unique challenges posed by climate change have recently led to a growing push in the adoption of new technologies to reduce carbon emissions. In particular, road transportation accounted for more than 70% of the total transport emissions in the EU, which in turn was responsible for about a quarter of the EU's total CO2 emissions in 2019 (source: shorturl.at/anryE).

Electric vehicles represent a viable solution to tackle this problem, but they require specific infrastructure and knowledge about availability of chargers, cost of energy and distribution.

eMall, a new startup based in Italy, is aiming at improving the experience of charging an electric vehicle. They offer a various range of services that will be able to take care of every aspect of charging, from displaying the location and properties of charging stations nearby and making smart suggestions that take into consideration both economic and logistical factors, to actually owning and managing charging points themselves.

1.1 Purpose

The system should provide the user with information about every charging point location nearby. Each location should present the characteristics of its charging stalls: cost, availability, charging speed, special offers.

The user through the system should also be able to book a charge at a specific location for a certain timeframe, selecting through the GUI a charging station and inserting a starting time and ending time (to be able to create a real schedule, as to give the other users the possibility to book a consecutive timeframe if they're willing to).

The system should also allow the user to manage the whole charging process: they should be able to start the charging, to monitor its status and to be notified when it is done or when the booked timeframe has finished and make the user pay for the service.

The system should provide some smart suggestions too to the users, based on their location, their schedule (by having access their calendar), the current offers, the battery status of the vehicle and the availability of charging stations.

From a CPO point of view instead the system should manage the charging station directly owned by eMall and manage the acquisition of energy from the different available DSOs.

1.1.1 Goals

Goal	Description
G1	Provide the user information about the nearby Charging Points (this includes their cost and the special offers they provide)
G2	Make the user able to book a specific Charging Point in a specific timeframe
G3	Make the user able to start and monitor the charging process, from start to end
G4	Notify the user when the charging process is finished or when the booked timeframe expires
G5	Make the user able to pay for the charging service
G6	Provide the user with smart suggestions about the charging port to go charge to, based on their location, schedule, prices and special offers (if available)
G7	Manage the charging stations that are directly owned by eMall

1.2 Scope

The system should interact with the end user, so the UI should be easy to use in order to be accessible to a widest range of people possible, as electric cars are spreading rapidly and such a system could further amplify this phenomenon.

eMall covers the role of a CPO (Charging Point Operator), an entity which owns and manages charging stations, beside the eMSP role, so it should communicate obviously with its CPMS (Charging Point Management System) and moreover with other CPOs, through their CPMSs as well, to broaden the service of prices and places comparison for the end user.

Each CPO, as stated previously, has its own CPMS which manages their physical infrastructure. For instance, it monitors the status of every socket

and regulates the flow of energy to each of them while a vehicle is charging. Furthermore, the CPMS is responsible for choosing the best DSO (Distribution System Operator) to retrieve energy from, based on the current price and the mix of energy sources used to produce power. With this information, the CPO can decide and set the prices per unit of energy and also create special offers for end users.

Additionally, CPMSs are tasked with managing energy storage (if present) of a certain charging station: if supplied with physical batteries, they can opt not to buy energy from DSOs and instead use the one stored in the batteries, otherwise energy can be purchased from the DSOs and partially used to recharge them. These decisions can be both made automatically or with input from a human operator.

Each of the entities previously mentioned (the system, CPOs and DSOs) can communicate through specific APIs. The system can communicate with one or more CPMSs, each owned by a different CPO, retrieving the external status of a charging station (location, number of charging sockets available, speed of every socket, cost, estimated time left until a socket is freed).

1.2.1 World Phenomena

ID	Description
WP1	The user wants to charge their vehicle
WP2	The user books a certain charging station in a certain timeframe
WP3	The user goes to the selected charging station
WP4	The CPMS of another CPO acquires energy from a DSO, based on the prices the latter offers and the mix of sources it acquires the energy from
WP5	The CPMS of another CPO distributes the energy to the different connected vehicles
WP6	The CPMS of another CPO decides the prices for a specific charging point based on the prices it acquired it from a DSO
WP7	Another CPO decides whether to store energy in the batteries of a charging point (if any)
WP8	Another CPO decides whether to use the energy stored in the batteries of a charging port (if they are available) or to use the one directly purchased from a DSO or a mix of both

1.2.2 Shared Phenomena

ID	Description
SP1	The user selects a charging port to go charge to
SP2	The user starts the charging process
SP3	The user monitors their charging process
SP4	The user gets notified about the ending of their charging process
SP5	The user pays for the charge
SP6	eMall retrieves the information about the external status of a charging station from the CPMS of the CPO which manages it (this includes the number of charging sockets available, their type, their cost, and, if all sockets of a certain type are occupied, the estimated amount of time until the first socket of that type is freed)
SP7	eMall retrieves the information about the internal status of a charging station from the CPMS of the CPO which manages it (this includes the amount of energy available in its batteries, if any, number of vehicles being charged and, for each charging vehicle, amount of power absorbed and time left to the end of the charge)
SP8	eMall starts and monitors the charging process accordingly to the request of a user through an API provided by the CPMS of the CPO managing a certain charging station
SP9	eMall retrieves the information about current energy cost from the CPMS of the CPO which manages a certain charging station
SP10	eMall provides the user with smart suggestions about the best charging stations to go charge to (based on their location, schedule and status of charging stations nearby)
SP11	eMall CPMS acquires energy from a DSO, based on the prices the latter offers and the mix of sources it acquires the energy from
SP12	eMall CPMS distributes the energy to the different connected vehicles

SP13	eMall CPMS decides the prices for a specific charging point based on the prices it acquired it from a DSO
SP14	eMall decides whether to store energy in the batteries of a charging point (if any)
SP15	eMall decides whether to use the energy stored in the batteries of a charging port (if they are available) or to use the one directly purchased from a DSO or a mix of both

1.3 Definitions, Acronyms, Abbreviations

1. **eMall – e-Mobility for All**
   The system we are analyzing and specifying in this document

2. **eMPS – e-Mobility Service Provider**
   A company (like eMall) which offers the user various features regarding the e-Mobility field, from locating a charging station, to starting the charging process and paying for it

3. **CPO – Charging Point Operator**
   A company which manages the energy supply and the features of a physical charging station. eMall is a CPO as well

4. **CPMS – Charging Point Management System**
   The backend infrastructure of a CPO, which takes decisions and interfaces with providers of energy

5. **DSO – Distribution System Operator**
   A company which produces and provides electric energy to CPOs

6. **CP – Charging Point**
   A place where charging sockets are located and where a user can charge their vehicle

7. **SPX – Shared Phenomenon X**
   A phenomenon related to the system in analysis and the domain which it operates in

8. **WPX – World Phenomenon X**
   A phenomenon happening in the domain which the analyzed system operates in. The specified World Phenomena are relevant for us because system operations derive from them

9. **GX – Goal X**
   A purpose the system wants to be able to reach

10. **RX - Requirement X**
    A condition which needs to be satisfied in order for the system to be functionally operative according to the goals

11. **UCX - Use Case X**
    An example of how (a part of) the system works when used by one of its possible users

12. **API – Application Programming Interface**
    An interface which a system offers for other systems (human beings or software) to be able to perform specific operations or retrieve information on it

1.4 Revision History

- First release: RASD v1.0 - December 15, 2022
- Second release: RASD v1.1 - December 19, 2022

1.5 Reference Documents

- Specification document: "Assignment_RDD_AY_2022-2023_v3"
- Alloy code documentation: https://alloy.readthedocs.io/en/latest/language/

1.6 Document Structure

The document is organized in six well defined sections, which are as well divided into sub sections accordingly to modularity and concept separation.

- **Section 1: Introduction**
  This section provides the context of the problem, the scope, the main goals and the phenomena (either the shared ones or the world ones), summarized in the 2 tables in section 1.2. At the end of section 1 we can also find this paragraph, the reference documents and the abbreviations and definitions necessary to understand the technical and non sections.

- **Section 2: Overall Description**
  This section provides a high level description of the problem and the system itself. Here we can find various use cases and scenarios that can happen with regard to the application, further details about the shared phenomena and domain assumptions. Furthermore, here are specified the requirements about the features the system offers and hardware and software constraints too.

- **Section 3: Specific Requirements**
  In section 3 every aspect mentioned in section 2 is specified with regard to information which can be useful to further development of the application. We can find external interface requirements, functional requirements and performance requirements, as well as design constraints and software attributes.

- **Section 4: Formal Analysis using Alloy**
  In section 4 a formal analysis on the phenomena is performed using Alloy code.

- **Section 5: Effort Spent**
  Section 5 includes a table with the details about the effort spent on each section by each member of the group tackling the project.

- **Section 6: References**
  Here we can find all the reference to documents and web pages used to draw up this document.

2 Overall Description

2.1 Product Perspective

In this following section some shared phenomena are explained and detailed with diagrams to help them be understood. Moreover, some scenarios of usage of the application are presented, with regard to user side (entry conditions and user actions) and system side (application responses and communication between its components).

2.1.1 Scenarios

1. **User wants to start using eMall to better handle the charging sessions of their vehicle**
   A user owns an electric vehicle and comes to know about eMall through their friend who has owned an electric car for a year, so they decide to sign up. They download the app on their phone and open it, seeing as a first screen the sign-up form; they insert their name, surname, email, telephone number and password and then click on the Sign-Up button.
   The system then shows an "email sent" screen and sends a confirmation link to the user email which expires after 30 minutes: if the user clicks it in this timeframe, then the account is validated with the inserted credentials and it is stored in the database (obviously the password is encrypted) and the user gets redirected to the home screen of the application.
   Here, at last, they are asked to let the application access their location and their calendar app (which could be the default one or a third-party one), to know their schedule and make smart suggestions.

2. **The user registers their first vehicle and first payment method**
   A user can link to their account as many vehicles as they want and as many payment methods as they want.
   They want to register their first vehicle so they go in their "My Profile" section and click on the "My vehicles" button: they just see a "+" button with the text "Add Vehicle" (if they had also other registered vehicles, they would have seen all of them and the same "+" button at the end of the list). They then click on it and a pop up, where they can insert a Name for the vehicle, the capacity of the battery and the model, opens. The user then inserts these data (all required) and at

the end of the insertion they click on the "Confirm" button, thus saving the insertion of the vehicle; If they changed their mind they could have clicked on the "Cancel" button to abort the operation. The application then shows a "Vehicle saved" pop up (for cancel button it would have showed "Operation cancelled") and then redirects the user to the "My vehicles" section.
As soon as the user saves their vehicle it is automatically marked as default one, as it is the first, while if they already had some, it would have been put at the end of the list and could have been marked as default by the user (if they wished to) by clicking the "Set as Default" button on the entry of the vehicle itself.
The process to save the first payment method is basically equal as the one needed to save the first vehicle: the user goes to the "Wallet" section and clicks on the "+" button; as for the vehicles, they only see the + and clicking on it a pop up opens, where the user is able to insert the payment type (Credit Card or PayPal) and the needed data for each one of them (for example card number, expiry date and CVV for Credit Card or the login credentials for PayPal etc.).
The user clicks on PayPal, gets redirected to the PayPal login page and at the end of the login process they get redirected to the previous screen, where they click on the "Confirm" button to end the registering process (they could have also click on the "Cancel" button to abort the operation).
As for the vehicles the application then shows a "Payment method saved" pop up (it would have showed "Operation cancelled" if they had pressed the cancel button) and then redirects the user to the "Wallet" section.
Even in this case the inserted payment method is automatically marked as the default, as it is the first, while if they already had some, it would have been put at the end of the list and could have been marked as default by the user (if they wished to) by clicking the "Set as Default" button on the entry of the method itself.

3. **The user wants to retrieve information about their nearby charging locations**
   From the home screen the user can directly see all the charging points near their location. On the first access the user gets asked by the application to let it retrieve the information about their location through the GPS service of their phone: without consent to this, the user will not be able to use this feature.

Once granted the access, the application centers the map in the user location and shows with different colors all the different charging points managed by different CPOs.
It is important to clarify the operations eMall does in the backend: it indeed contacts the CPMSs of the other CPOs through dedicated APIs and retrieves the statuses of the charging stations and the prices they have, updating its database accordingly, which is then populated with charging points owned by eMall itself and other CPOs too.
The user then clicks on some of these marked locations and for each one of them (one at a time) are shown the distance from the current location, the price per kWh, the free charging sockets (if a charging point has no free charging sockets gets marked in grey) of each type and if all of a certain type are occupied also the estimated amount of time until the first socket of that type is freed. Moreover, if there are special offers related to that charging station, those are showed too (with their prices)

4. **The user charges their vehicle**
   When the user arrives at a charging station, from the app they select the correct marked location and a pop up opens with all the information about it and also the "Start charging" button, which only appears if the user is in a 100m range from the station. The user then clicks on it and another screen appears, where the user inserts the code of the charging socket (we suppose every single socket has a unique ID): the app then asks for a further confirmation and shows the default vehicle and payment method as selected.
   The user confirms both and sets the time to charge their vehicle until, then they click on the "START!" button which unlocks the socket and lets them physically plug the car; at that point the vehicle is charging and from the app the user can monitor the process, seeing the energy injected in the vehicle, the current amount to be paid in real time and the estimated battery level which the vehicle should be a the end of the process (in alternative they could have selected a target percentage of charge and the app would have shown the estimated amount of time until it was reached). The unlock operation of the charging socket is performed directly by eMall CPMS only if the charging station is owned by eMall itself, else it is performed by the CPMS of the CPO owning it, which communicates with eMall one through specific APIs and got notified of the operation to be handled by eMall.
   The user after 30 minutes gets bored and decides to leave even if

he had previously set the charge timer to 40 minutes: he opens the recharge popup in the homepage of eMall application, clicks on the "STOP" button and unplugs vehicle from the socket; the application takes the money the user owes from the selected payment method and shows a "Charge ended" pop up, while the outcome of the transaction is showed in the transaction history section of the user account. If the charging station is managed by eMall itself, then the money is due to it directly, while if it is managed by another CPO, it is notified about the outcome of the transaction by eMall and it will get the money through it, deducted the commission of the service.

5. **The user wants to book a charging point for a certain timeframe**
   The user decides to schedule a charging session for the next day to better organize their day. So, they open app and tap on the "Schedule" section: a calendar opens up, where they select the desired day and the desired timeframe (they select the 9:00 – 11:00 frame as they have a charging station pretty close to the office they work at) and the desired charging station and socket type (Fast, Rapid or Slow), based on availability. The app then asks for a future confirmation ad then shows, default vehicle and payment method as selected, the estimated amount of money for the charge and also a red text which says:

   > "Be aware that half of the estimated amount of money due for the charge will be pre-authorized from the selected payment method. You can delete your reservation from the "My reservation" section in your account.
   > If you fail to delete it before 2 hours from the starting time the half of the pre-authorized amount will be taken".

   The user then confirms the vehicle and the payment method and at the end clicks on the "Reserve your spot" button; the application shows a "Socket booked" pop up and redirects the user to the "My reservations" section (if the user had wanted to abort the operation they could have clicked on the "Cancel" button and the application would have showed an "Operation cancelled" pop up without redirecting the user).

6. **The user receives a suggestion**
   The user receives a notification from the eMall application:

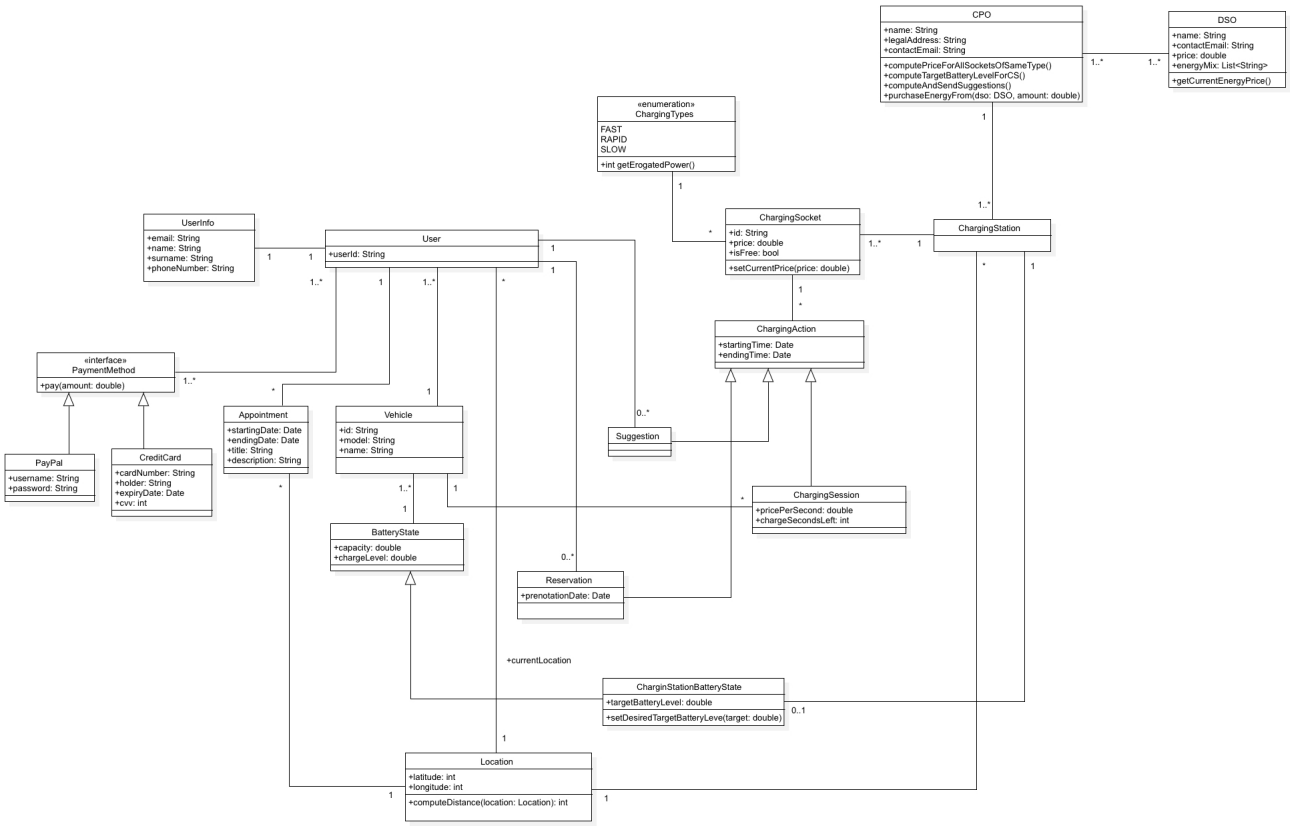
"We have a suggestion from you!"

The user then clicks on it and the application on their phone opens directly in the "My Suggestions" section. Here they can see a list of all past suggestions and the new one at the top of the list marked with a "NEW" green text at the right of the entry.
The user clicks on this new suggestion and a new screen opens, where some information appear: the application has indeed found that the user has a charging station near the restaurant he has planned to go to that evening between 20:00 and 23:00 and so it shows that same charging station, its prices and the fact that there are free rapid charging sockets for the 20 – 23 timeframe (it shows the rapid sockets because they are the cheapest ones due to a special weekend offer).
The user has the possibility, from this page itself, to click on a "Accept suggestion" that will book that same charging station, in rapid charge mode, in that same timeframe, so they click it and the application shows a "Reservation confirmed" pop up. If they would have not liked it and not clicked it then the user could have clicked on the "X" at the top right of the screen to go back to the "My suggestions" section, where this new suggestion would not have been marked as new anymore.

7. **The system CPMS decides which DSO to acquire energy from and sets the prices for the users**
   eMall CPMS contacts through the provided APIs a list of DSOs to acquire energy: each one of them offers a different price and some of them offer some monthly based solutions to be more competitive, still making money out of it obviously.
   After receiving all the answers from the DSOs and after an attentive evaluation, eMall decides to acquire the energy for the next three months from E-Distribuzione SpA, the biggest one in Italy for 0.21€/kWh.
   Knowing this, the eMall CPMS is able to compute the prices for the end users, so it decides to go for 0.35€/kWh for Slow charging sockets, 0.42€/kWh for Rapid charging sockets and 0.55€/kWh for Fast charging sockets.
   The CMPS then sets these prices in their database and updates the user application with the current values in real time.

8. **The System decides to charge a connected vehicle at the "Piazza del Duomo" charging station through the energy stored in the batteries of the station itself**
   The CMPS of eMall has seen that many users in Milan tend to charge their vehicles at the Piazza del Duomo charging station, since they are able to reach the station by car and then have a walk in the city center while it charges. To better handle the workload and the amount of energy to be dispensed, the station has been previously provided with batteries (providentially) that are charged during the night from the energy acquired from a DSO at a discounted price. Once a new car arrives, the system then checks the level of the battery of the charging station and since they are more than 80% full, the CMPS decides not to charge it through the energy coming in real time from the DSO, but through the batteries. The CPMS then through and API switches the incoming source of energy from DSO to batteries and the vehicle charges as nothing happens; this decision is completely transparent to the end user themselves, they indeed do not need to do anything different on their interface, nor about what they physically need to do at the charging station.

2.1.2 Class Diagrams

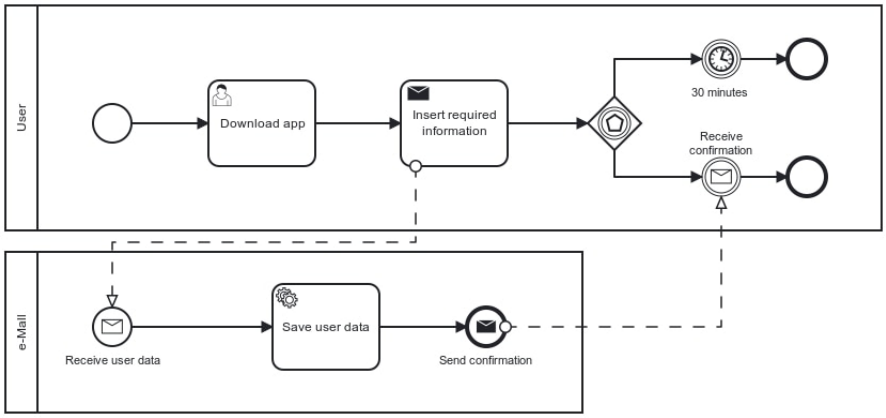


The UML diagram gives an overview of all the components needed to describe the system. In the following lines a brief description of the main entities is provided.
A *User*, in addition to all the contact information – that have to be verified - needed to minimize the probability that automated or unofficial clients can access the functionalities of eMall, has a *currentLocation* attribute, updated in real time in order to show the user the charging stations closer to him, one or more saved *PaymentMethods*, one or more electrical Vehicles, a list of active *Reservations* and *Suggestions* – based on any discount on energy price offered by some CPO or on the *Appointments* of the user.
A CPO communicates with many DSOs through its CPMS and, according to the data retrieved from them, many decisions can be taken. For example, observing the fluctuation of the energy price and other internal and external factors, it can update the energy price of the *ChargingStations* it owns,

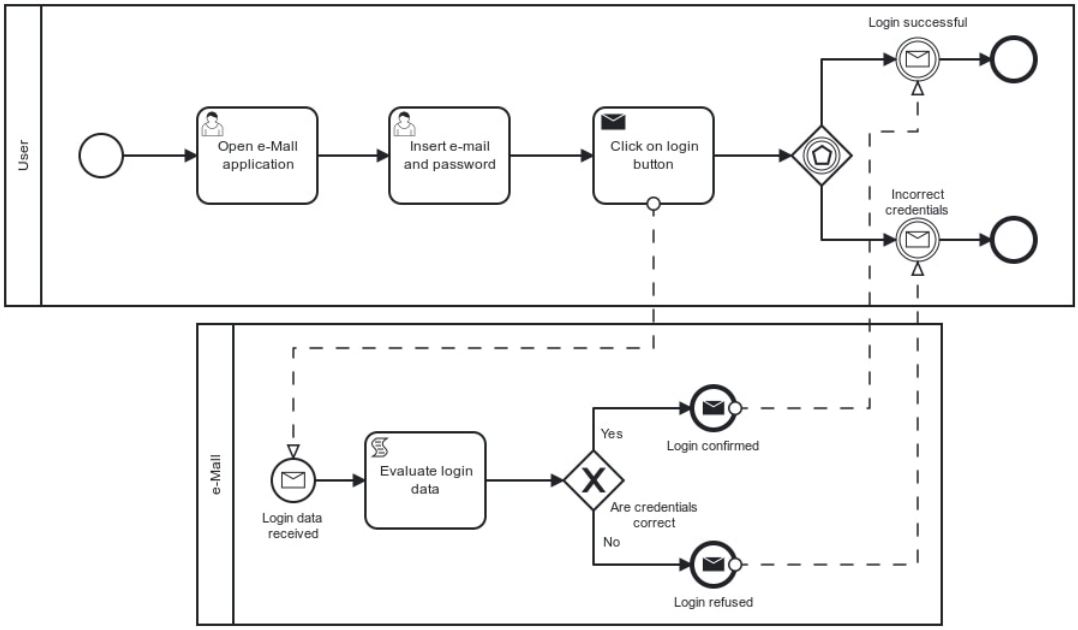
offering discounts if some conditions hold, or decide whether to store energy in the batteries of the most popular stations.
It's important to clarify that all the decisions associated to the CPMS entity can be taken automatically by the system (which schedules periodical tasks to check DSO prices or batteries levels and performs actions based on their results) or manually by a human operator, who interacts with the backend through a desktop application installed only on eMall's computers. Note that all the operations performed by humans have higher priority than the corresponding ones done by the automated software; in addition, an agent can validate, cancel or delay any order carried out by the system. The order is automatically validated in absence of an explicit decision by an operator within 15 minutes: the reason why the order is automatically processed – rather than rolled back – is that if no one takes an explicit decision, then the operation is based on the trust on the software, which is able to individuate favorable buy-pattern.
A *ChargingStation* is composed of many *ChargingSockets*, each one possibly associated to multiple *Suggestions* and *Reservations*, but at most to an active *ChargingSession*, that is paid with the provided *PaymentMethod*.

2.1.3 State Diagrams

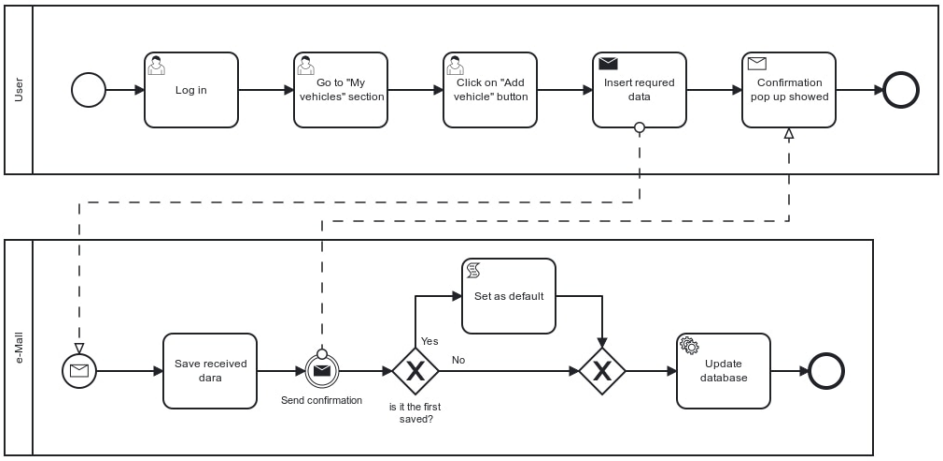
- Registration



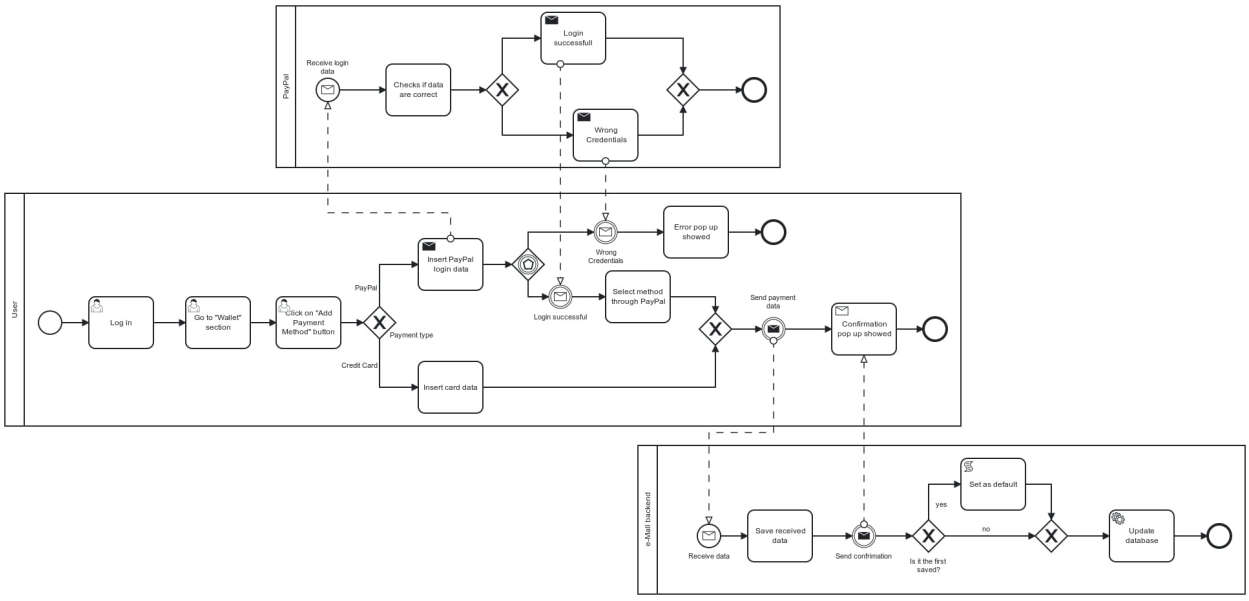
- Login



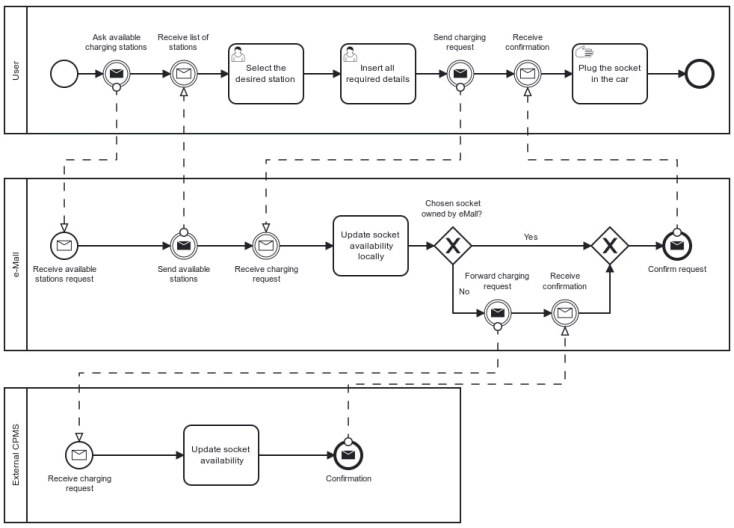
- Save Vehicle



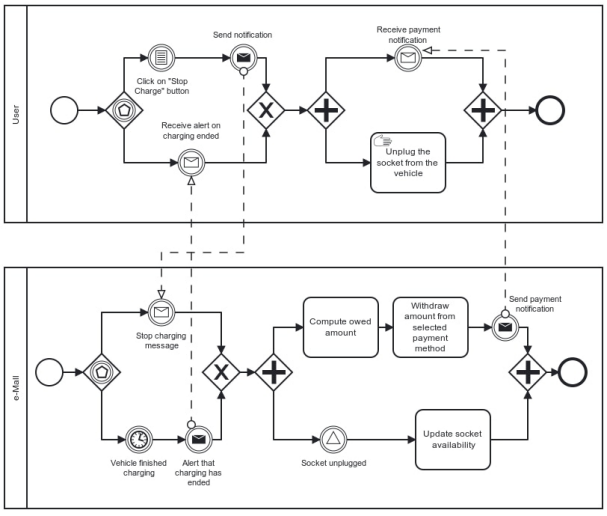
- Save Payment Method



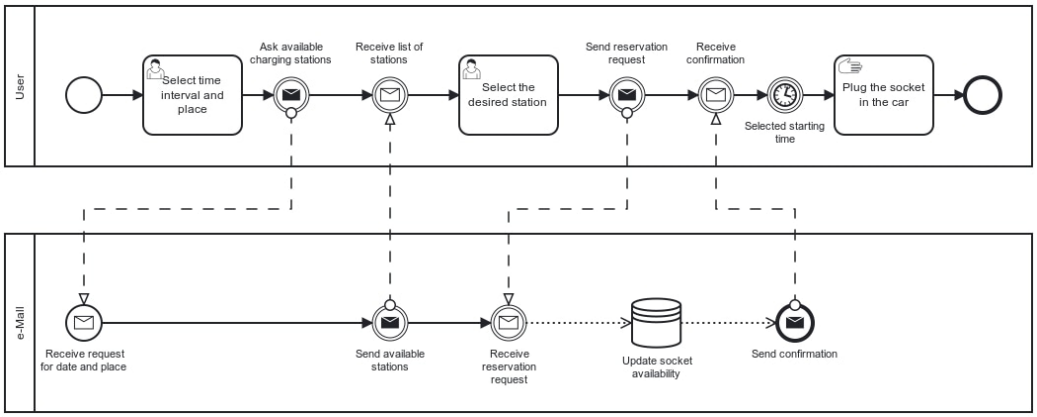
- Charge the Vehicle



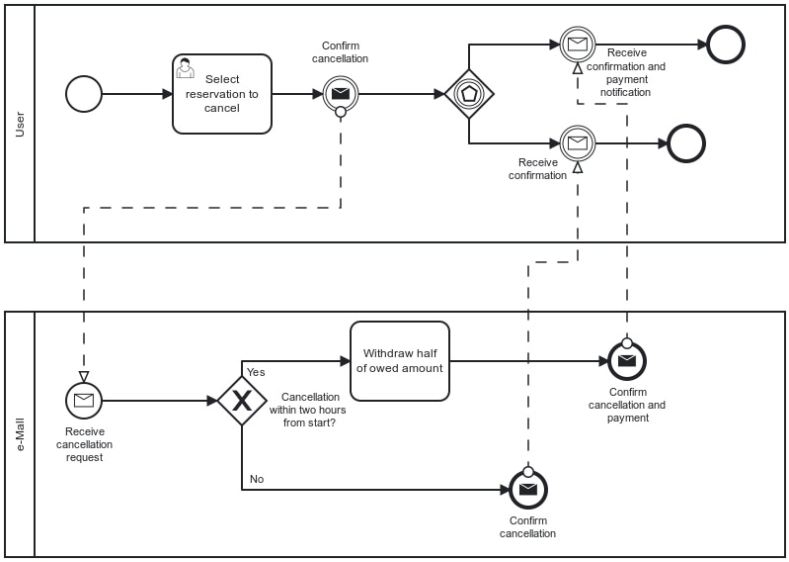
- End a Charging Session



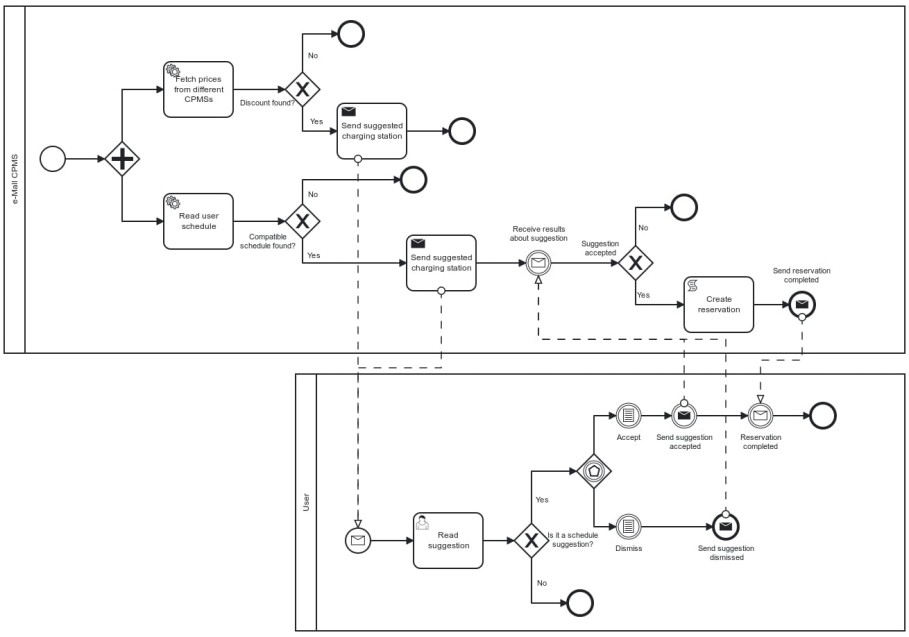
- Reservation



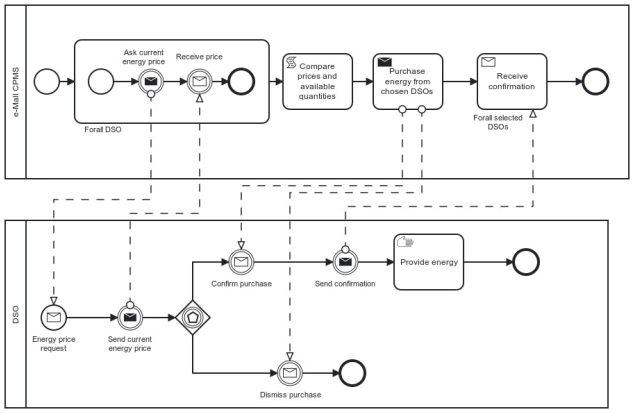
- Cancel Reservation



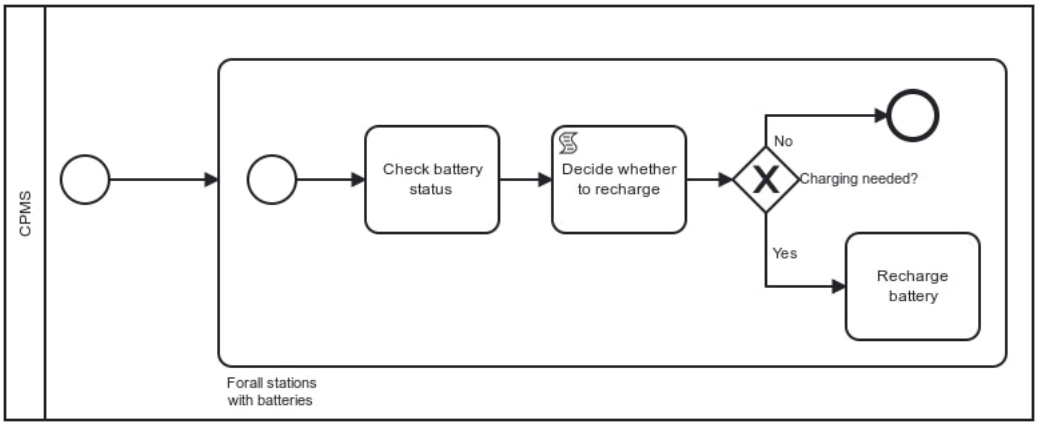
- User Receives Suggestion



- Purchase Energy from DSO



- Charge Batteries from DSO



2.2 Product Fuctions

In this section the main functionalities of the eMall system are explained in a more orderly manner.

2.2.1 Control of the charging process

The system will allow end-users to manage the whole charging process of their electric vehicle. It will be able to show every available location in certain area, along with specific details about each of those.
If the user is near the selected station, it will allow them to immediately begin the charging process.
It will also let them reserve a charging spot at a given time on a given day.
While the vehicle is charging, the system shows the status and the remaining time to completion (based on the battery capacity provided by the user).
When the charging process is finished, the system proceeds to make the user pay.
To make it faster for the end-user to compete the former steps, the system will have the possibility to remember payment methods, details about the vehicle and other personal information the user will provide.
Additionally, the system can store details about the user's vehicle, like the model and battery to better compute estimated costs and remaining time for charging sessions.

2.2.2 Energy management in every charging station

The system will allow to make decisions regarding many aspects of energy management in a certain charging station. This will consist mainly of two aspects:

- Being able to select the current best DSO (based on the price of energy offered, the mix of sources they produce energy from...) and set the prices for end-users accordingly.
- Being able to strategically use the batteries that may be present at a charging station, based on what is most convenient at a given time.

All the functions described above can be either automatically fulfilled by eMall's CPMS or manually by a human operator, which can directly interact with the system itself.

2.2.3 Proactive prediction of end-user needs

The system will be able to identify which course of action best fits the user needs, based on the information it has (if the user grants access to them):

- current position
- calendar schedule
- battery level of the vehicle

The system will then communicate with the end-user with personalized notifications, who can then either accept the suggestion or dismiss it.

2.3 User Characteristics

There are three types of users considered in the eMall system.

1. **Unregistered end-user**
   A user that has not registered yet in the eMall system. In order to use its functionalities, they first have to sign-up.
2. **End-user**
   A user that has successfully completed the registration process and can have full access to the system's functionalities, with which they interact using the eMall app.

3. **eMall admin**
   A person within the company eMall that is responsible to directly modify data (as described above) in eMall's CPMS. They interact with the system using a specific desktop application designed for internal use only.

2.4 Assumptions, Dependencies and Constraints

2.4.1 Domain assumptions

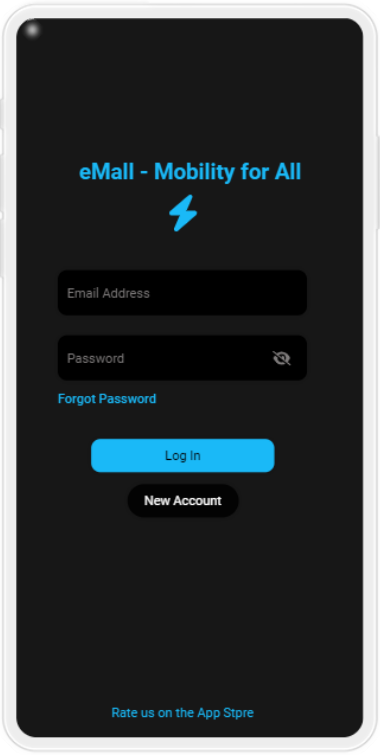
ID	Description
D1	The end-user has an active Internet connection
D2	The end-user does not input fraudulent information (e.g., fake reservations to disturb other users)
D3	The vehicle's details about the battery capacity are true and exact
D4	Physical entities at eMall's charging stations (e.g., sockets, batteries) work properly
D5	Each external CPO provides an API compatible with the system
D6	Data provided by other CPOs is valid and trustworthy
D7	Each DSO provides an API compatible with the system
D8	The information upon which the system provides suggestions (location, calendar...) is valid and reflects the user's real status

3 Specific Requirements

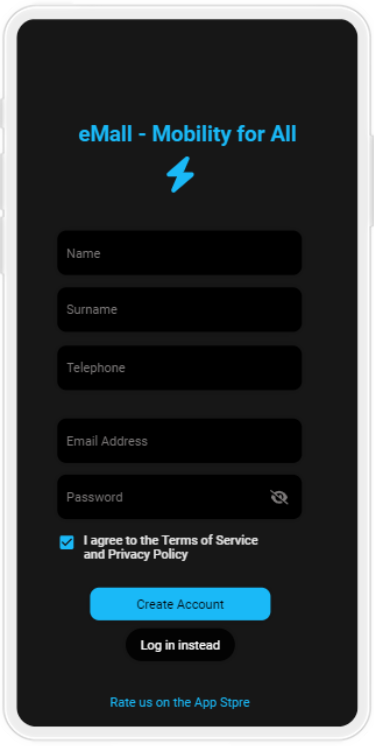
3.1 External Interface Requirements

3.1.1 User interfaces

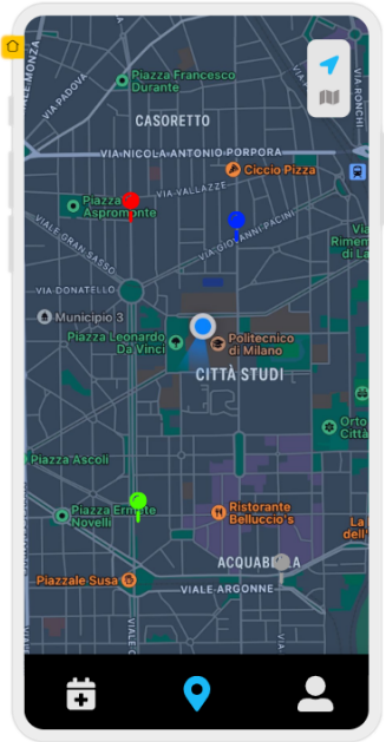
eMall mobile application, as stated previously, should ideally be used by a vast percentage of people, with heterogeneous age and technical ability, so it should have an easy-to-use and intuitive graphical interface, offering clear and simple functions that.
eMall was not thought to be available as a web application due to the fact that the end user should be using it mainly in a charging station and it is improbable that they have a PC with them, while it is far more probable that they have their phone, nonetheless it could be extended to be accessed from a web browser for those who prefer not to download the app on their device.
Following this paragraph, you can find the most significative screens from the mobile application dedicated to the end user.
The interface of the desktop application installed only on eMall devices does not have particular requirements and it is also thought to be as clear and as "empty" as possible, to offer the administrators their needed features (the possibility to check the CPMS automatic work and to communicate directly with charging stations and DSOs), without distractions and time losses deriving from rendering the graphics; actually all the features could be implemented through a CLI but we opted for a desktop application for security reasons: it has indeed in it a check for the physical machine which it is installed on, to prevent its use in computers different from eMall ones.



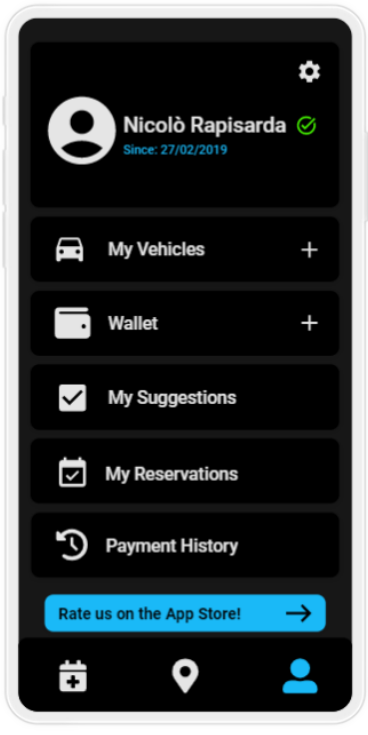
Login



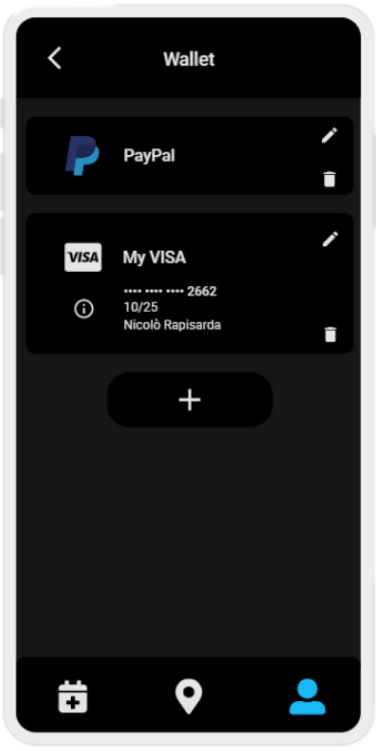
Sign-Up



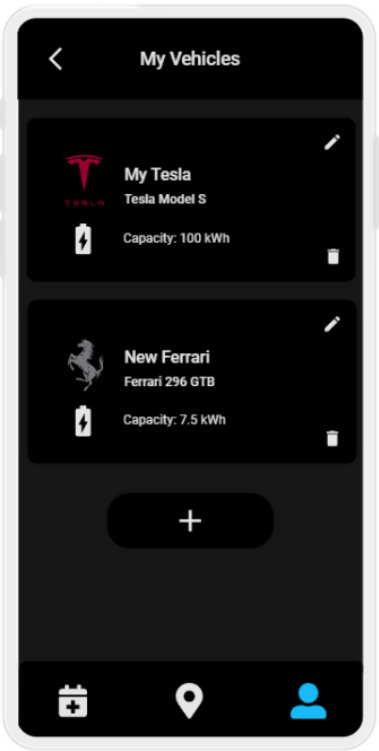
Home - Map



My account

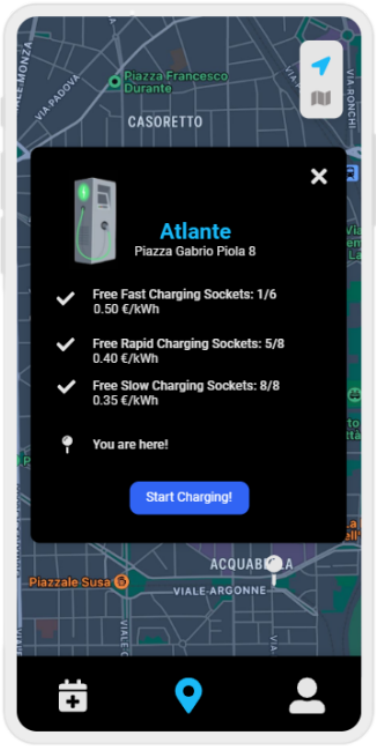


Wallet

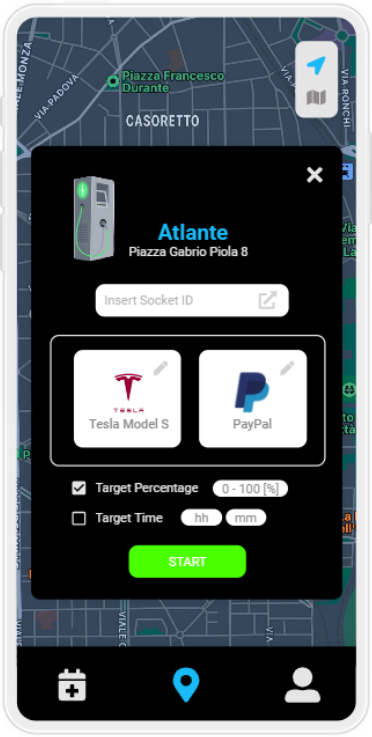


My vehicles

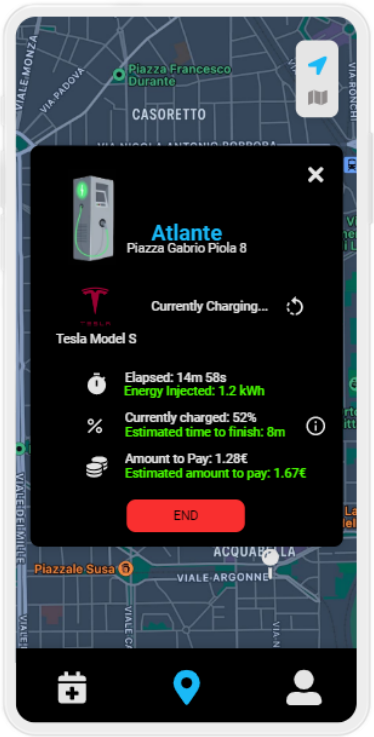
Charging Station (away)



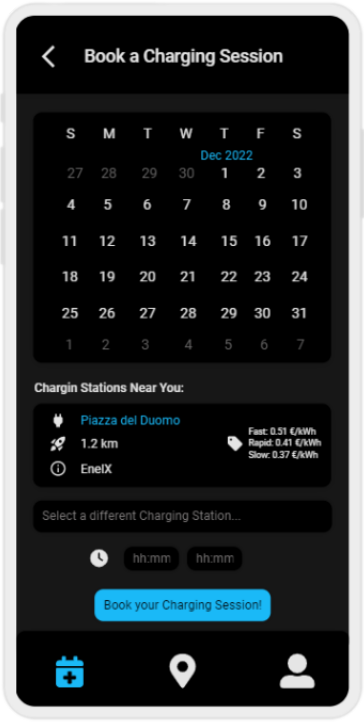
Charging Station (nearby)



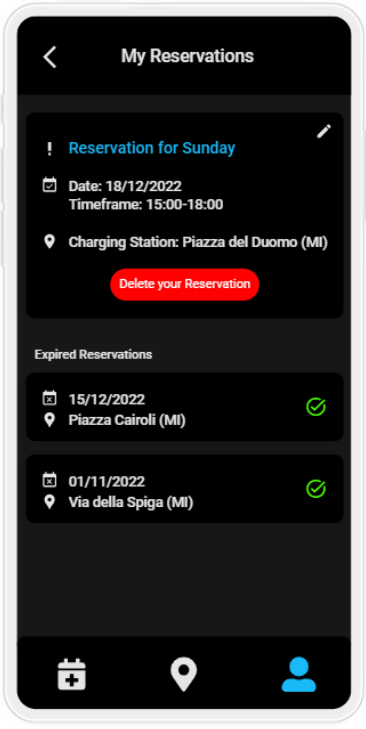
Start Charging



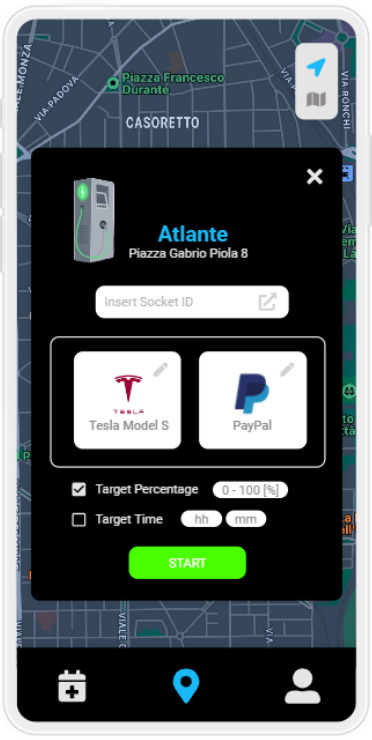
Charging Info



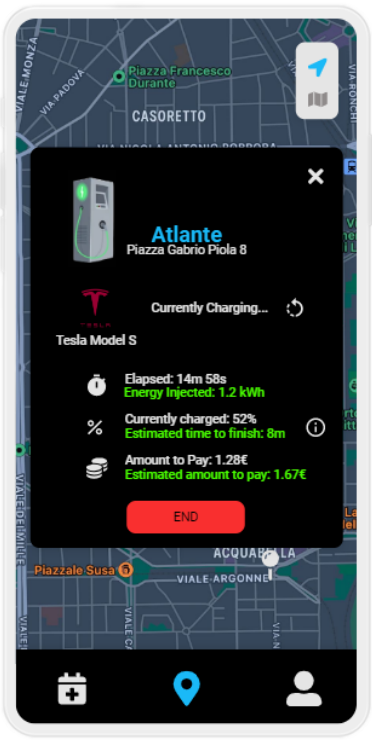
Book a Charging Session



My Reservations



Start Charging



Charging Info

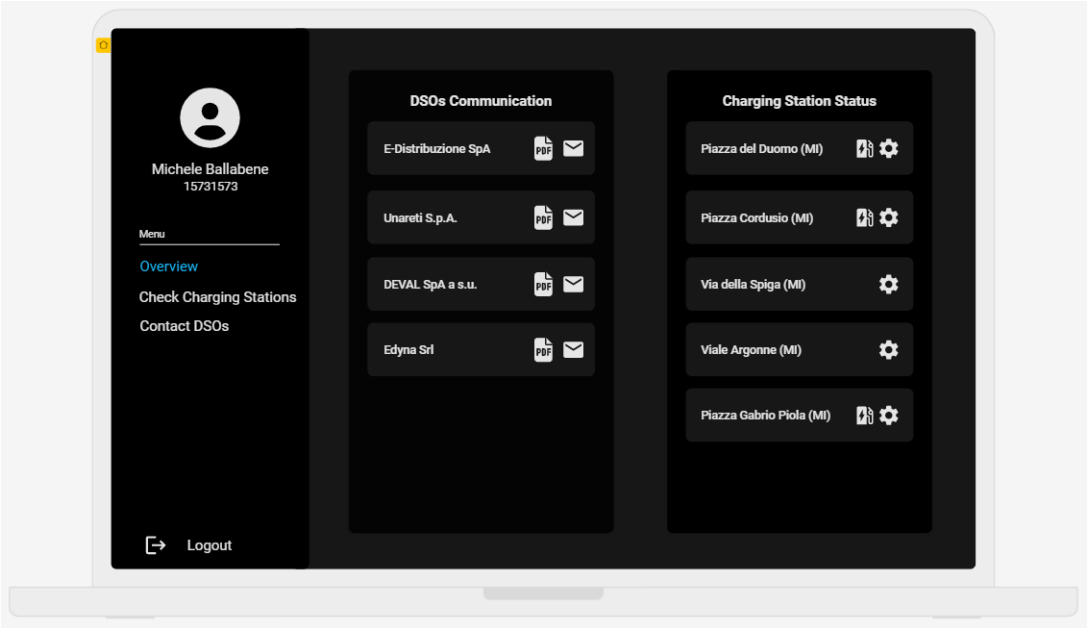
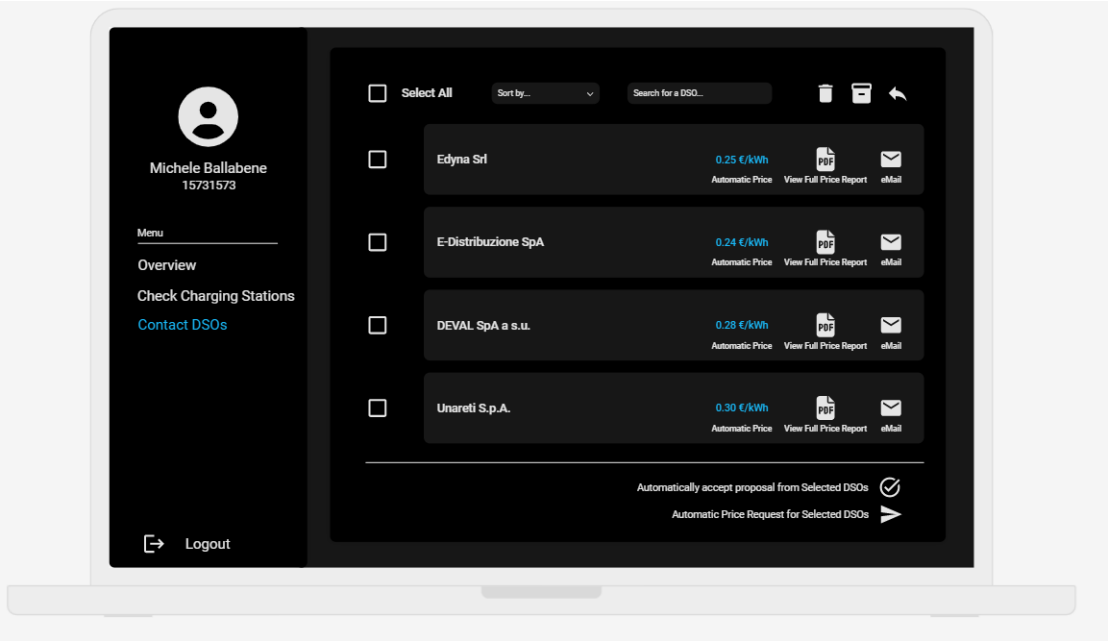
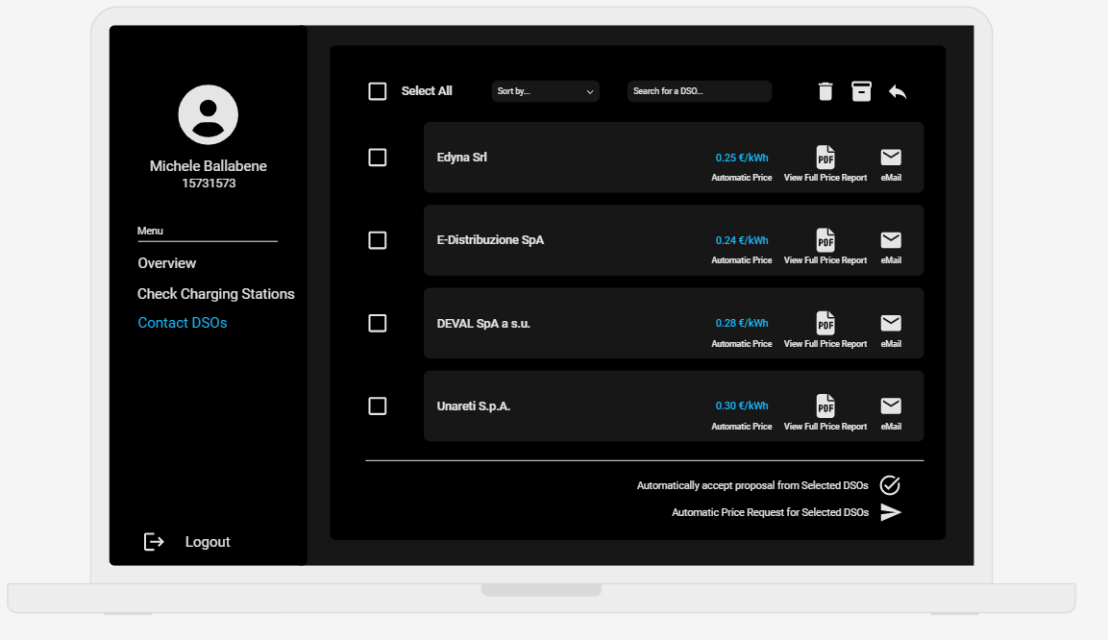


Figure 1: admin desktop application - home



Admin desktop application - charging stations management



Admin desktop application - DSO management

3.1.2 Hardware Interfaces

As the application should be used by many people, there are no strict hardware requirements, so the users just need a mobile phone able to connect to the internet and able to download the eMall application; eMall admins instead just need their company PC where the desktop application to communicate with DSOs and CPMSs is installed, so also in this case there are no strict hardware requirements

3.1.3 Communication Interfaces

eMall system also relies on other services, such as other CPMSs beside eMall's one and the DSOs to acquire energy from, so there should be some communications interfaces (in particular, some dedicated APIs) to be able to exchange data with them:

1. **Interface with other CPMSs**
   Through this interface, eMall is able to retrieve all the status info about the charging stations managed by other CPOs, which include the location, the number of charging sockets of each type of charge and if each one of them is free, their cost per kWh, the amount of energy available in the batteries (if the selected charging station has any), and for each charging vehicle the amount of energy that is being injected.
   Moreover, eMall can retrieve information about special offers a certain CPO has active in a specific moment that can be useful to give smart suggestions to the end user.
   It is important to clarify that there information are retrieved periodically (and automatically) by eMall once every minute – the delay is set and can be changed by eMall admins manually – and they are used to update its database, to offer to the user a service regarding all charging stations of all CPOs and not only eMall ones; if needed, an eMall admin can also request these information whenever they want, bypassing the software scheduling.

2. **Interface with DSOs**
   Through this interface eMall CPMS is able to retrieve prices and more generally energy solutions available at different DSOs which provide energy for charging stations.
   The interaction regards only the energy needed for eMall charging stations obviously because other CPOs will interact themselves with

a list of DSOs they choose to have.
It is important to underline that the communication between eMall CPMS and DSOs is either automatic or initiated by an administrator and it is synchronous, i.e. eMall asks for energy solutions and then the DSOs respond with their proposals, as seen in the sequence diagrams in section 3.2.2 – Purchase energy from DSO.

3.2 Functional requirements

Requirement	Description
R1	The system should allow an unregistered user to create a new account
R2	The system should allow (registered) end users to log in
R3	The system should allow end users to save information about their vehicle(s)
R4	The system should allow end users to save payment methods in their account
R5	The system should allow end users to see a map with information about nearby charging stations
R6	The system should allow end users to start/terminate the charging process at an available charging socket
R7	The system should allow end users to make/cancel a reservation for a charging socket for a certain time
R8	The system should allow end users to make payments for the services used
R9	The system should send end users special offers (when available)
R10	The system should be able to choose which DSO to buy energy from
R11	The system should be able to recharge the batteries of a charging station
R12	The system can send e-mails to the end user
R13	The system can store personal information about the end users
R14	The system can access the user's location
R15	The system can access the user's schedule
R16	The system can control (and show data about) the charging sockets owned directly by e-Mall
R17	The system can communicate with external CPMSs (owned by third-party entities) via specific APIs
R18	The system can communicate with DSOs via specific APIs
R19	The system can interact with human operators at eMall using a desktop application

3.2.1 Mapping on goals

- **G1: Provide the user information about the nearby Charging Points (this includes their cost and the special offers they provide)**
  - **R1**: The system should allow an unregistered user to create a new account
  - **R2**: The system should allow (registered) end users to log in
  - **R5**: The system should allow end users to see a map with information about nearby charging stations
  - **R14**: The system can access the user's location
  - **R16**: The system can control (and show data about) the charging sockets owned directly by e-Mall
  - **R17**: The system can communicate with external CPMSs (owned by third-party entities) via specific APIs
  - **D1**: The end-user has an active Internet connection
  - **D5**: Each external CPO provides an API compatible with the system
  - **D6**: Data provided by other CPOs is valid and trustworthy
- **G2: Make the user able to book a specific Charging Point in a specific timeframe**
  - **R1**: The system should allow an unregistered user to create a new account
  - **R2**: The system should allow (registered) end users to log in
  - **R3**: The system should allow end users to save information about their vehicle(s)
  - **R4**: The system should allow end users to save payment methods in their account
  - **R5**: The system should allow end users to see a map with information about nearby charging stations
  - **R7**: The system should allow end users to make/cancel a reservation for a charging socket for a certain time
  - **R14**: The system can access the user's location
  - **R16**: The system can control (and show data about) the charging sockets owned directly by e-Mall

  - **R17**: The system can communicate with external CPMSs (owned by third-party entities) via specific APIs
  - **D1**: The end-user has an active Internet connection
  - **D2**: The end-user does not input fraudulent information (e.g., fake reservations to disturb other users)
  - **D5**: Each external CPO provides an API compatible with the system
  - **D6**: Data provided by other CPOs is valid and trustworthy

- **G3: Make the user able to start and monitor the charging process, from start to end**
  - **R1**: The system should allow an unregistered user to create a new account
  - **R2**: The system should allow (registered) end users to log in
  - **R3**: The system should allow end users to save information about their vehicle(s)
  - **R4**: The system should allow end users to save payment methods in their account
  - **R5**: The system should allow end users to see a map with information about nearby charging stations
  - **R6**: The system should allow end users to start/terminate the charging process at an available charging socket
  - **R13**: The system can store personal information about the end users
  - **R14**: The system can access the user's location
  - **R16**: The system can control (and show data about) the charging sockets owned directly by e-Mall
  - **R17**: The system can communicate with external CPMSs (owned by third-party entities) via specific APIs
  - **D1**: The end-user has an active Internet conncetion
  - **D2**: The end-user does not input fraudulent information (e.g., fake reservations to disturb other users)
  - **D3**: The vehicle's details about the battery capacity are true and exact

  - **D4**: Physical entities at eMall's charging stations (e.g., sockets, batteries) work properly
  - **D5**: Each external CPO provides an API compatible with the system
  - **D6**: Data provided by other CPOs is valid and trustworthy

- **G4: Notify the user when the charging process is finished or when the booked timeframe expires**
  - **R1**: The system should allow an unregistered user to create a new account
  - **R2**: The system should allow (registered) end users to log in
  - **R6**: The system should allow end users to start/terminate the charging process at an available charging socket
  - **R13**: The system can store personal information about the end users
  - **R16**: The system can control (and show data about) the charging sockets owned directly by e-Mall
  - **R17**: The system can communicate with external CPMSs (owned by third-party entities) via specific APIs
  - **D1**: The end-user has an active Internet connection
  - **D3**: The vehicle's details about the battery capacity are true and exact
  - **D4**: Physical entities at eMall's charging stations (e.g., sockets, batteries) work properly
  - **D5**: Each external CPO provides an API compatible with the system
  - **D6**: Data provided by other CPOs is valid and trustworthy

- **G5: Make the user able to pay for the charging service**
  - **R1**: The system should allow an unregistered user to create a new account
  - **R2**: The system should allow (registered) end users to log in
  - **R4**: The system should allow end users to save payment methods in their account
  - **R8**: The system should allow end users to make payments for the services used

  - **R13**: The system can store personal information about the end users
  - **D1**: The end-user has an active Internet connection

- **G6: Provide the user with smart suggestions about the charging port to go charge to, based on their location, schedule, prices and special offers (if available)**
  - **R1**: The system should allow an unregistered user to create a new account
  - **R2**: The system should allow (registered) end users to log in
  - **R3**: The system should allow end users to save information about their vehicle(s)
  - **R9**: The system should send end users special offers (when available)
  - **R13**: The system can store personal information about the end users
  - **R14**: The system can access the user's location
  - **R15**: The system can access the user's schedule
  - **R17**: The system can communicate with external CPMSs (owned by third-party entities) via specific APIs
  - **R18**: The system can communicate with DSOs via specific APIs
  - **D1**: The end-user has an active Internet connection
  - **D5**: Each external CPO provides an API compatible with the system
  - **D6**: Data provided by other CPOs is valid and trustworthy
  - **D8**: The information upon which the system provides suggestions (location, calendar...) is valid and reflects the user's real status

- **G7: Manage the charging stations that are directly owned by eMall**
  - **R1**: The system should allow an unregistered user to create a new account
  - **R2**: The system should allow (registered) end users to log in
  - **R10**: The system should be able to choose which DSO to buy energy from

- **R11**: The system should be able to recharge the batteries of a charging station
- **R16**: The system can control (and show data about) the charging sockets owned directly by e-Mall
- **R18**: The system can communicate with DSOs via specific APIs
- **D4**: Physical entities at eMall's charging stations (e.g., sockets, batteries) work properly
- **D7**: Each DSO provides an API compatible with the system

3.2.2 Use case diagrams

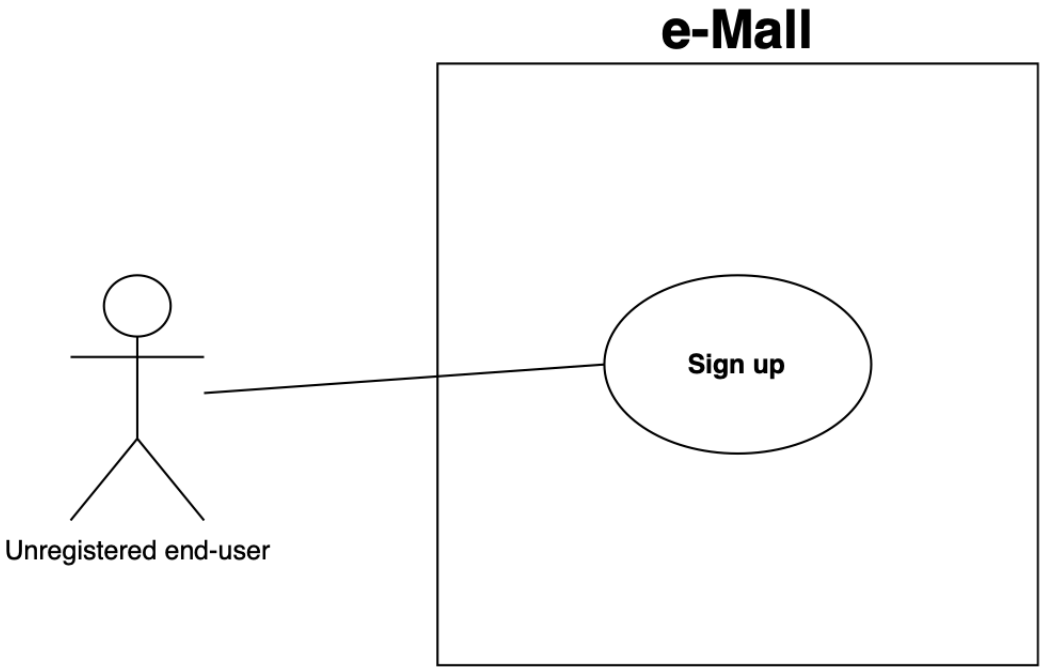


Figure 2: use case diagram representing the interaction of an unregistered user with the system

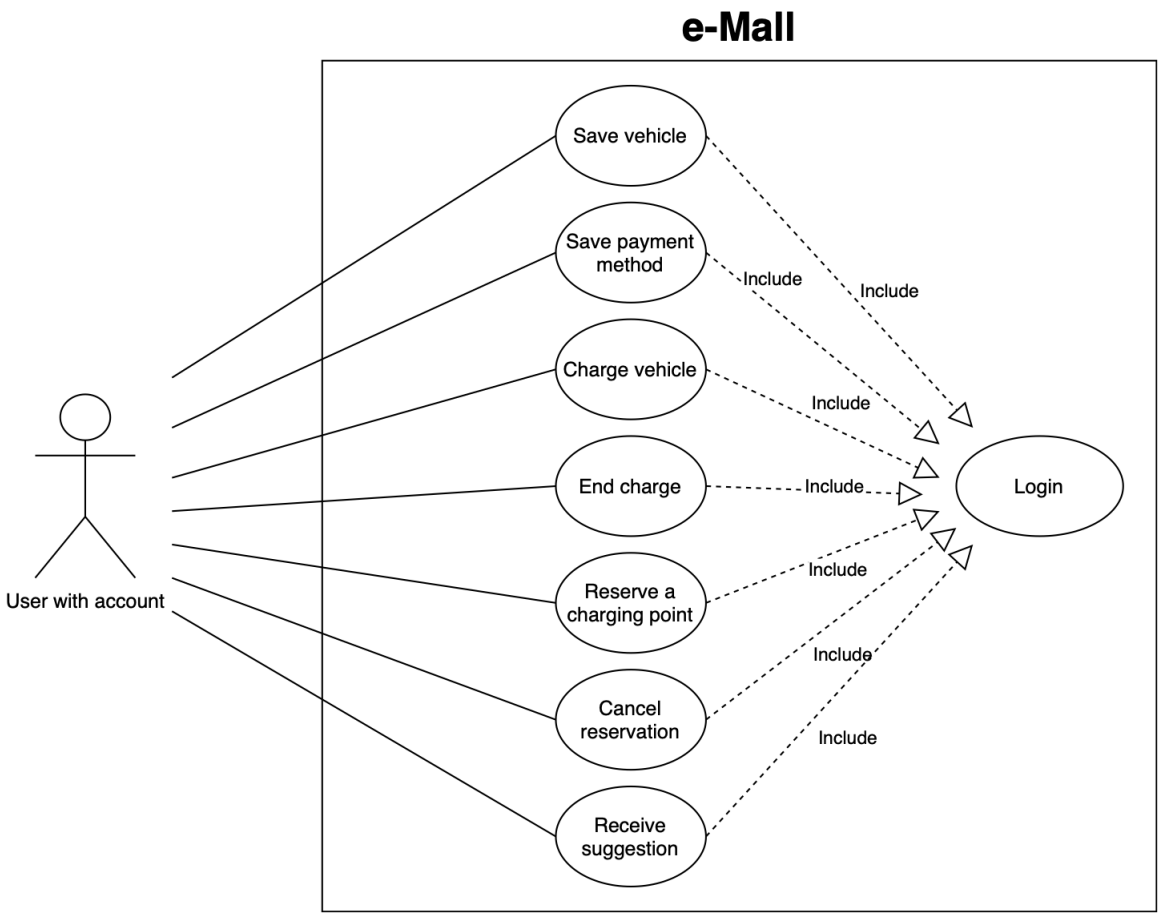


Figure 3: use case diagram representing the interactions of a successfully logged-in user with the system

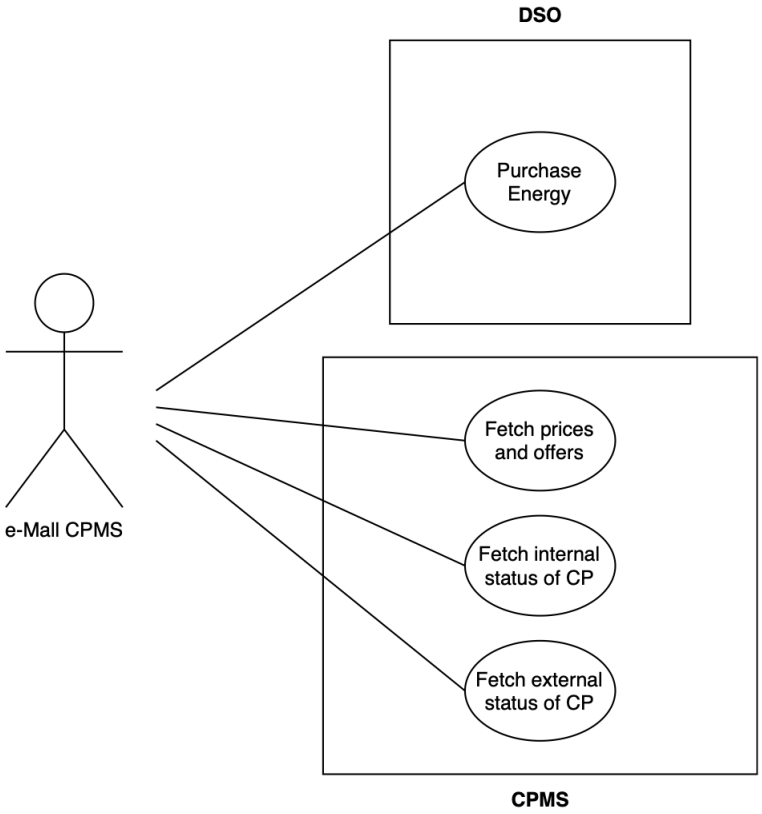


Figure 4: use case diagram representing the interactions of the system with an external CPMS

3.2.3 Use cases

In the following sections are explained various use cases of eMall system, with use case tables and sequence diagrams.

1. **Sign up of a new user**

Actor(s)	User
Entry conditions	The user has downloaded the application on his smartphone and does not have a previously created account
Event flow	1. The user clicks on the "Create new account" button 2. The user enters their name, surname, email and phone number and presses the "Next" button 3. An email with a link to confirm it is sent to the address specified by the user 4. If the link in the email is clicked within 30 minutes, then the email address is correctly associated to the user's profile and can be used to log into the account 5. After receiving from the backend a confirmation of the click on the link, a success message is showed, welcoming the new user into eMall
Exit condition	A new account with a verified email address is created
Exceptions	• The user does not specify all the required data or insert badly formatted values in one or more fields • A user with the same email or the same phone number is already present in the DBMS • The system is unable to send the verification email to the provided address • The user does not click on the verification link within 30 minutes from the dispatch For all the listed cases eMall will provide to the user an appropriate error message

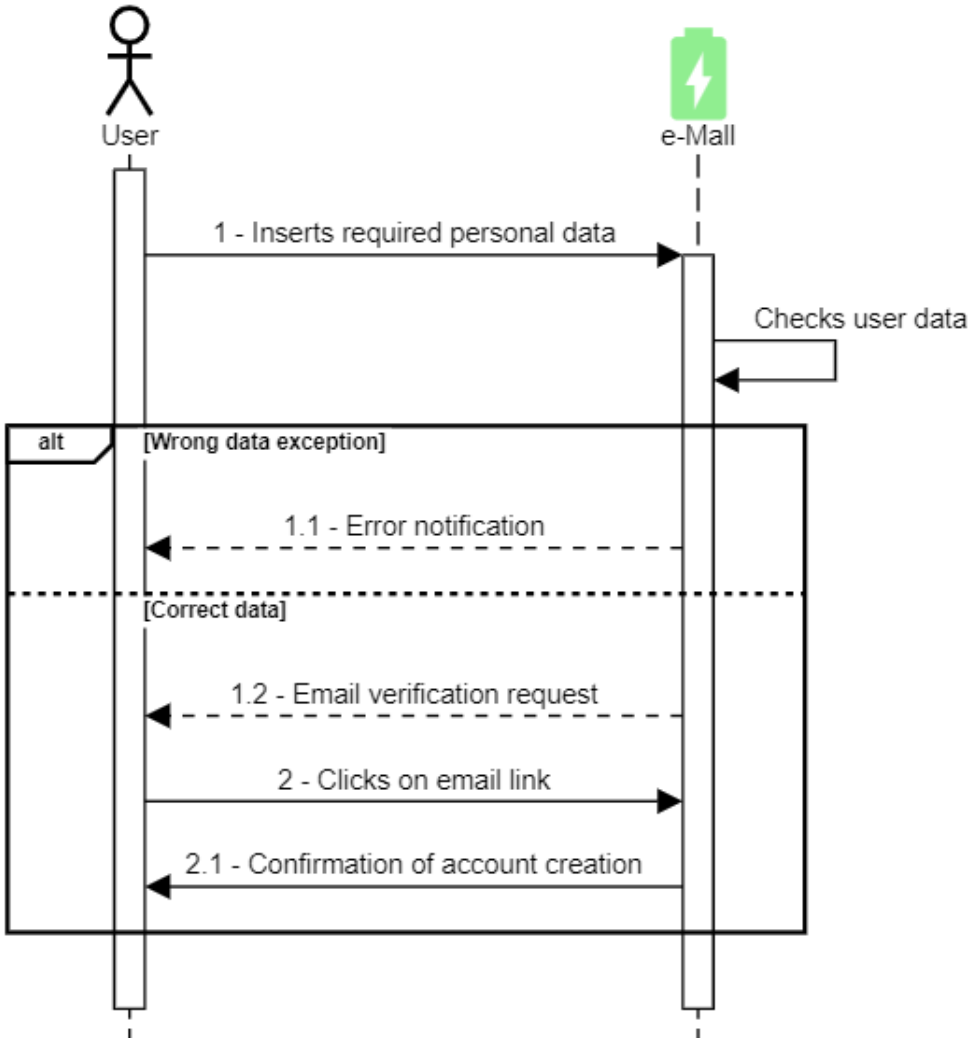


Figure 5: sign up of a new user (sequence diagram)

2. **Login of registered user in eMall application**

Actor(s)	User
Entry conditions	The user already has an account and a ready mobile phone to use
Event flow	1. The user proceeds to open the eMall application 2. The user waits for the login form to appear 3. The user inserts mail and password they used to sign up 4. The user clicks on the “login” button 5. eMall application evaluates the inserted credentials and if they’re correct lets the user in
Exit condition	The user is logged in the eMall application and is able to exploit its functionalities
Exceptions	• The user does not insert all required data The application does not let them click on the “Confirm” button • The user does not insert the correct data they used to sign up The application shows an “invalid credentials” pop up message

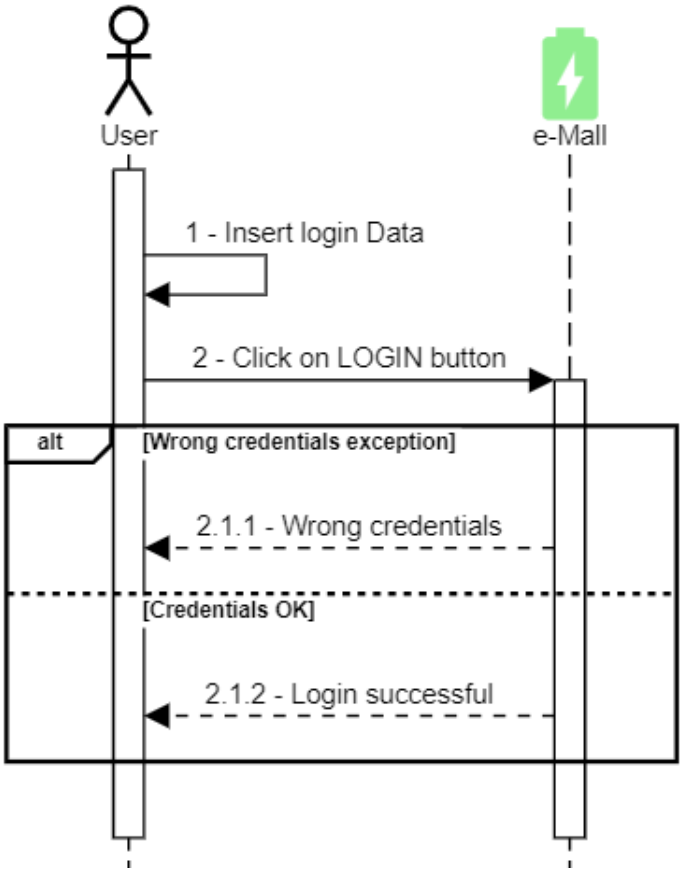


Figure 6: login of registered user in eMall application (sequence diagram)

3. **User saves a vehicle into their account**

Actor(s)	User
Entry conditions	The user already has an account and a ready mobile phone to use
Event flow	1. The user proceeds to open the eMall application 2. The user goes on the "My account" page and taps on the "My vehicles" section 3. The user clicks on the "Add vehicle" button marked by a "+" sign 4. The user inserts all the required data: name for the vehicle, model and battery capacity 5. The user clicks on the "Confirm" button 6. The application shows a "Vehicle saved" pop up and updates its database in the backend 7. The application redirects the user to the "My vehicles" section
Exit condition	The inserted vehicle is saved into the account of the user and associated to them in the database
Exceptions	• The user does not insert all required data The application does not let them click on the "Confirm" button

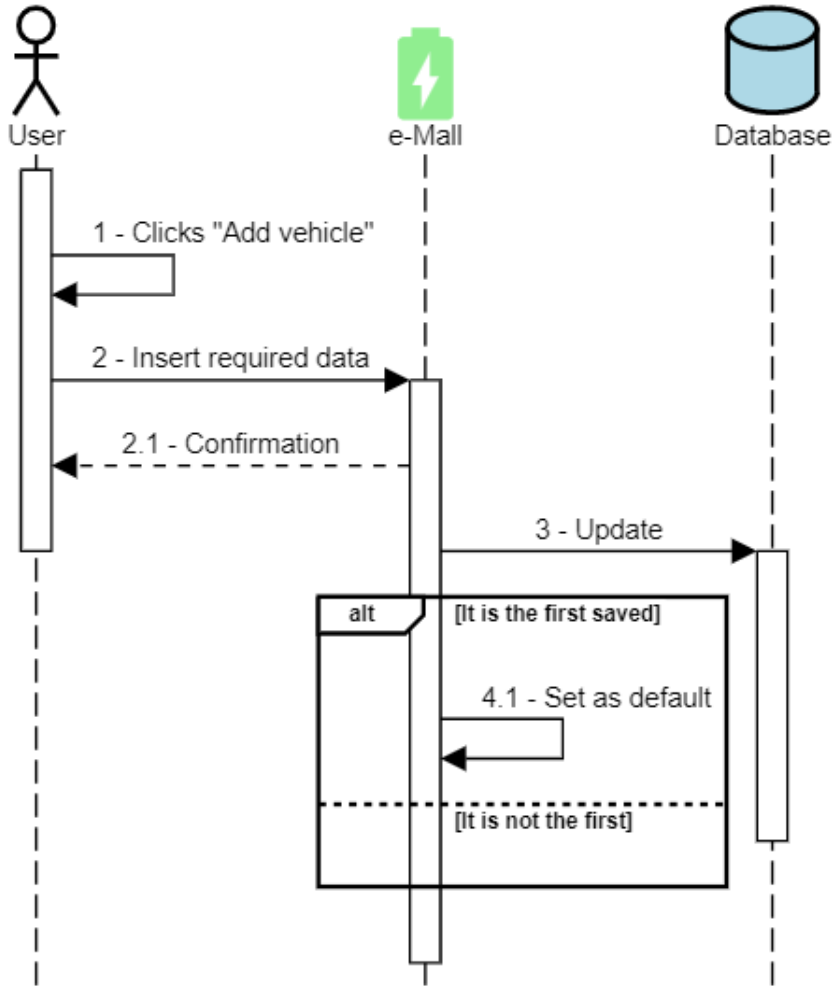


Figure 7: user saves a vehicle into their account (sequence diagram)

4. **User saves a payment method into their account**

Actor(s)	User
Entry conditions	The user already has an account and a ready mobile phone to use
Event flow	1. The user proceeds to open the eMall application 2. The user goes on the "My account" page and taps on the "Wallet" section 3. The user clicks on the "Add payment method" button marked by a "+" sign 4. The user selects the payment method type between PayPal and Credit Card 5. If the selected method is CC, then the user inserts card holder name, card number, expiry date and cvv; else, if the selected method is PayPal, the user gets redirected to the PayPal login page and logs in 6. The user clicks on the "Confirm" button 7. The application shows a "Payment method saved" pop up 8. The user gets redirected to their "Wallet" section
Exit condition	The inserted payment method is saved into the account of the user and associated to them in the database
Exceptions	• User does not insert all required data The application does not let them click on the "Confirm" button • User inserts wrong formatted data in the credit card fields The application shows an "Incorrect data" message

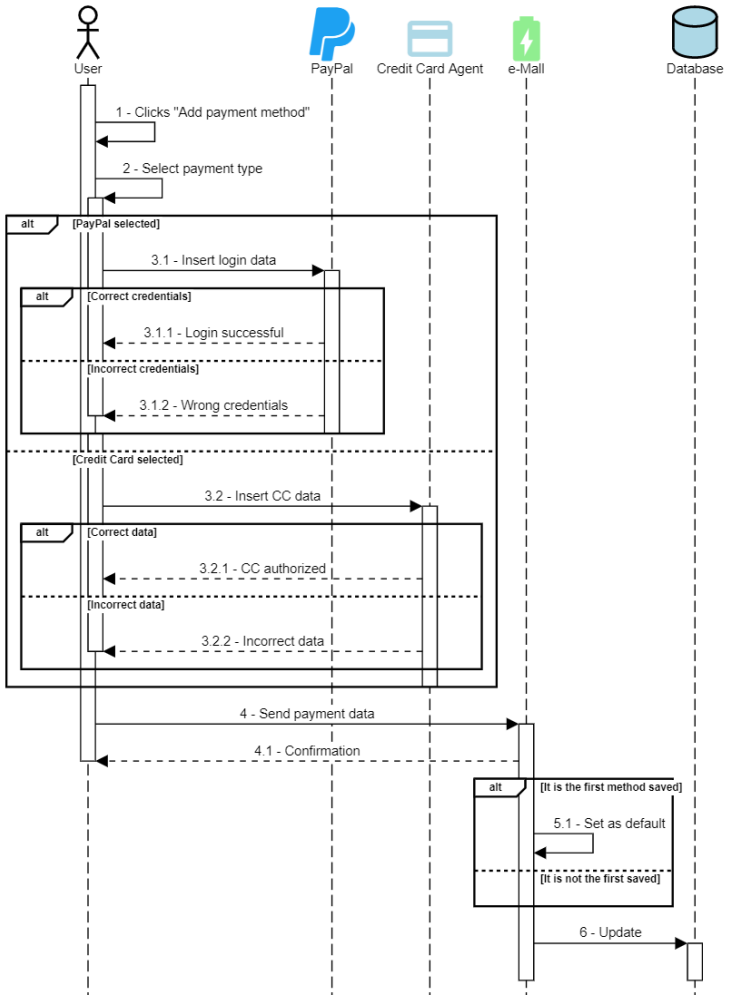


Figure 8: user saves a payment method into their account (sequence diagram)

5. **Charge the vehicle on an available socket**

Actor(s)	User
Entry conditions	• The user is logged into the system • The user has at least one payment method and one vehicle correctly registered on the application • The user is in 100 meters range from the charging station • The user has no pending payment for a previous recharge
Event flow	1. The user selects the station he wants to charge the vehicle on, clicking the corresponding indicator on the map 2. The user clicks on "Start charging" button 3. The user specifies the ID of the socket they want to connect to 4. The user selects one of their vehicles and one of their payment methods and proceeds with the operation 5. The user selects a date and time in which the charge will stop or a target charge percentage to be reached – the application provides an estimation of the time required to fully charge the vehicle 6. The user clicks on "START" button 7. The charging socket is physically unlocked, so the user can take it and plug it into the vehicle. This action is done directly by eMall CPMS (if the chosen station belongs to the ones owned by eMall) or by the external CPMS which manages the selected charging point, appropriately notified by eMall system

Event flow	8. The user can check the status of the recharge from a dedicated popup in the homepage, showing the energy injected in the vehicle, the current amount to be paid and the estimated battery level at the end of the recharge
Exit condition	The charging socket specified by the user is unlocked and can be used to charge the car. The system keeps track of this recharge
Exceptions	• The date/time of recharge end is badly formatted or too far • The target charge percentage is invalid In these two cases the application shows an appropriate error message to the user • The user doesn't plug the socket into the vehicle within 10 minutes from the final confirmation The system cancels the operation, sending a notification to the user

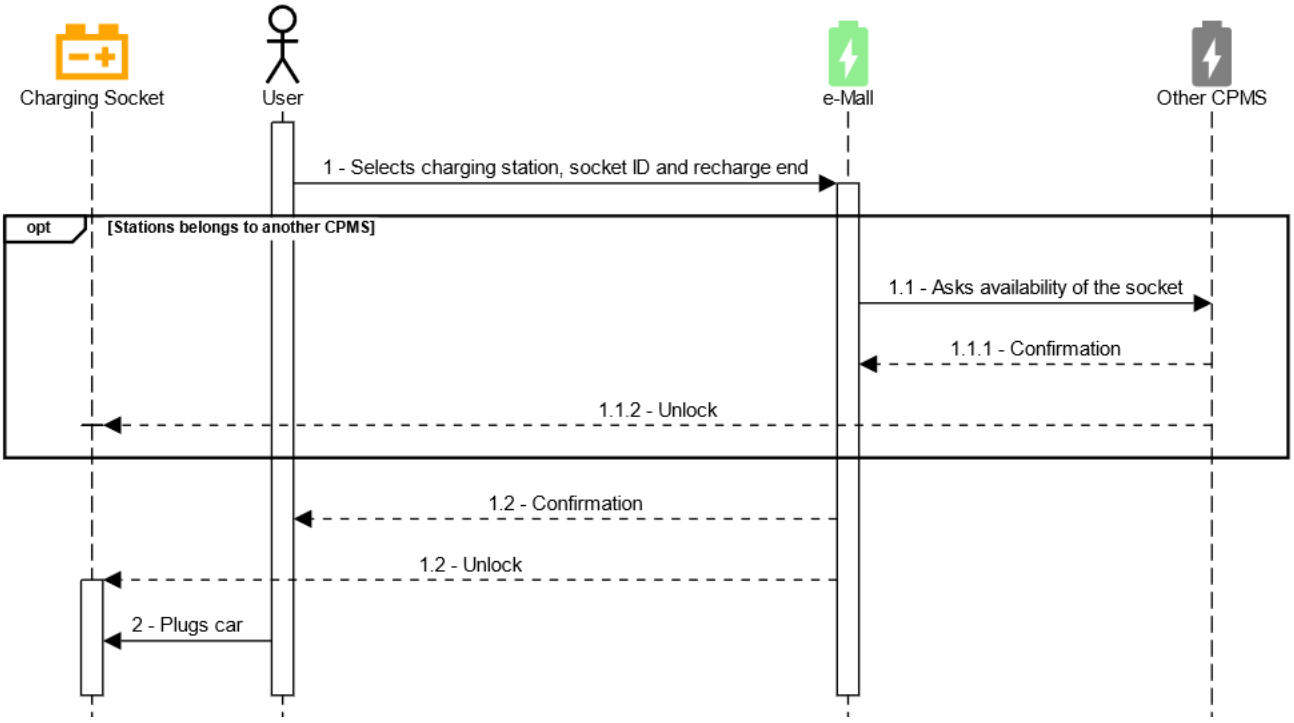


Figure 9: user charges the vehicle on an available socket (sequence diagram)

6. **Ending the recharge of a vehicle**

Actor(s)	User
Entry conditions	• The user is logged into the system • The user has a recharge in progress
Event flow	1. The user clicks on the popup regarding the recharge, which can be found in the homepage of the application 2. The user clicks on "STOP" button 3. The socket stops the delivery of energy 4. The user unplugs the socket from the car 5. The system takes the owed amount for the recharge from the user's selected payment method. If the station is owned by a CPMS different from eMall's one, it is notified about the correct outcome of the transaction. 6. The recharge now appears in the Recharge History section of the application and its status depends on the outcome of the payment
Exit condition	The user can unplug the socket from their car and pays for the provided service
Exceptions	• The chosen payment method does not have enough funds to pay for the recharge In this case, the recharge status shows that the payment went wrong and the user is prevented to do further actions on the application until they successfully retry the payment, even with a different method

Note: the sequence diagram shows the case in which the station is owned by eMall, in order to make the diagram clearer. If the station is managed by another CPMS the interactions are almost the same (the only difference is that e-Mall, after every significant operation, informs the other CPMS, as described in BPMN diagram)

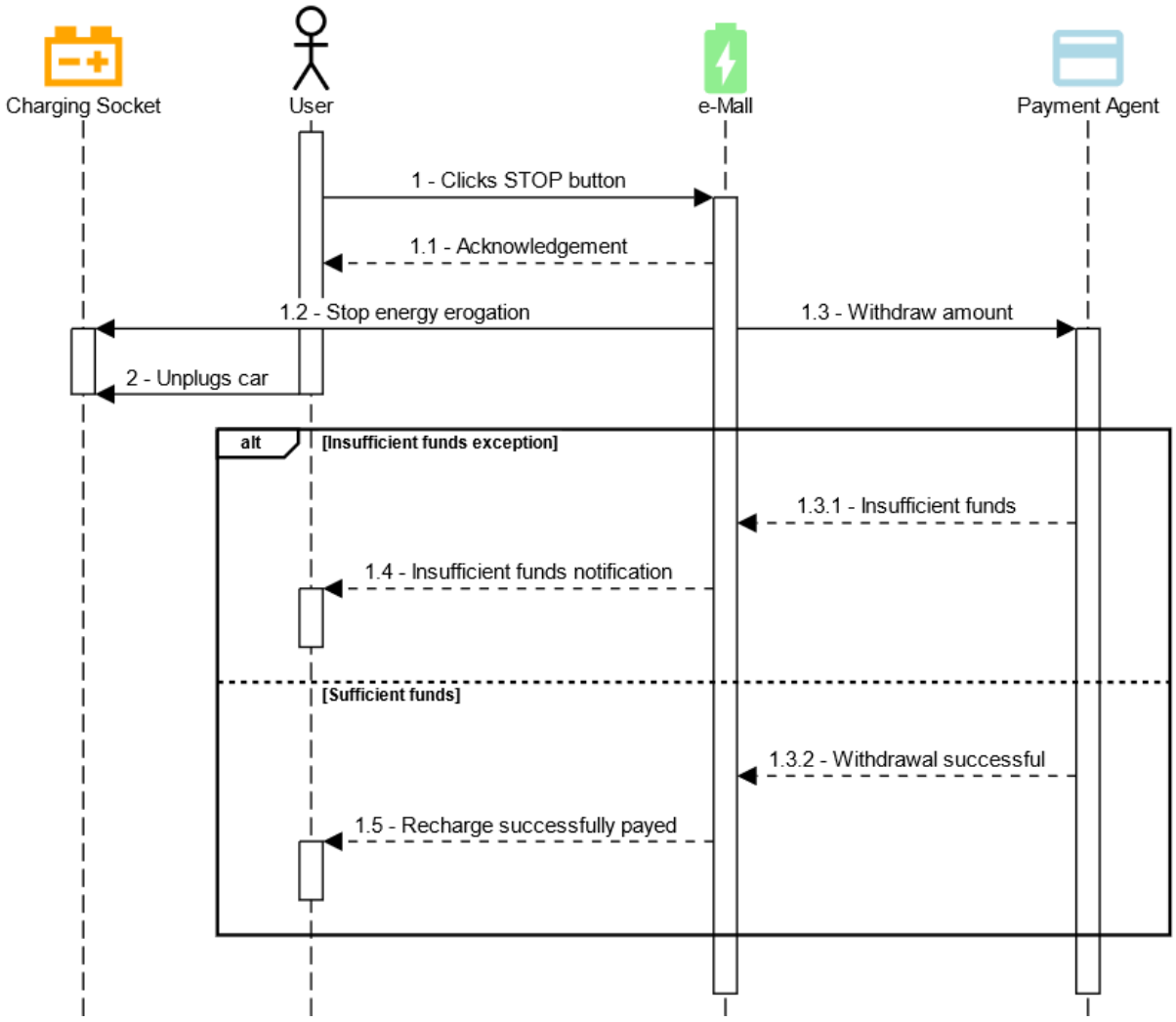


Figure 10: user ends the recharge of a vehicle (sequence diagram)

7. **Reservation of a socket for a certain time**

Actor(s)	User
Entry conditions	• The user is logged into the system • The user has at least one payment method and one vehicle correctly registered on the application • The user has no pending payment for a previous recharge
Event flow	1. The user clicks on the Schedule icon in the bottom left of the screen 2. A calendar popup opens up and the user can choose the desired socket type (fast, rapid or slow), date and a time interval of availability 3. A map with all the charging stations that are available for reservations and match the previous filters 4. The user clicks on one of the charging stations highlighted on the map 5. The user selects one of his vehicles and one of his payment methods for the reservation 6. A text written in red alerts the user that half of the estimated owed amount of the charge session will be retained from the selected payment method 7. The user clicks on the "Reserve your spot" button 8. The amount described above is taken from the user's payment method. Meanwhile, the user is redirected to "My reservations" section of the app

Exit condition	A new reservation for a vehicle on a specific socket type in a certain station and in a precise time interval is created
Exceptions	• The user chooses an invalid or too short time interval • The user chooses an invalid date • The payment method chosen by the user does not have sufficient funds to authorize the transaction For all the listed cases eMall will cancel the pending reservation and show an appropriate error message to the user

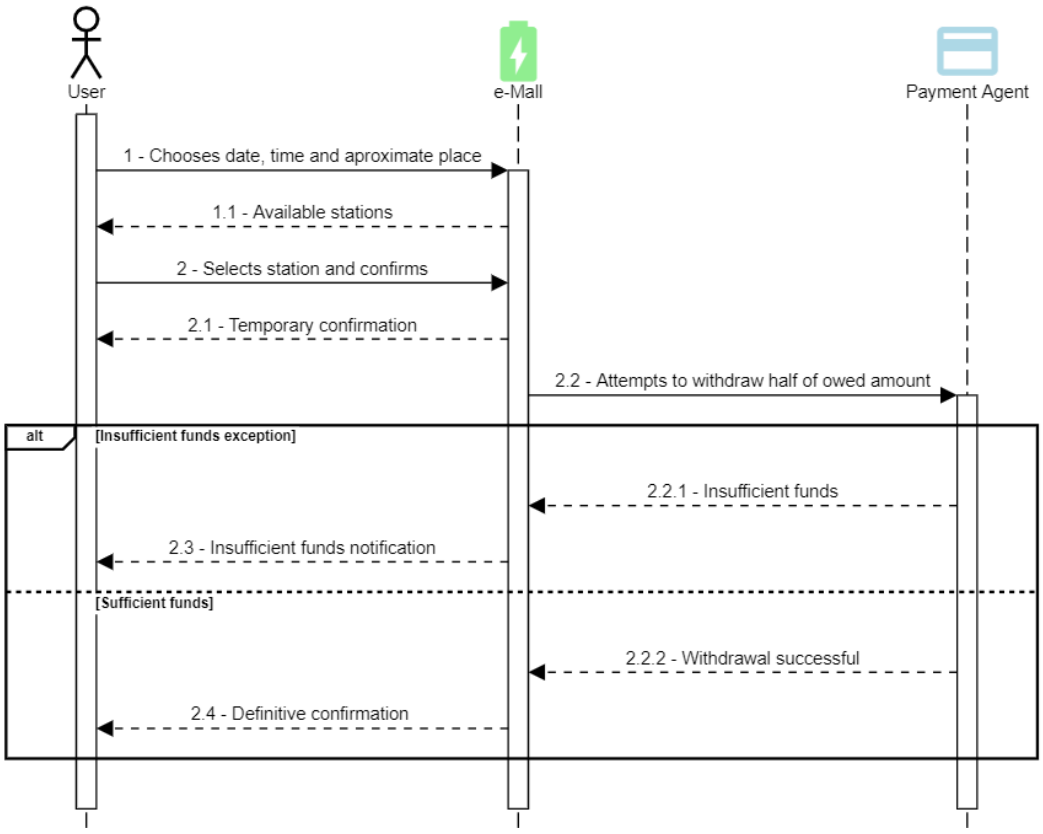


Figure 11: reservation of a socket for a certain time (sequence diagram)

8. **Cancel a reservation**

Actor(s)	User
Entry conditions	The user has a reservation for a certain charging station and time interval
Event flow	1. The user clicks on the reservation they want to cancel from the list showed in the section "My Reservations" 2. The user clicks on "Cancel" button 3. The application asks for a confirmation and, if the cancel action is taken within two hours from the start of the recharge, reminds the user that he will be charged with half of the estimated owed amount 4. The user clicks on "Confirm" button
Exit condition	The user successfully cancels his reservation from the system and the previously booked socket can be considered free in that time interval
Exceptions	• The user canceled the reservation within two hours from the moment in which the recharge would have started and does not have sufficient funds on the provided payment method As for any other error due to insufficient funds, the user is notified about their payment failing and they are prevented to do further actions on the application until they successfully retry the payment, even with a different method

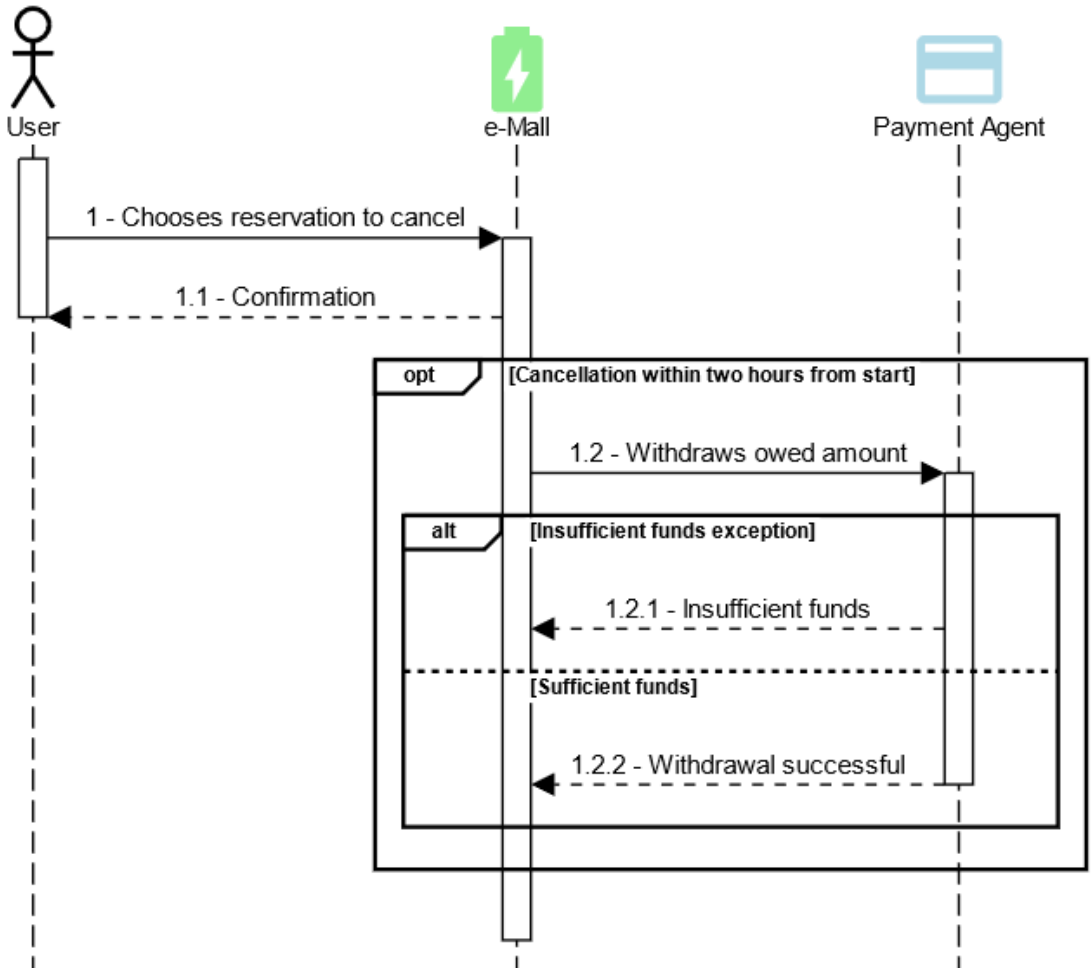


Figure 12: user cancels a reservation (sequence diagram)

9. **User receives a suggestion due to a special offer**

Actor(s)	• eMall CPMS • User
Entry conditions	The end user has already an account and has already given the application permissions to access their position
Event flow	1. eMall CPMS automatically (or actively by manual input of an operator) fetches the different prices of charging from different CPOs 2. The CPMS detects a special offer by a certain CPO which manages some charging stations in the city of the user and notifies all the users in that city 3. The user receives a notification on their phone from eMall application 4. The user clicks on it and the eMall mobile application opens in their "My suggestions" section 5. The user sees the list of their past suggestions and the latest one at the top marked as new 6. The user sees the suggested charging stations, the respective addresses and the discounted prices 7. The user takes note of the offers and closes the application
Exit condition	The user knows about the offer and can decide to go and charge to those specific charging stations
Exceptions	• Network errors • No special offers found The application does not send any notifications

10. **User receives a suggestion due to their schedule**

Actor(s)	• eMall CPMS • User
Entry conditions	The end user has already an account and has already given the application permissions to access their schedule
Event flow	1. eMall CPMS automatically (or actively by manual input of an operator) fetches user's schedule 2. The CPMS detects the fact that the user has an appointment in a location near to a charging point in a not booked timeframe 3. The user receives a notification on their phone from eMall application 4. The user clicks on it and the eMall mobile application opens in their "My suggestions" section 5. The user sees the list of their past suggestions and the latest one at the top marked as new 6. The user sees the suggested charging station and the suggested timeframe 7. The user can accept or dismiss the suggestion 8. If the user accepts the suggestion the application proceeds to book the selected charging station in the selected timeframe; if the user dismisses the suggestion the application does not do anything
Exit condition	Either the user has a reservation in their "My reservations" section regarding the selected charging station and the selected timeframe, or they know about the suggestion and they are able to accept it later
Exceptions	• Network errors • No schedule compatibility found The application does not send any notifications

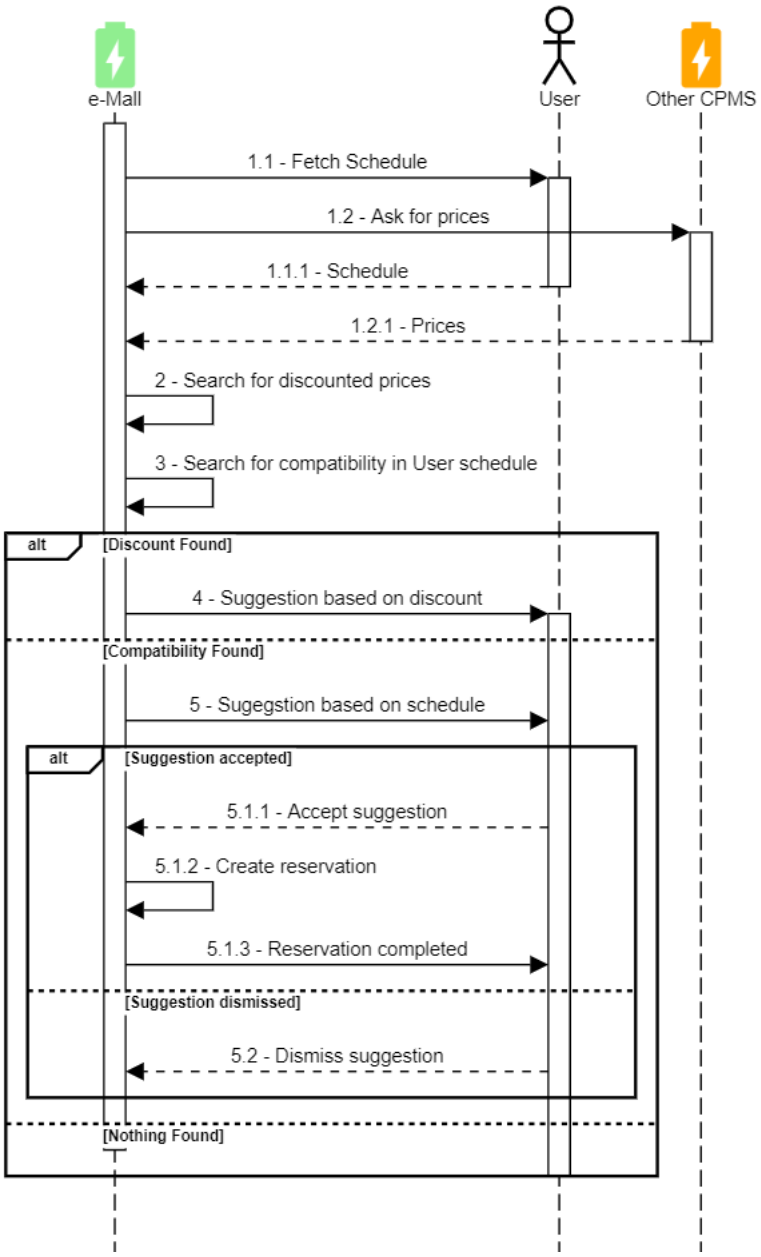


Figure 13: user receives a suggestion, due to their schedule or special offers (sequence diagram)

11. **The CPMS purchases energy from a DSO**

Actor(s)	• eMall CPMS • DSOs
Entry conditions	• eMall CPMS has certain API to communicate with all the DSOs it can purchase energy from, and they have an API to send needed data back. • A special desktop app to communicate with DSOs is installed on eMall computers
Event flow	1. eMall CPMS asks the current energy price for all DSO it is able to contact 2. The ones which decide to answer then send the prices and the monthly based solutions for their energy (if they have any) 3. eMall CPMS then evaluates the received offers following several criteria, for example the prices and the availabilities 4. eMall CPMS eventually decides to purchase energy from a list of DSOs (that can be also made of only one DSO) and sends confirmation to them and refuses the other offers. It is important to underline that, as seen from this message exchange, the communication between CPMS and DSOs is synchronous, and it is as important to clarify that in each moment the manual work of an operator can interrupt and modify the automatic work of the CPMS. The decision of the operator will always have a higher priority with respect to the ones made by the software but if no one interferes the process keeps going with the purchase

Event flow	5. The DSOs which received the confirmation from eMall CPMS then send a further confirmation for the successful purchase 6. eMall CPMS updates the prices of the company's charging stations based on the new prices agreed with the DSOs.
Exit condition	The DSOs which eMall purchased energy from provide energy to eMall charging stations
Exceptions	• Network errors The application will show an appropriate error message to the operator • CPMS automatic schedule stops working eMall operators will be notified to restore it

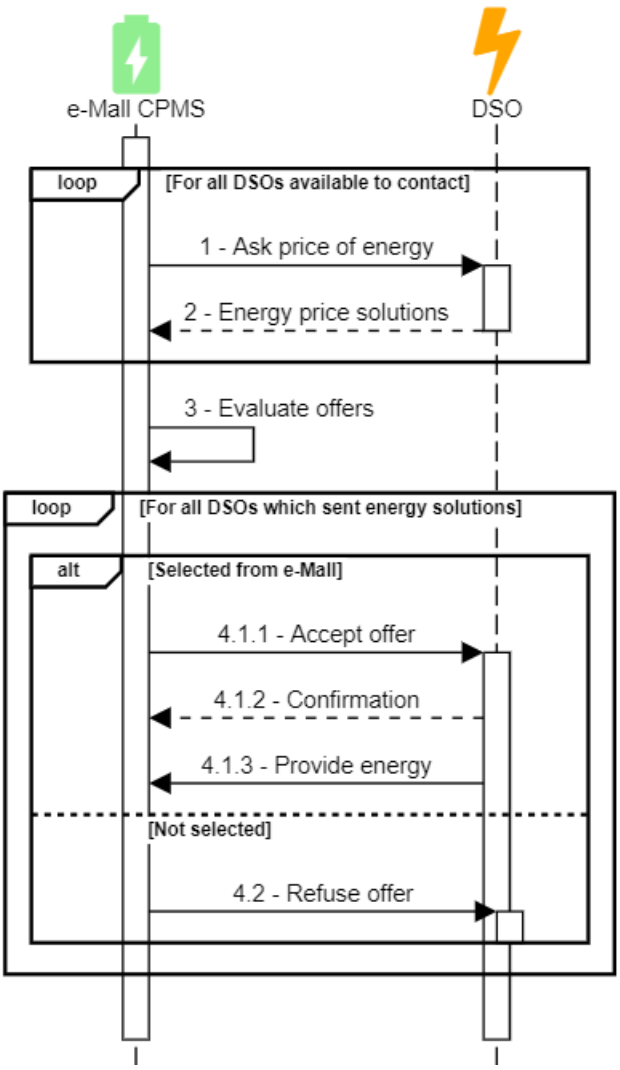


Figure 14: the CPMS purchases energy from a DSO automatically (sequence diagram)

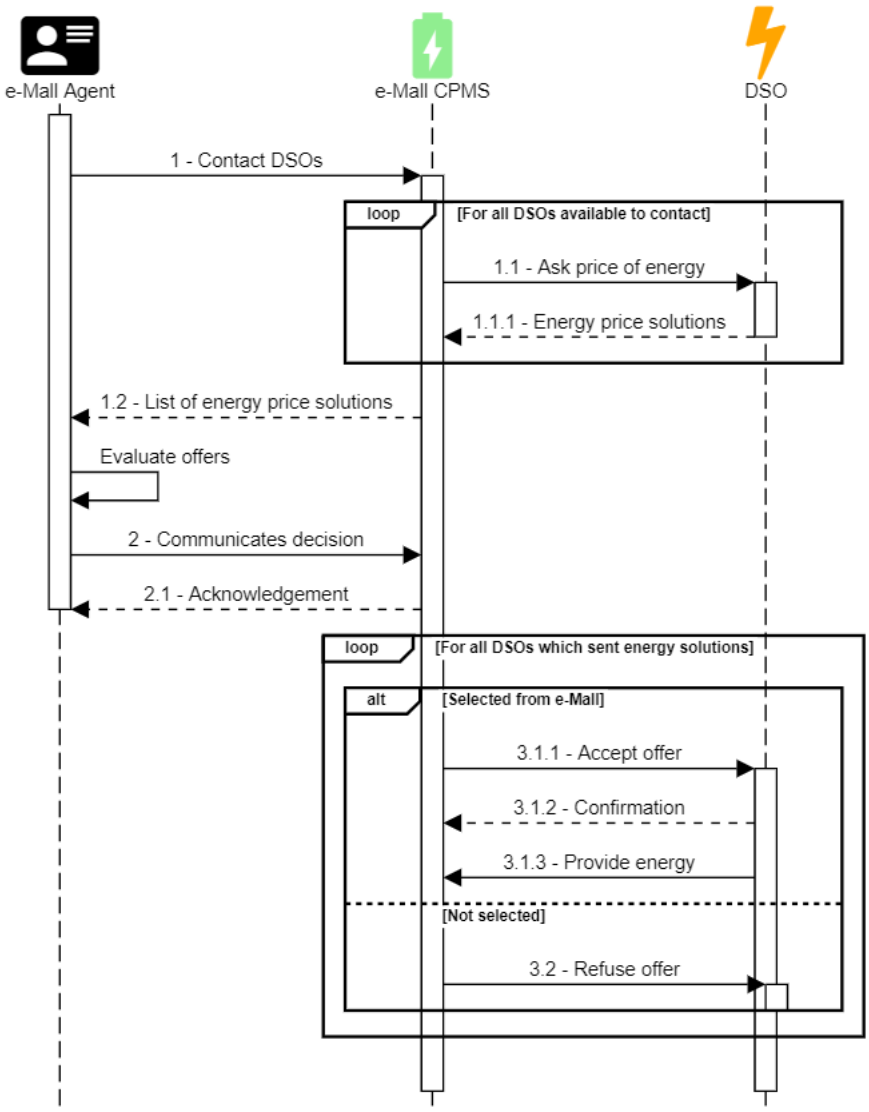


Figure 15: the CPMS purchases energy from a DSO with input from human agent (sequence diagram)

12. **Charge the batteries of a charging station**

Actor(s)	• eMall CPMS
Entry conditions	• The CPMS manages some station which have batteries to store energy
Event flow	1. Once every 30 minutes – automatically – or upon request of a human agent, the CPMS checks the status of the batteries of all the stations that have them 2. For each station, decide whether and how much to recharge batteries. This decision can be influenced by multiple factors such as the average amount of energy required by the station in a fixed period and the cost to purchase it and can be taken automatically by an algorithm based on the above data or manually by a human operator. The automated decisions can be reverted by an agent within 15 minutes from their submission – see section 2.1.2 for details and motivations of this choice. 3. For each station, recharge batteries of the amount computed in the previous point
Exit condition	For every station that has the possibility to store energy in its batteries, a decision on whether and how much to recharge them is taken
Exceptions	• An operator rolls back the actions computed by the algorithm within 15 minutes No effect on the system at all: the decisions are reverted, so no battery is recharged

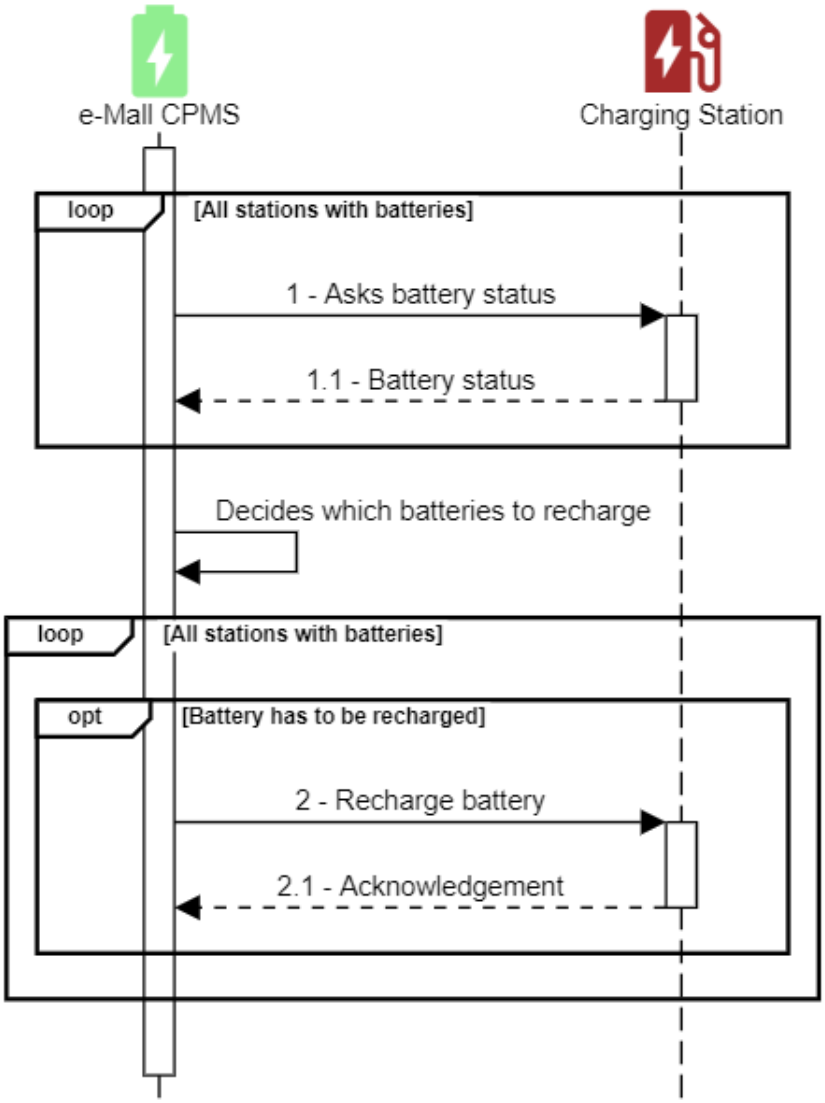


Figure 16: charge the batteries of a charging station automatically (sequence diagram)

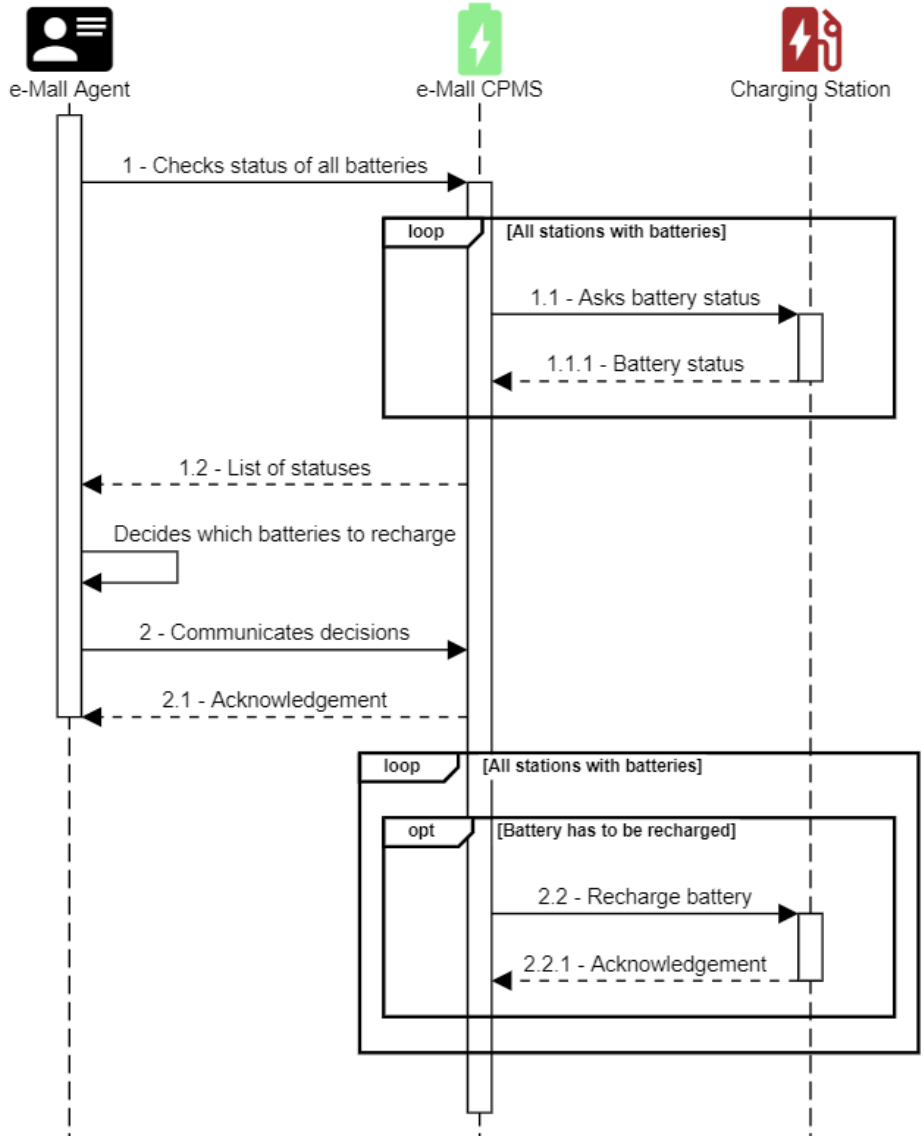


Figure 17: charge the batteries of a charging station with human operator (sequence diagram)

3.2.4 Mapping on requirements

Use case	**ID**
Sign-up of a new user	UC1
Login of a registered user in e-Mall application	UC2
User saves a vehicle into their account	UC3
User saves a payment method into their account	UC4
Charging of the vehicle on an available socket	UC5
Ending the recharge of a vehicle	UC6
Reservation of a socket for a certain time	UC7
Cancel a reservation	UC8
User receives a suggestion due to a special offer	UC9
User receives a suggestion due to their schedule	UC10
The CPMS purchases energy from a DSO	UC11
Charge the batteries of a charging station	UC12

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16	R17	R18	R19
UC1	•											•	•						
UC2		•											•						
UC3		•	•										•						
UC4		•		•									•						
UC5		•	•	•	•	•							•	•		•	•		
UC6		•		•		•		•					•			•	•		
UC7		•	•	•	•		•	•					•	•					
UC8		•		•			•	•					•						
UC9		•							•				•	•		•	•		
UC10		•					•		•				•		•				
UC11										•								•	•
UC12											•							•	•

3.3 Performance requirements

The system should be very responsive, to make the user experience as fast as possible, ideally with a response time below 2 seconds: users will probably have to use the app while on their daily routine, so in this way it won't be a hurdle for them.

Because this is going to be a utility app, it should also be lightweight (both in app size and Internet bandwidth consumption) to be usable by the largest amount of people possible.

Finally, the design of the system's backend should take into account the potentially very high number of users connected at the same time: given the current number of electric cars in Italy, a system designed for 100.000 simultaneous connections should be more than enough for the short/medium term. Room for later expansions on this front would be advised, given the fast rate of adoption that electric vehicles are seeing.

3.4 Design constraints

3.4.1 Hardware constraints

The eMall system will be available to end users through a mobile app, so they need to have a smartphone with an active Internet connection. Because of the "on-the-go" nature of the eMall app, it is very unlikely that a Wi-Fi connection would be usable in practice, so for most cases a mobile data plan will be needed.

The system will also be accessible to eMall admins (human operators) via a dedicated desktop application. No special hardware requirements are expected for this purpose, a normal desktop PC should be enough.

3.4.2 Privacy and Data security constraints

As the system is going to be deployed in the European market, it must be GDPR compliant for what concerns personal data protection. Every user must read and accept the privacy policy when they sign-up for the service, otherwise they won't be able to use it. When the system communicates with third-party entities (external CPOs, DSOs...) it must not share any

personal information about costumers without their explicit consent.

The system must also guarantee strong levels of protection against data theft, as it contains both personal information and payment methods. To do so, every communication between the server and the client must be encrypted following industry standards; data stored in the DBMSs must also be adequately protected.

3.5 Software system attributes

3.5.1 Reliability

The system must have a high reliability, i.e. the capacity of running continuously for long time. This is a very important requirement for eMall, because of the difficulty of recovering from a failure in such a system which keeps track of recharge progress in real time. Restoring the states of all recharges in progress and recomputing all costs and information about them could involve many complex and tricky fixes, so it's highly preferable to avoid this situation.
The maximum load for eMall is expected in the morning and in the first afternoon, when most of the people go out from their home by car, so it's key to adequately project and size the system in such a way that it can handle a large number of concurrent accesses (at least 10000).
Preventive maintenance has to be performed regularly on all the servers which eMall is deployed on and which its services are offered on; moreover, many of the machines could be duplicated in order to better cope with transient failures.

3.5.2 Availability

The system should be available as much as possible for the same reasons described in the above paragraph, as well as for guaranteeing the best possible user experience. A user that is in a hurry and wants to recharge their vehicle must not indeed experience interruptions of services, otherwise they would probably switch to a competitor application offering similar services.
Given these premises, we think that a three nines availability (99.9%) could be acceptable for eMall, because it would imply an average downtime of

9 hours/year, during which targeted maintenance interventions could be performed.
It's important to underline that the full availability of the system depends also on external APIs of DSOs and other CPMSs – and therefore not under control of eMall, but a "minimum availability" (e.g., showing only eMall charging stations) should always be assured in the uptime of the system.

3.5.3 Security

The system stores many personal data about its users that must be kept private whatever it takes, so security is one of the main concerns to think of while planning the development of eMall.
It's crucially important that there are no faults leading to the leakage of personal identities, and, above all, information about credit cards. For this reason, the communication between all the parties involved in the system has to be on a secure channel and encrypted following the guidelines of use of the TLS protocol.
In addition, eMall must guarantee that every operation performed on the system comes from a user that is authenticated and has the authorization to fulfill it. At last, there should be a password recovery system for users who happen to forget theirs, but the passwords themselves must remain encrypted also in eMall database, not to violate the privacy of the users.

3.5.4 Maintainability

The system must be designed and implemented in such a way that it can be easily maintained, extended with further functionalities and scaled in case the current infrastructure would not fit the growing number of users.
This requirement can be achieved through the use of appropriate and well-known design patterns (such as MVVM) and the presence of documentation and comments to help future developers understand every part of the code.

3.5.5 Portability

The system has been thought to be a mobile application rather than a web app, for convenience of the users and to be a valid alternative to the other

e-mobility applications available on Google Play Store or Apple Store. A web application offering the same functionalities of the mobile app could be implemented in the future, in order to provide eMall users more possibilities to access its functionalities.
The part regarding eMall operators and their communication with other CPMSs instead relies on a desktop application installed only in company computers and has to be used and exploited exclusively from the operators themselves, with administrator operations and interfaces.

4 Formal analysis using Alloy

```
------------------------------SIGNATURES-------------------------------

sig UserId{}

sig User  {
        id: disj one UserId,
        vehicle: one Vehicle, // For simplicity, in the Alloy model
            ↪ the User isn't allowed to register more than one
            ↪ vehicle
        schedule: set Appointment
}

sig Vehicle {
        batteryState: one BatteryState,
        absorbedPower: one Int,
        chargeSecondsLeft: one Int
} {
        chargeSecondsLeft ≥ 0
        absorbedPower ≥ 0
}

abstract sig BatteryState {}
lone sig NEEDS_CHARGING extends BatteryState {}
lone sig CHARGING extends BatteryState {}
lone sig CHARGED extends BatteryState {}

sig Appointment {
        startingTime: one Timestamp,
        endingTime: one Timestamp,
        location: one Location
} {
        startingTime.value < endingTime.value
}

sig Timestamp {
        // This value represents the "epoch time", i.e. the number of
            ↪ seconds since 1st January 1970, as done in practice by
            ↪ many systems
        value: one Int
} {
        value > 0
}

sig Location {}

sig CPO {
        stations: set ChargingStation
}

sig DSO {
        proposedPrice: one EnergyPrice
}
```

```
abstract sig EnergyPrice {
        // The declaration of an Int field "price", although natural,
            ↪ is omitted because not relevant for the model
}
sig STANDARD extends EnergyPrice {}
sig DISCOUNT extends EnergyPrice {}

sig EnergyPurchase {
        cpo: one CPO,
        dso: one DSO
}

-- How is the logical structure of a ChargingStation organized?
-- Every station contains a set of ChargingSocketsGroup, which,
-- as the name itself says, represent a set of sockets of the same
-- ChargingSocketType (i.e. fast/rapid/slow charging mode).
-- Every group is associated to his current EnergyPrice, which
-- of course does not depend on the specific socket within the group.
-- Finally, every ChargingSocket can have - or not - an attached
-- Vehicle, which obviously has to be in charging state
sig ChargingStation {
        location: one Location,
        chargingSocketsGroups: some ChargingSocketsGroup
}

sig ChargingSocketsGroup {
        sockets: set ChargingSocket,
        currentEnergyPrice: one EnergyPrice,
        secondsUntilFree: one Int
} {
        secondsUntilFree ≥ 0
}

sig ChargingSocket {
        type: one ChargingSocketType,
        attachedVehicle: lone Vehicle
}

abstract sig ChargingSocketType {
        // We assume that the maximum power erogated by each socket
            ↪ type is standardized and doesn't depend on the specific
            ↪  charging station
        maxErogatedPower: one Int
} {
        maxErogatedPower > 0
}
lone sig SLOW extends ChargingSocketType {}
lone sig RAPID extends ChargingSocketType {}
lone sig FAST extends ChargingSocketType {}

abstract sig ChargingAction {
        // We assume that a user can book a specific ChargingSocket
        socket: one ChargingSocket,
        user: one User,
        validFrom: one Timestamp,
        validUntil: one Timestamp
```

```
} {
        validFrom.value < validUntil.value
}
sig Suggestion extends ChargingAction {}
sig Reservation extends ChargingAction {}

--------------------------------FACTS---------------------------------

-- Facts related to power

fact erogatedPowerConstraint {
        // The maximum erogated power for fast charge is grater than
            ↪ the one for rapid charge, which is greater than the one
            ↪  for slow charge
        FAST.maxErogatedPower > RAPID.maxErogatedPower and RAPID.
            ↪ maxErogatedPower > SLOW.maxErogatedPower
}

fact onlyChargingVehiclesAbsorbePower {
        // Only the vehicles that are actually charging can absorbe
            ↪ power
        all v: Vehicle | (v.absorbedPower = 0) iff not (v.batteryState
            ↪  = CHARGING)
}

fact maxErogatedPowerSufficient {
        // The amount of power absorbed by a vehicle cannot exceed the
            ↪  power erogated by the socket it's connected to
        all group: ChargingSocketsGroup | (all sock: ChargingSocket |
                (sock in group.sockets) implies (sock.attachedVehicle.
                    ↪ absorbedPower ≤ sock.type.maxErogatedPower))
                    ↪ and
                (some sock: ChargingSocket | (sock in group.sockets)
                    ↪ implies (sock.attachedVehicle.absorbedPower =
                    ↪ sock.type.maxErogatedPower))
}


-- Fact related to seconds left until one socket of a certain type is
    ↪ free

fact timeUntilOneGroupSocketIsFreeIsCoherent {
        // If there's a free socket then the attribute
            ↪ secondsUntilFreed = 0, otherwise "secondsUntilFree" is
            ↪ the minimum among all "secondsLeft" attributes of the
            ↪ charging vehicles
        all group: ChargingSocketsGroup | (group.secondsUntilFree = 0
            ↪ iff (
                some socket: ChargingSocket | (socket in group.sockets
                    ↪  and no v: Vehicle | (v = socket.
                    ↪ attachedVehicle) )))
        and
        all group: ChargingSocketsGroup, socket: ChargingSocket, v:
            ↪ Vehicle |
                (socket in group.sockets and v = socket.
                    ↪ attachedVehicle) implies group.secondsUntilFree
```

```
                    ↪  ≤ v.chargeSecondsLeft
        and
        some group: ChargingSocketsGroup, socket: ChargingSocket, v:
            ↪ Vehicle |
                socket in group.sockets and v = socket.attachedVehicle
                    ↪  and group.secondsUntilFree = v.
                    ↪ chargeSecondsLeft
}


-- Fact related to coherent existence of entities

fact noChargingGroupWithoutStation {
        // A ChargingSocketsGroup cannot exist if not associated to a
            ↪ ChargingStation
        all g: ChargingSocketsGroup | (one s: ChargingStation | g in s
            ↪ .chargingSocketsGroups)
}

fact noSocketWithoutChargingGroup {
        // A ChargingSocket cannot exist if not associated to a
            ↪ ChargingSocketsGroup
        all sock: ChargingSocket | (one group: ChargingSocketsGroup |
            ↪ sock in group.sockets)
}

fact noVehicleWithoutUser {
        // A Vehicle cannot exist if not associated to a User
        all v: Vehicle | (one u: User | v = u.vehicle)
}

fact noAppointmentWithoutUser {
        // An Appointment cannot exist if not associated to a User
        all a: Appointment | (one u: User | a in u.schedule)
}

fact noBatteryStateWithoutVehicle {
        // A BatteryState cannot exist if not associated to a Vehicle
        all state: BatteryState | (some v: Vehicle | state = v.
            ↪ batteryState)
}

fact noEnergyPriceWithoutGroup {
        // A EnergyPrice cannot exist if not associated to a
            ↪ ChargingSocketsGroup
        all e: EnergyPrice | (one group: ChargingSocketsGroup | e =
            ↪ group.currentEnergyPrice)
}

fact noChargingStationWithoutCPO {
        // A ChargingStation cannot exist if not associated to a CPO
        all s: ChargingStation | (one cpo: CPO | s  in cpo.stations)
}

fact noDSOWithoutEnergyPurchase {
```

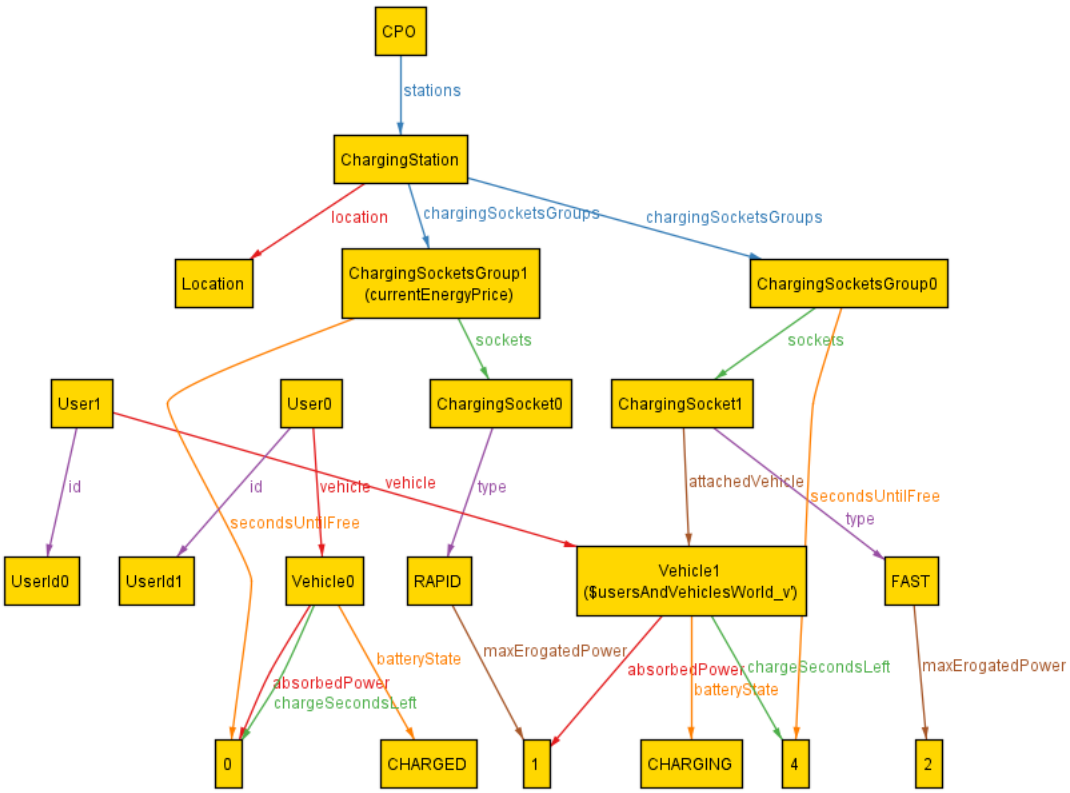
```
        // A DSO entity cannot exist if not associated to an
            ↪ EnergyPurchase
        all d: DSO | (one purchase: EnergyPurchase | d = purchase.dso)
}


-- Facts related to ChargingSocketGroups

fact allSocketsInGroupHaveSameType {
        // All the sockets belonging to the same group have the same
            ↪ charging type (fast/rapid/slow)
        all group: ChargingSocketsGroup, socket1, socket2:
            ↪ ChargingSocket |
                (socket1 in group.sockets and socket2 in group.sockets
                    ↪ ) implies socket1.type = socket2.type
}

fact noTwoGroupsWithSameChargingType {
        // There cannot exists two charging groups belonging to the
            ↪ same station and providing the same charging type
        no disjoint g1, g2: ChargingSocketsGroup | (one station:
            ↪ ChargingStation | g1 in station.chargingSocketsGroups
            ↪ and g2 in station.chargingSocketsGroups) and
                (some disjoint s1, s2: ChargingSocket | (s1 in g1.
                    ↪ sockets and s2 in g2.sockets and s1.type = s2.
                    ↪ type) )
}

fact noEmptyGroups {
        // Every charging group can exist only if it contains at least
            ↪  a socket
        all g: ChargingSocketsGroup | (some s: ChargingSocket | s in g
            ↪ .sockets)
}


-- Facts related to Vehicle entities

fact chargingVehicleHasCorrectType {
        // If a vehicle is being charged, its BatteryState must be
            ↪ CHARGING or CHARGED - it could happen that the recharge
            ↪  is finished but the vehicle is still attached
        all socket: ChargingSocket | (socket.attachedVehicle ≠ none
            ↪ implies socket.attachedVehicle.batteryState = CHARGING)
}

fact onlyChargingVehiclesHaveSecondsLeft {
        // If a vehicle is not charging or is fully charged it does
            ↪ not absorbe power from the socket
        all v: Vehicle | (v.chargeSecondsLeft = 0) iff not (v.
            ↪ batteryState = CHARGING)
}

fact vehicleMustBeChargingOnASocket {
        // If a vehicle is in charging state it must be connected to a
            ↪  socket
```

```
        all v: Vehicle | v.batteryState = CHARGING implies (
                one s: ChargingSocket | s.attachedVehicle = v)
}


-- Facts related to ChargingAction entities - i.e. Reservation and
    ↪ Suggestion entities

fact chargingActionPresentOnlyIfVehicleNeedsCharging {
        // Every charging action - a Reservation or a Suggestion - is
            ↪ related to a vehicle that needs charging
        all a: ChargingAction | a.user.vehicle.batteryState =
            ↪ NEEDS_CHARGING
}

fact chargingActionOnlyIfSocketFree {
        // The Reservation or the Suggestion can be showed only if the
            ↪  proposed ChargingSocket is currently free
        all a: ChargingAction | (a.socket.attachedVehicle = none)
}

fact noSuggestionsIfSocketBooked {
        // The Suggestion can be showed only if the proposed
            ↪ ChargingSocket is not already booked
        no s: Suggestion, r: Reservation | (r.socket = s.socket and (r
            ↪ .validFrom.value ≤ s.validUntil.value))
}

fact noReservationsIfSocketBooked {
        // The Reservation can be showed only if the proposed
            ↪ ChargingSocket is not already booked
        no disjoint r1, r2: Reservation | (r1.socket = r2.socket)
}

fact noFurtherReservationsForSameUser {
        // A user can't book a socket in a time interval during which
            ↪ he has another active reservation
        no disjoint r1, r2: Reservation | (r1.user = r2.user)
}

fact noDuplicateSuggestions {
        // There can't be two suggestions for the same user and socket
            ↪  in a colliding time interval
        no disjoint s1, s2: Suggestion | (s1.socket = s2.socket and s1
            ↪ .user = s2.user and
                (s1.validFrom.value = s2.validFrom.value or s1.
                    ↪ validUntil.value = s2.validUntil.value))
}

fact noSuggestionIfUserHasReservation {
        // No suggestions are showed to users which already have a
            ↪ reservation
        no r: Reservation, s: Suggestion | r.user = s.user
}

fact suggestionIsCoherentWithUserAppointment {
```

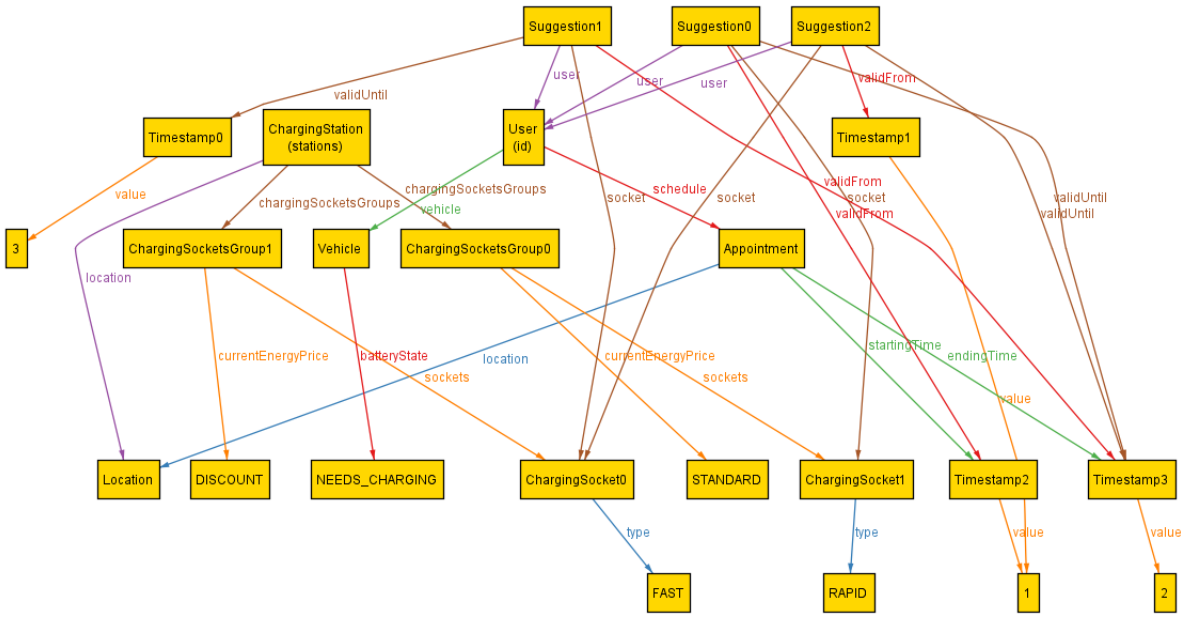
```
        // Each suggestion is based on an appointment of the user or
            ↪ on a price reduction
        all sugg: Suggestion | (some a: Appointment | a in sugg.user.
            ↪ schedule and sugg.validFrom.value ≤ a.startingTime.
            ↪ value and
                (one station: ChargingStation, group:
                    ↪ ChargingSocketsGroup | group in station.
                    ↪ chargingSocketsGroups and sugg.socket in group.
                    ↪ sockets and
                station.location = a.location) )
                or
                (one group: ChargingSocketsGroup | sugg.socket in
                    ↪ group.sockets and
                        group.currentEnergyPrice = DISCOUNT)
}

fact cpoAlwaysPurchasesEnergyIfDiscounted {
    // We assume that if there's a DSO that sells energy at discount
        ↪ price there's at least a CPO that buys it. This fact
        ↪ doesn't necessarily hold in a complex real-world system
        ↪ where the purchase of energy depends on many different
        ↪ factors, both internal and external to the CPO, but is
        ↪ reasonable for our simplified model
    all d: DSO | d.proposedPrice = DISCOUNT implies (some purchase:
        ↪ EnergyPurchase | purchase.dso = d)
}
```

```
------------------------------ASSERTIONS--------------------------------


assert correctNumberOfChargingGroups {
        // Every station can't have at maximum one group for every
            ↪ charging type
        all s: ChargingStation | #s.chargingSocketsGroups ≤ #
            ↪ ChargingSocketType
}

check correctNumberOfChargingGroups for 5

assert vehicleIsCharging {
        // Assert that when a vehicle is in charging state all the
            ↪ related attributes have to be coherent
        all v: Vehicle | v.batteryState = CHARGING iff (
                v.chargeSecondsLeft > 0 and v.absorbedPower > 0 and
                    ↪ one s: ChargingSocket | (
                        s.attachedVehicle = v and s.type.
                            ↪ maxErogatedPower ≥ v.absorbedPower))
}

check vehicleIsCharging for 5

assert grantReservationAndSuggestionsToNonChargingVehicles {
    // Assert that all the reservations and the suggestions involves
        ↪ vehicles that are not currently charging
        no v: Vehicle | v.batteryState = CHARGING and (some action:
            ↪ ChargingAction | action.user.vehicle = v)
}

check grantReservationAndSuggestionsToNonChargingVehicles for 5

assert giveSuggestionsBasedOnPriceOrLocation {
        // There are no suggestions for a non-discount price in a
            ↪ Location where the user doesn't have an appointment in
        no s: Suggestion | (one group: ChargingSocketsGroup | s.socket
            ↪  in group.sockets and group.currentEnergyPrice =
            ↪ STANDARD and
                (no a: Appointment | a in s.user.schedule and (
                        one station: ChargingStation |  group in
                            ↪ station.chargingSocketsGroups and
                            ↪ station.location = a.location)))
}

check giveSuggestionsBasedOnPriceOrLocation for 5
```

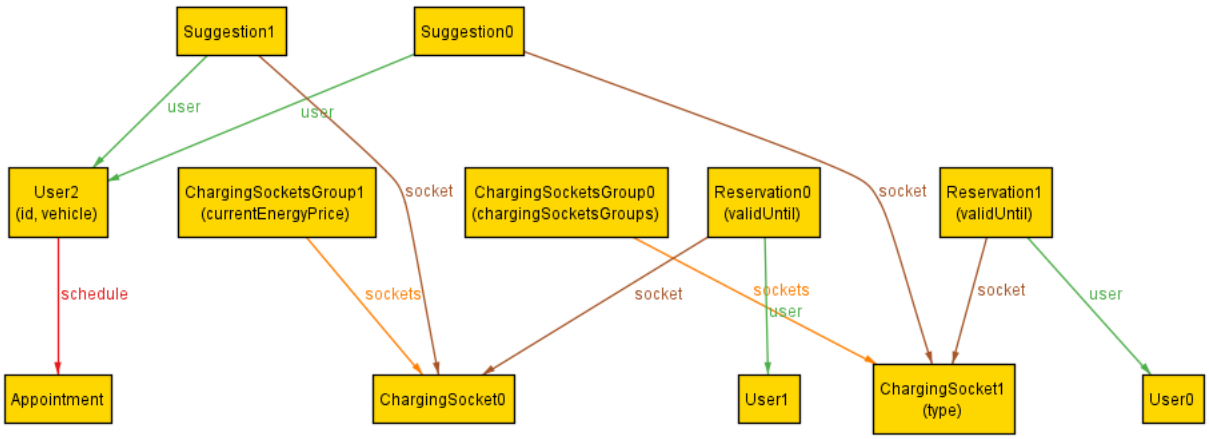
```
-------------------------------PREDICATES--------------------------------

-- This generated instance reported in the picture shows a CPO with
-- one station which has two sockets, one with fast charge and the
-- other one with rapid charge. The former (ChargingSocket1)
-- is charging Vehicle1, owned by User1. Vehicle0 doesn't need
-- to be attached to a socket because its battery is charged.
-- We can observe that the number of seconds left until
-- the vehicle is charged is equal to the number of seconds until
-- a fast-charging socket is free (4) and that the power absorbed
-- by the vehicle (1) is less than the maximum power
-- available in the socket (2), as expected
pred usersAndVehiclesWorld {
        #Vehicle ≥ 2
        #CPO = 1
        #ChargingSocketsGroup ≥ 2
        some v: Vehicle | v.batteryState = CHARGING
}

run usersAndVehiclesWorld for 4
```



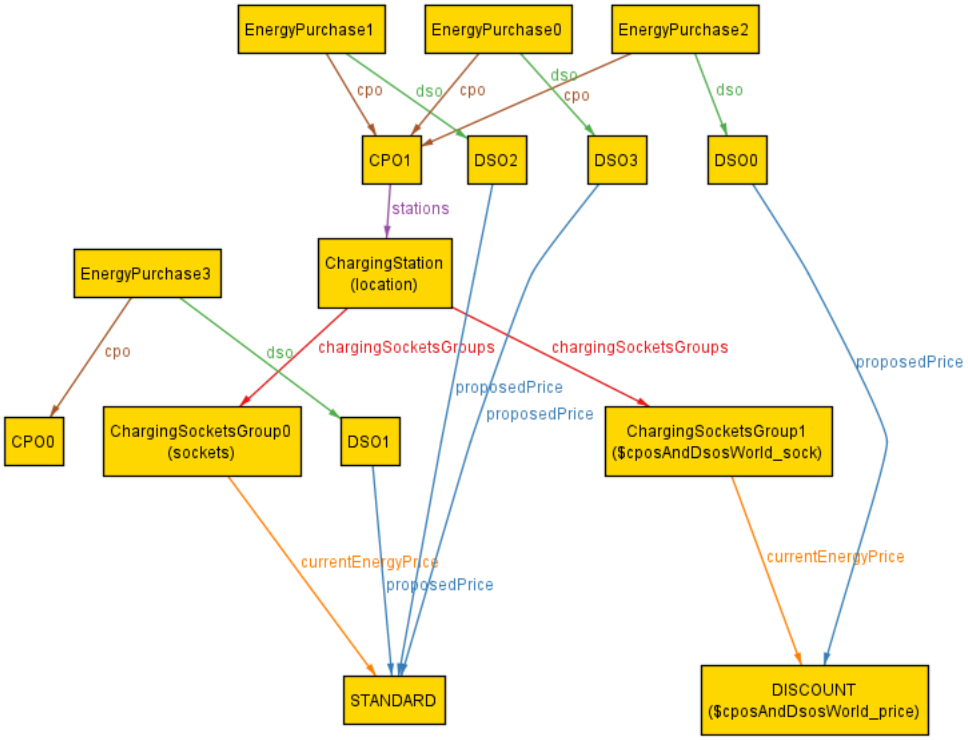
```
-- The instance reported in the picture hides many relations
-- in order to focus only on the aspects described below
-- All the three suggestions are for the same user. Suggestion2
-- is based on the entity Appointment associated to the user:
-- in fact, the proposed socket (ChargingSocket1) belongs
-- to a station which is in the same location as the one
-- the user saved in their schedule, and both the schedule
-- and the suggestion are in the time interval 1-2.
-- Suggestion1 and Suggestion2 are two suggestions on
-- the same socket (ChargingSocket0) but regarding different
-- time intervals. They are valid suggestions because
-- the charging group the socket belongs to (ChargingSocketsGroup1)
-- offers a discount on the energy price
pred suggestionsWorld {
        #Suggestion = 3
        #Appointment = 1
        #User = 1
        #ChargingSocket < #Suggestion
        #Reservation = 0
        one c: ChargingSocketsGroup | c.currentEnergyPrice = DISCOUNT
        one s: Suggestion | (one c: ChargingSocketsGroup | c.
            ↪ currentEnergyPrice = STANDARD and s.socket in c.sockets
            ↪  )
}

run suggestionsWorld for 4
```



```
-- Simulation that shows how reservations and suggestions
-- can combine together
-- The instance is projected over many sigs and relations
-- (including all temporal ones), in order to focus more on
-- the actual interactions between the entities
-- As expected, the users which have an active reservation
-- (i.e. User0 and User1) don't receive suggestions, while
-- User2, which has no reservations, can receive more
-- than one suggestion.
-- There are a reservation and a suggestion for each
-- charging socket, but the two don't collide because they
-- consider different sockets
pred reservationsAndSuggestionsWorld  {
        #User = 3
        #Suggestion > 1
        #Reservation > 1
        #ChargingSocket ≤ 2
        all u: User | (some c: ChargingAction | c.user = u)
}

run reservationsAndSuggestionsWorld for 4
```



```
-- This simulation shows how CPOs interact with DSOs to purchase
-- energy. In the shown image, CPO1 buys energy from DSO0 and DSO1,
-- both offering it with a discounted price, while CPO0 purchases
-- energy from DSO2 and DSO3 at the standard cost.
-- Notice that the price at which CPOs acquire energy is not
-- necessarily related to the price at which they sell it, as expected

pred cposAndDsosWorld {
    #CPO ≥ 2
    #DSO ≥ 3
    #EnergyPrice = 2
    some e: EnergyPrice | e = DISCOUNT
}

run cposAndDsosWorld for 4
```



5 Effort spent

Student	Section 1	Section 2	Section 3	Section 4
Claudio Arione	2h	7h	20h	12h
Riccardo Begliomini	5h	12h	20h	2h
Niccolò Bindi	5h	9h	16h	8h

6 References

- https://www.europarl.europa.eu/news/en/headlines/society/20190313STO31218/co2-emissions-from-cars-facts-and-figures-infographics
- https://en.wikipedia.org/wiki/Open_Charge_Point_Protocol
- https://developer.ibm.com/articles/the-sequence-diagram/
- https://www.uml-diagrams.org
- https://www.bpmn.org