DD
Design Document

Software Engineering 2 - Matteo Rossi

Niccolò Nicolosi, Francesco Negri, Marcos Pietrucci

08/01/2023

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to describe in details the design of e-Mobility for all (eMall), the system to be. The document contains the specification of the system architecture in details, including the most relevant examples on the run-time behavior of the system and the way it will be deployed on hardware, UI mockups to describe the idea of the final look of the application and plans on how to integrate the different components of the system, optimizing developing and testing time. The objective of this document is to support the developers of eMall throughout the implementation of the system.

1.2 Scope

The eMall system has two types of users: end-users and CPOs.

The end-user has access to the following main functionalities:

- view and select charging stations and start a charge from the stations page
- view the details of a charging station, book a charge and ask for directions to reach the station from the details page
- manage their vehicles from the vehicles page
- view the charges and payments history and active reservations and cancel reservations from his personal page
- link his calendar cloud service from the settings page

The CPO has access the following main functionalities:

- view and select proprietary charging stations from the stations page
- view the details of proprietary charging stations and manage them from the details section in the stations page
- view the available and active DSOs from the DSO page
- view the details of DSOs from the details section in the DSO page

1.3 Abbreviations and definitions

- eMall: e-Mobility for all (the system to be)
- eMSPs: eMall Service Providers
- eMSS: eMall Service System (subsystem 1)

- end-users: eMSS users
- CPMS: Charging Point Management System (subsystem 2)
- CPOs: Charging Point Operators (CPMS users)
- DSOs: Distribution System Operators
- users: generic users of the system (both end-users and CPOs)
- external status of a charging station:
  - types of charging sockets offered (slow/fast/rapid) and their cost
  - number of charging sockets available and their type
  - estimated amount of time until the first socket of a specific type is freed
  - special offers
- internal status of a charging station:
  - current DSO supplying it
  - amount of energy available in its batteries (if any)
  - number of vehicles being charged
  - amount of power absorbed by each charge
  - time left to the end of each charge

1.4 Reference documents

- eMall RASD document
- Project assignment specification document

2 Architectural Design

2.1 Architectural styles and patterns

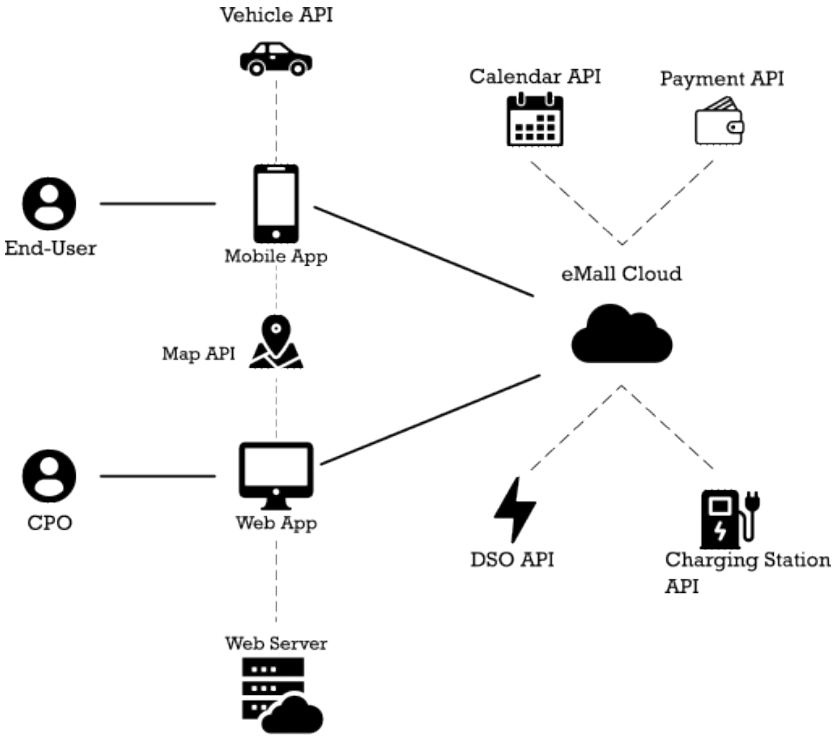
The system will be developed following the architectural styles and patterns proposed below:

- **Microservices architecture**: with this architecture the system is divided in many microservices hosted on the cloud that communicate with each other, with external systems and with clients, to provide the final service. This architecture gives the possibility to adopt the following architectural patterns that improve the scalability, the availability, the maintainability, the elasticity and the fault tolerance of the system.

- **Composite application**: the system divides its functionalities in different components that can be developed and maintained almost independently. These components can be deployed on different machines to improve scalability and can be replicated to improve scalability and availability.
- **Stateless components**: the components should be stateless by themselves and rely on a database to save their state, so that, if they fail, they can recover their computation simply reading from their database.
- **Watchdog**: the component instances are monitored by a dedicated watchdog component that can detect failures and start new instances to replace failing ones.
- **Loose coupling**: the server components interact with one another through brokers that redirect requests to available components, in order to loose coupling and improve scalability, availability, maintainability and fault tolerance.
- **Elastic load balancer**: the requests between components are managed by a load balancer component that can decide the required number of instances to run for each component. This improves scalability and elasticity.
- **Update transition**: the load balancer that addresses requests is also in charge of seamlessly transitioning requests from older versions of the components to newer versions and of decommissioning old versions when no more used.

2.2 Overview

Below we present an overview of the system:

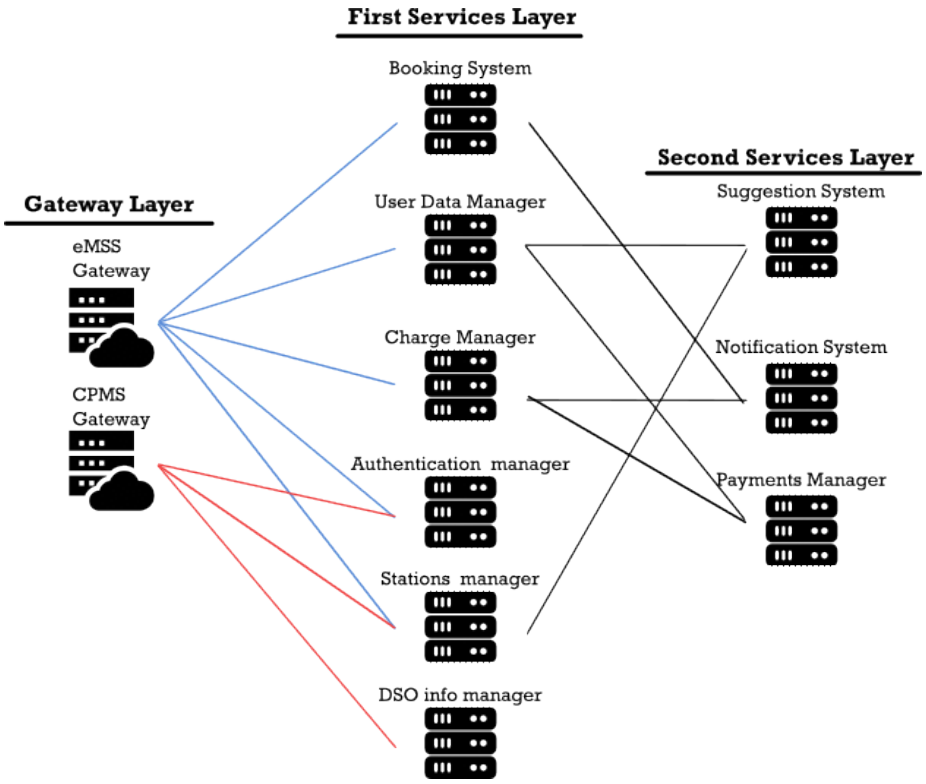


The elements in the diagram should be considered as follows:

- Mobile app: the eMall app that must be downloaded and installed in the end-users' smartphone. It is the only way an end-user can register and interact with the system
- Web app: the web app specifically designed for the CPOs to interact with the system
- Web server: the server that stores the resources of the web app (HTML pages, JavaScript code, images, etc...) to be downloaded from web browsers
- Vehicle API: this API allows our application to communicate with the end-user's vehicle navigation system in order to know its battery level
- Map API: this API is used to obtain the map that will be displayed both in the mobile app and in the web app
- eMall cloud: the cloud services offered by the system
- Calendar API: this API is used to read the personal calendar of end-users, needed to provide automatic suggestions
- Payment API: this API is used to verify the end-users' payment method and to charge them payments

- Charging station API: this API is used to get information about a specific station connected to the system and to modify its settings
- DSO API: this API is used to get information about a specific DSO connected to the system

The cloud services offered by the system are the following:

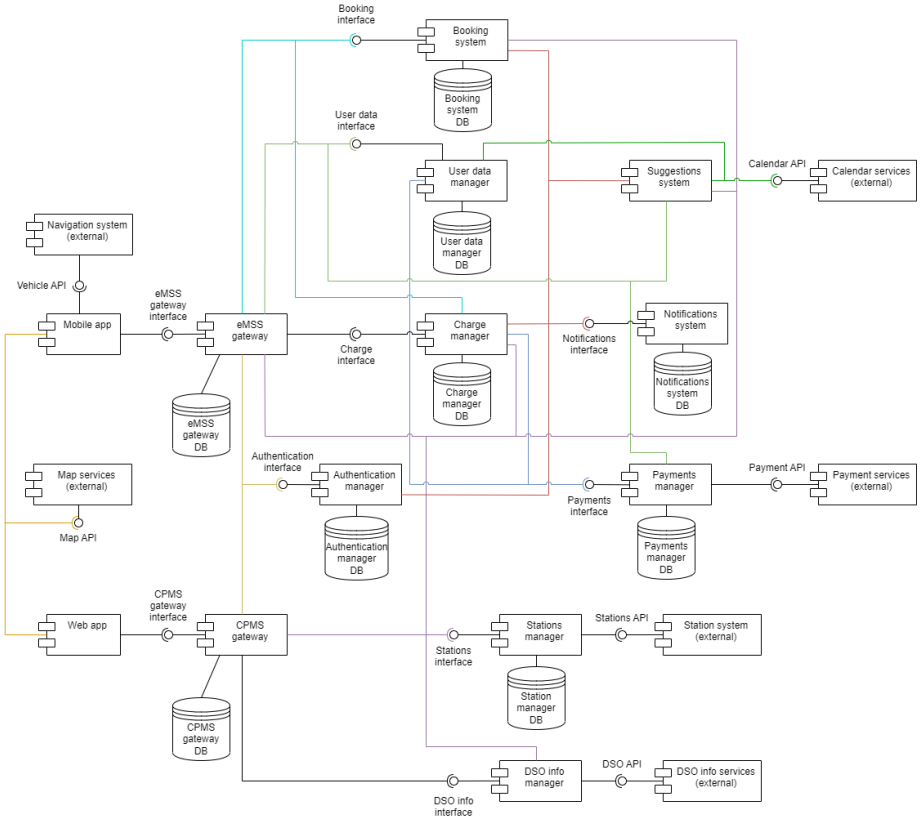


Given the structure of the microservices and considering that each microservice has its own database, used to store and access data relative to its purpose, the architecture of the system as a whole can be viewed as a five tier architecture divided in:

1. Presentation tier
2. Gateway tier
3. First services tier
4. Second services tier
5. Data tier

2.3 Component view

The following diagram describes the high-level architecture of the system, showing the core components and the main dependencies.



2.3.1 External components

The components marked as *external* in the diagram above are already existing components the system interacts with through the APIs they provide:

- **Navigation system**: the navigation system of a vehicle. It provides information about the vehicle's battery to the system
- **Map services**: services that provide the system with a navigable and interactive map and the possibility to represent points of interest at specific locations in the map
- **Calendar services**: services that provide the system with read access to end-users' calendars
- **Payment services**: services that provide the system with the possibility to verify payment methods and charge payments to end-users

- **Station system**: the internal system of a charging station. It provides information about the station and the possibility to manage it through specific settings
- **DSO info services**: services that provide information about a specific DSO to the system

2.3.2 Mobile app

This component represents the mobile app that will be installed on end-users' smartphones. The app will provide a graphical user interface dedicated to the end-users for them to interact with the eMSS.

2.3.3 Web app

This component represents the web app which CPOs can access through their web browser. The app will provide a graphical user interface dedicated to the CPOs for them to interact with the CPMS.

2.3.4 eMSS gateway

This component is the access point to the eMSS, used by the mobile app for every request to the system. The module stores the tuples [authenticatedIP, userID] relative to end-users in its database, each for the time the corresponding end-user is authenticated, and uses them to make requests to other components.

2.3.5 CPMS gateway

This component is the access point to the CPMS, used by the web app for every request to the system. The module stores the tuples [authenticatedIP, userID] relative to CPOs in its database, each for the time the corresponding CPO is authenticated, and uses them to make requests to other components.

2.3.6 Authentication manager

This component is responsible of authenticating users with email and password, in order to grant them access to the application. It also requests a verification code which is sent to the user by email if the authentication device is not associated with the authenticating user in the system.

2.3.7 User data manager

This component provides the mobile app with an interface to read or modify data relative to the end-user, like his vehicles, payment method and calendar account. The interface is also used by other modules to access user data. The module is also responsible of registering and unregistering end-users.

2.3.8 Booking system

This component is responsible of adding and removing reservations, providing the mobile app with the available time frames and socket types to choose from. The module also adds a reminder notification for every reservation, interacting with the notification system.

2.3.9 Charge manager

This component is responsible of starting new charges by interacting with the stations manager and with the booking system, of maintaining the users' charge history, of sending a notification whenever a charge ends by interacting with the notification system and of starting the payment process interacting with the payments manager.

2.3.10 Notifications system

This component is responsible of creating notifications (both push notifications and emails), storing them in its database until needed and sending them to end-users when scheduled.

2.3.11 Payments manager

This component is responsible of charging payments to end-users, communicating with external payment services and ensuring they succeed at some point. The module is also responsible of verifying if the payment methods entered by end-users are valid and of managing their payments history.

2.3.12 Suggestions system

This component is responsible of periodically analyzing end-users' data and calendar to suggest them to charge their vehicles in a convenient station when low on battery. The module interacts with the user data manager to analyze the user data, with the stations manager to analyze stations and with the notification system to send the suggestion notifications.

2.3.13 Stations manager

This component is responsible of providing stations data and current charges data to clients and other modules. The module saves static and weakly dynamic data (location, prices, offers, provided socket types, etc...) in its database for fast access and retrieves highly dynamic data (current available sockets, estimated time until specific socket types are freed, etc...) communicating with the station systems. The module also provides an interface to manage the stations through specific settings.

2.3.14 DSO info manager

This component is responsible of providing DSO data to the web app, retrieving them from specific DSOs' services (energy price and sources) and from the stations manager (station-DSO associations).

2.4 Component interfaces

In this section we illustrate the specific interfaces offered by the components listed above.

NOTE: user is a data structure containing both the userID associated to the user (end-user or CPO) and the IP to use for communications.

- eMSS gateway interface
  - authenticate(email, password)
  - verifyDevice(verificationCode)
  - register(registrationData)
  - unregister()
  - getVehicles()
  - addVehicle(vehicle)
  - removeVehicle(vehicleID)
  - getCalendar()
  - addCalendar(calendar)
  - removeCalendar()
  - getPaymentMethod()
  - setPaymentMethod(paymentMethod)
  - getAvailabilities(stationID, date)
  - book(availability)
  - getReservations()
  - removeReservation(reservationID)
  - startCharge(stationID, socketID)
  - getChargesHistory()
  - getRealTimeChargeDetails(chargeID)
  - stopRealTimeChargeDetails(chargeID)
  - getStationsInRange(position, range)
  - getStationDetails(stationID)
- CPMS gateway interface

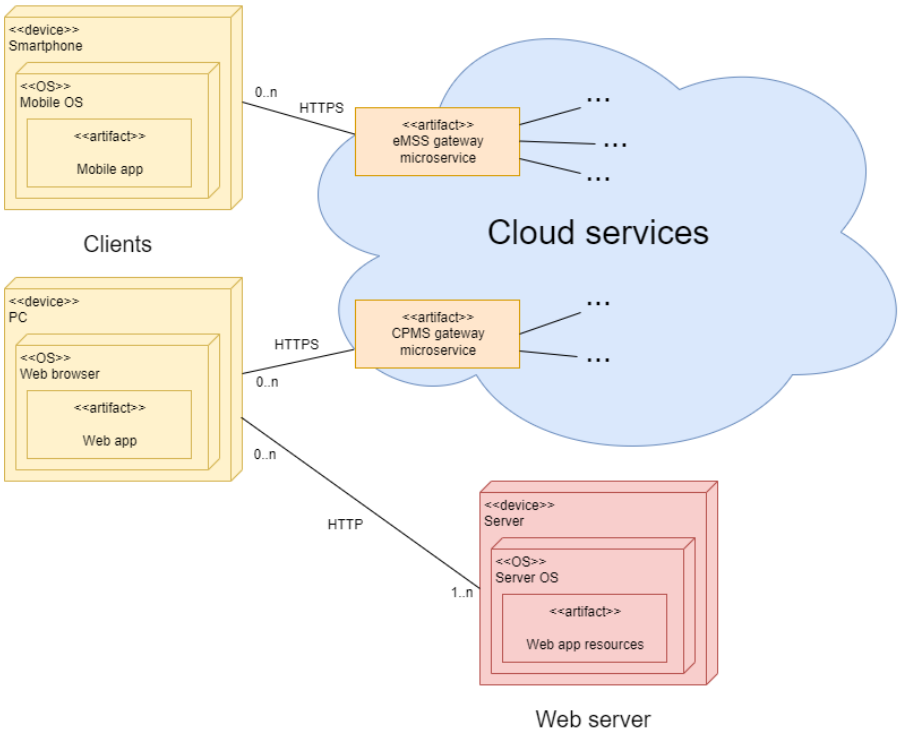
  - authenticate(email, password)
  - verifyDevice(verificationCode)
  - getStationsInRange(position, range)
  - getExternalDetails(stationIDs)
  - getInternalDetails(stationIDs)
  - setPrice(stationIDs, socketType, price)
  - setOffer(stationIDs, socketType, price, start, end)
  - setDSO(stationIDs, DSO_ID)
  - setEnergyMix(stationIDs, batteryUsagePercent)
  - chargeBatteries(stationIDs)
  - setAuto(stationIDs, profits, chargeThreshold)
  - getDSOdetails(DSO_IDs)
- Authentication interface
  - authenticate(userIP, email, password)
  - verifyDevice(user, verificationCode)
- User data interface
  - register(registrationData)
  - unregister(user)
  - getUserPosition(user)
  - getVehicles(user)
  - sendVehicles(user)
  - addVehicle(user, vehicle)
  - removeVehicle(user, vehicleID)
  - getCalendar(user)
  - sendCalendar(user)
  - addCalendar(user, calendar)
  - removeCalendar(user)
  - getPaymentMethod(user)
  - sendPaymentMethod(user)
  - setPaymentMethod(user, paymentMethod)
- Booking interface
  - sendAvailabilities(user, stationID, date)
  - book(user, availability)

  - sendReservations(user)
  - removeReservation(user, reservationID)
  - removeAllReservations(user)
  - checkValid(user, stationID, socketID)
- Charge interface
  - startCharge(user, stationID, socketID)
  - sendChargesHistory(user)
- Notifications interface
  - sendNotification(notification)
  - scheduleNotification(notification, datetime)
- Payments interface
  - pay(user, price)
  - verify(paymentMethod)
- Stations interface
  - getStationsInRange(position, range)
  - sendStationsInRange(user, position, range)
  - getExternalDetails(stationIDs)
  - sendExternalDetails(user, stationIDs)
  - sendAllDetails(user, stationIDs)
  - setPrice(user, stationIDs, socketType, price)
  - setOffer(user, stationIDs, socketType, price, start, end)
  - setDSO(user, stationIDs, DSO_ID)
  - setEnergyMix(user, stationIDs, batteryUsagePercent)
  - chargeBatteries(user, stationIDs)
  - setAuto(user, stationIDs, profits, chargeThreshold)
  - startCharge(chargeID, stationID, socketID)
  - sendRealTimeChargeDetails(user, chargeID)
  - stopRealTimeChargeDetails(user, chargeID)
- DSO info interface
  - sendDSOdetails(user, DSO_IDs)

2.5 Deployment view

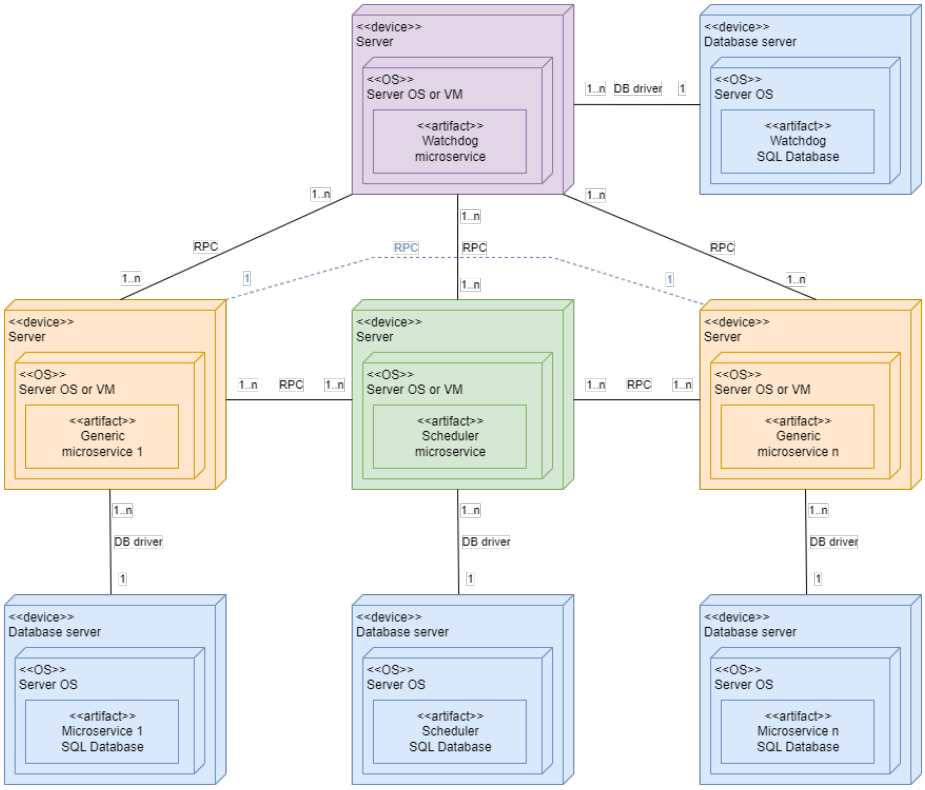
In this section we present how the system is intended to be deployed.

2.5.1 Overview



As shown in the diagram, the mobile clients will communicate with the cloud services through the gateway microservice using the HTTPS protocol.
Instead, the web clients will have to communicate with the web server first, in order to fetch the application resources like HTML pages, JavaScript code, images, etc... Using those resources, they will then be able to communicate with the gateway microservice using the HTTPS protocol.

2.5.2 Detailed cloud services view



Looking at the details of the cloud services, the above diagram shows how each microservice runs on a separate (virtual) OS, so that it can be assigned computational resources dynamically. Furthermore, each microservice can be replicated an arbitrary number of times for the purposes mentioned in the section Architectural styles and patterns, but all the replicas refer to a unique database dedicated to the specific microservice.

Additionally, generic microservices should be distinguished from two specific microservices that serve no functional purpose:

- **The scheduler microservice**:
  - Each generic microservice makes requests to others by first asking the scheduler which replica to communicate with. The scheduler replies with the address to the available replica of the requested microservice with less computational load, then the two microservices can communicate to satisfy the request

  - Analyzes the request traffic to dynamically instantiate new replicas of specific microservices or decommission excess replicas. It can also replicate itself to redistribute the traffic if needed
  - Seamlessly transitions requests from older versions of the components to newer versions when released and decommissions old versions when no more used

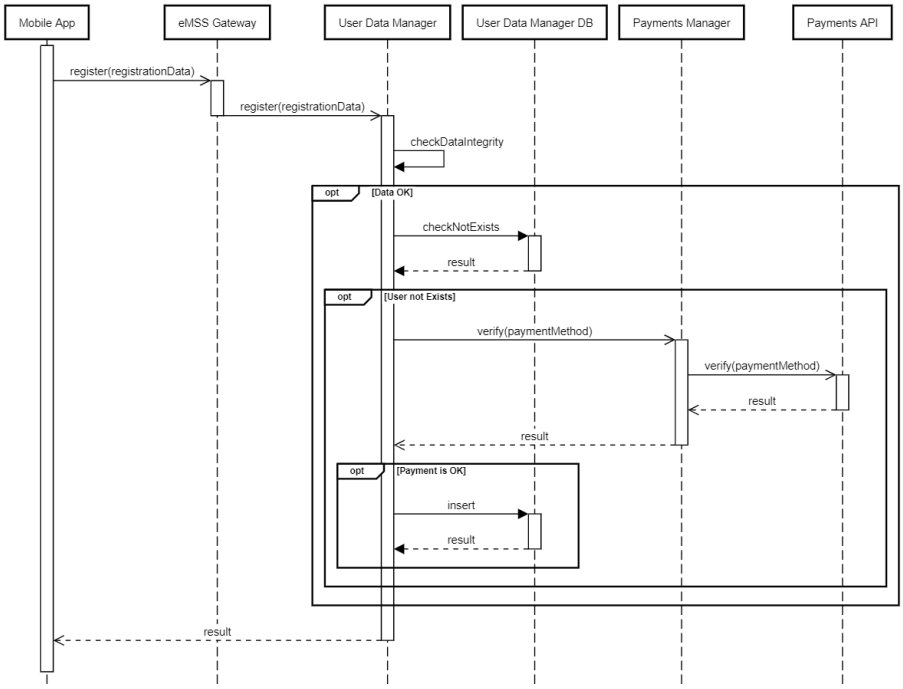
- **The watchdog microservice**: periodically pings each replica of each microservice to detect failing ones and restarts them. If replicated, it shares its workload between replicas.

All the communications between microservices use the Remote Procedure Call protocol, to hide the complexity of communication under a dedicated middleware and ease the development process.

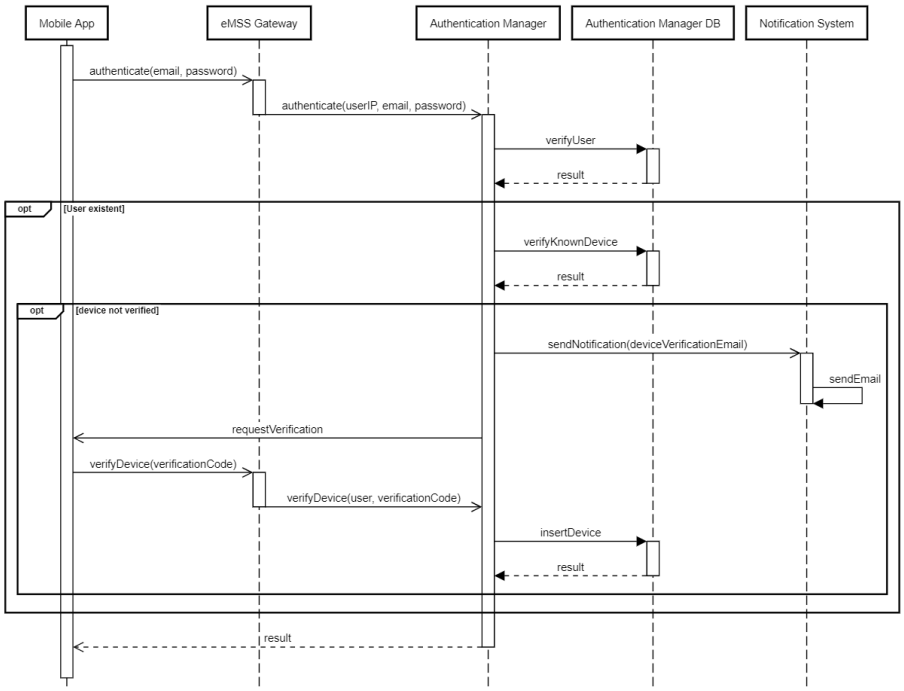
2.6 Runtime View

In this section we will show in details the operations and interactions between the components of our system. These operations will be represented with sequence diagrams.

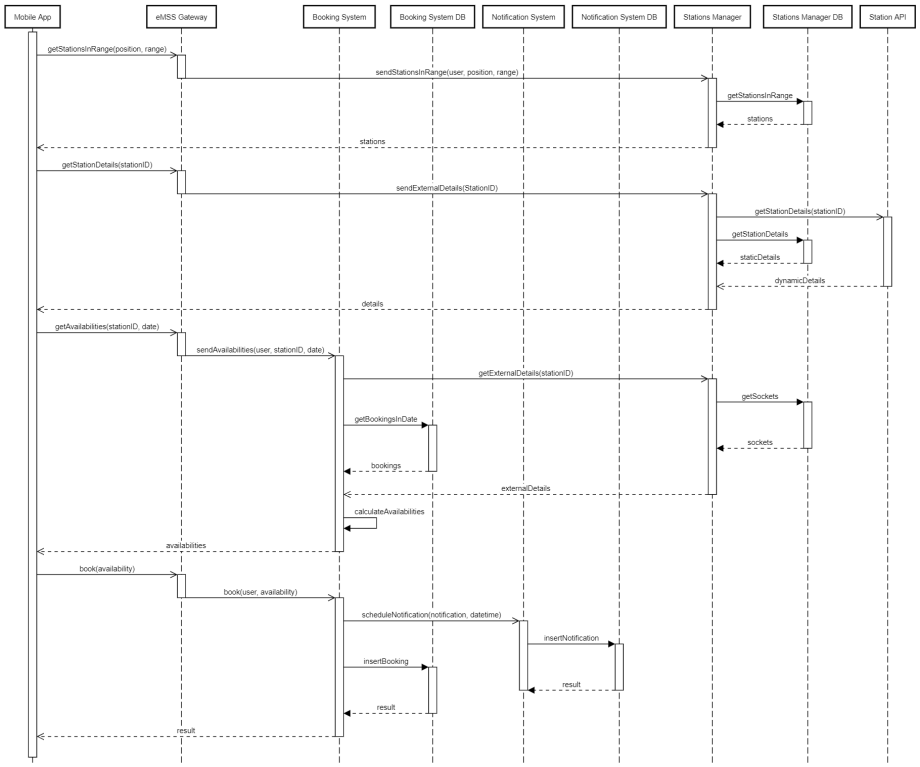
2.6.1 Mobile registration



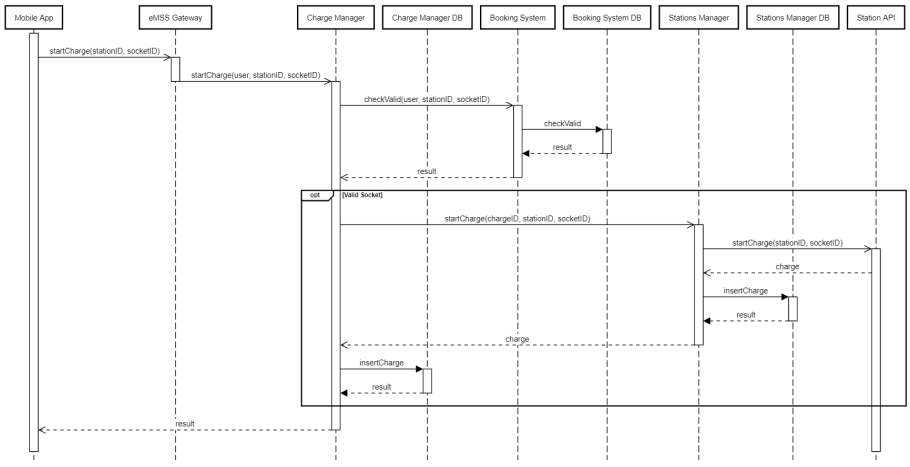
2.6.2 Mobile login



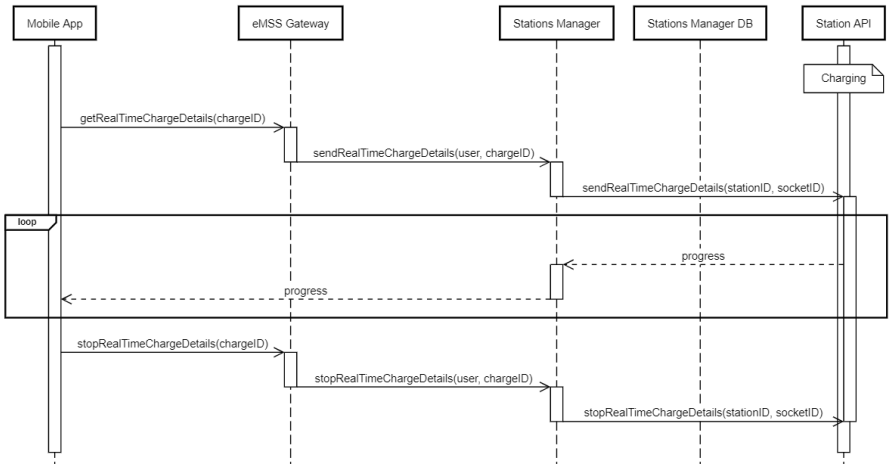
2.6.3 Booking a charge



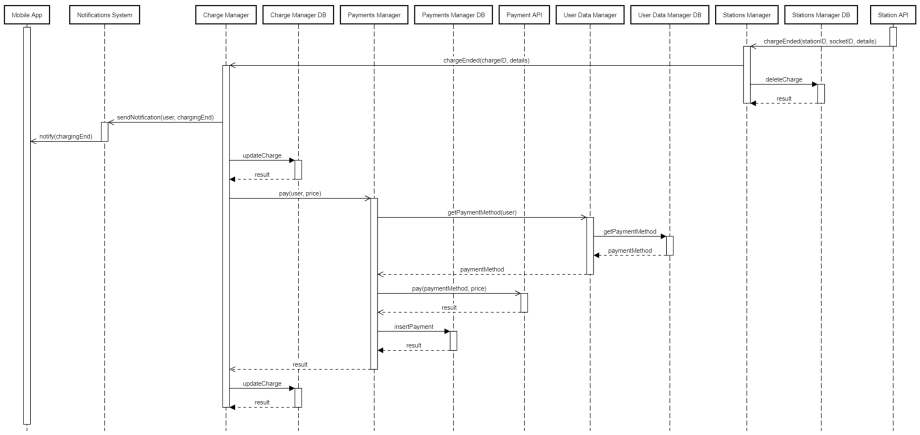
2.6.4 Start charging



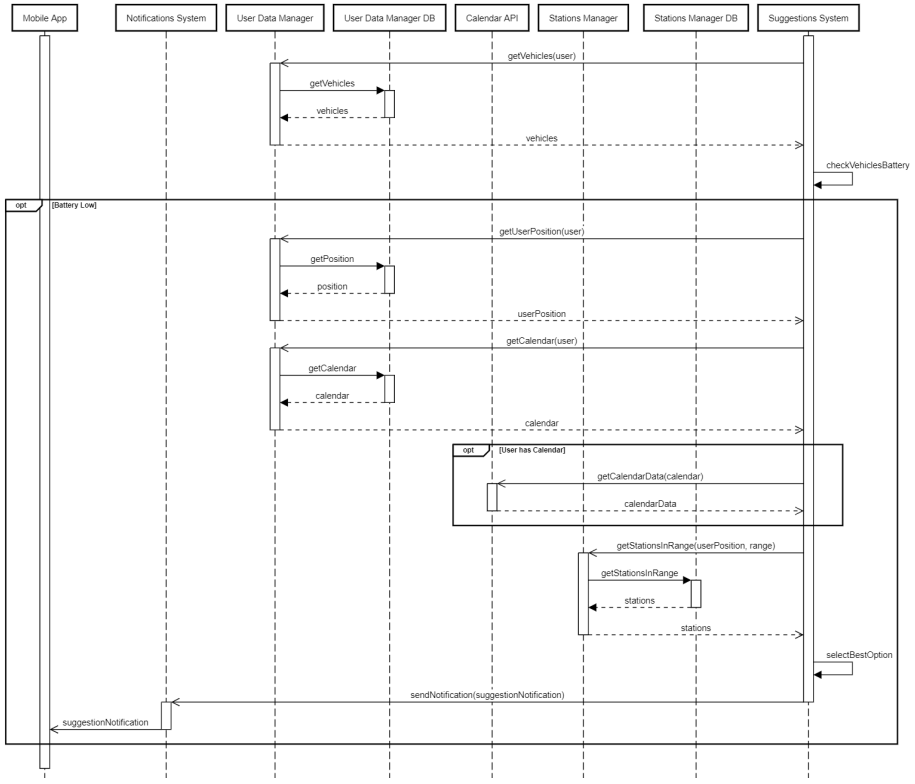
2.6.5 Real-Time Charge Details



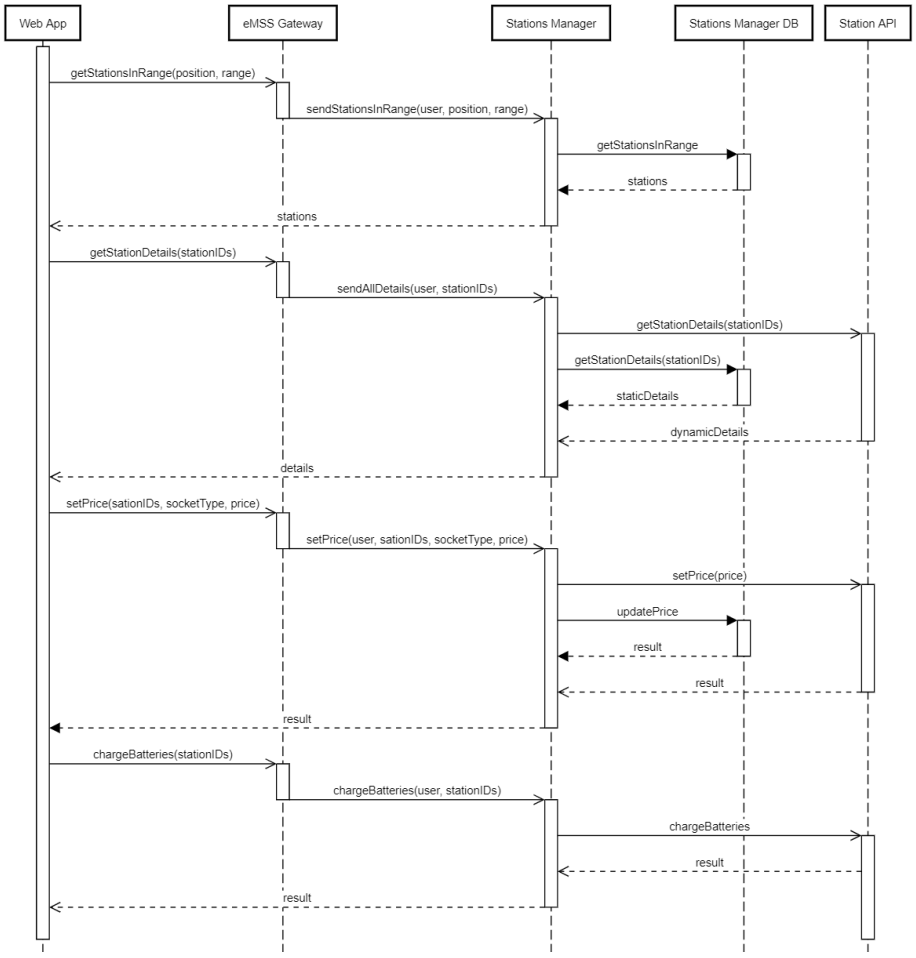
2.6.6 Charge End



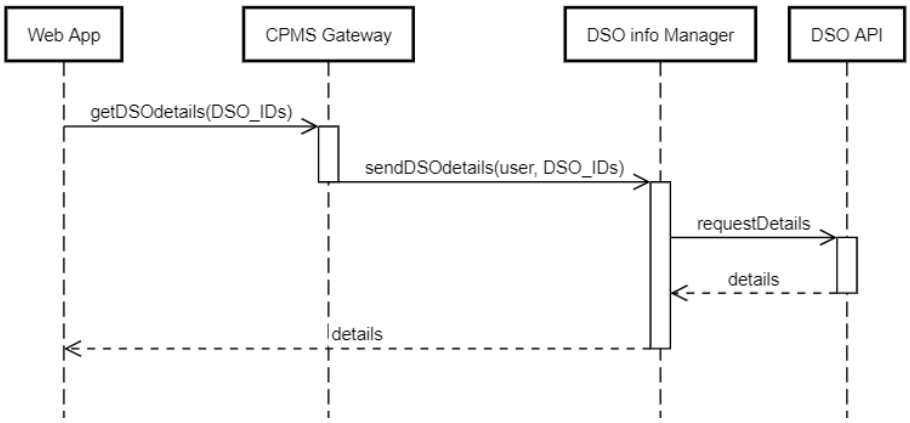
2.6.7 Suggestion



2.6.8 Set station parameters



2.6.9 View DSO details



3 User Interface Design

3.1 UI mockups

In the following section the mobile and the web core User Interfaces are presented.

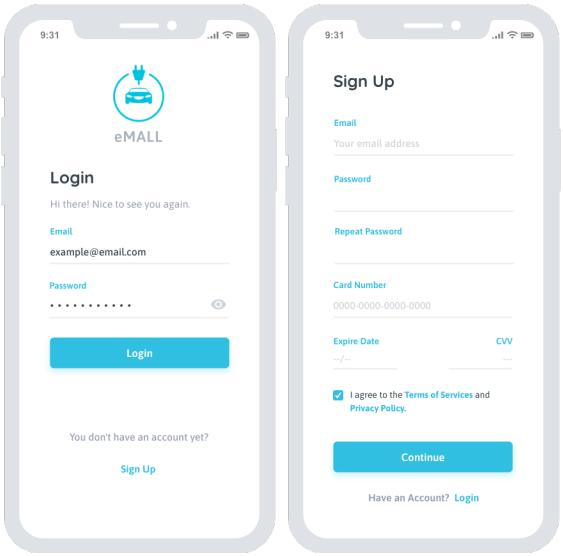


Figure 3.1: mobile app Login and Sing Up interface

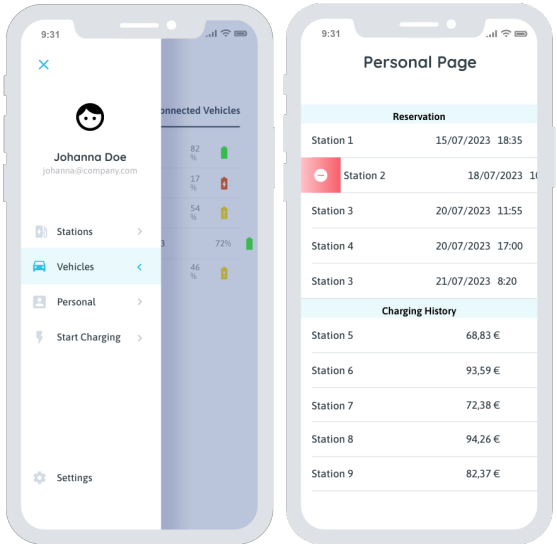


Figure 3.2: mobile *menu* interface and end-user's *personal page* interface

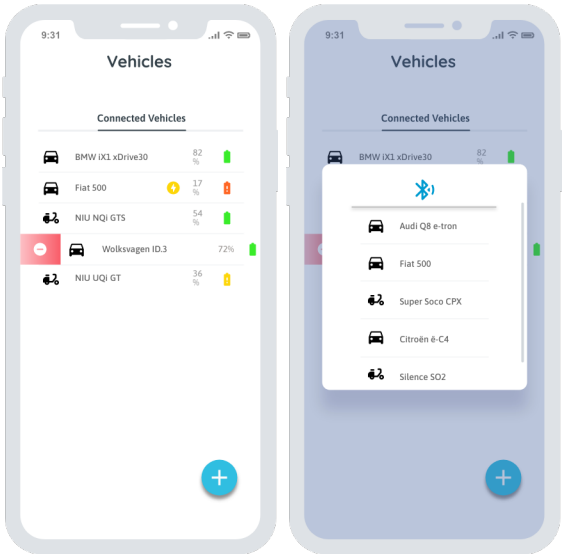


Figure 3.3: mobile *vehicles page* interface and *add vehicle* interface

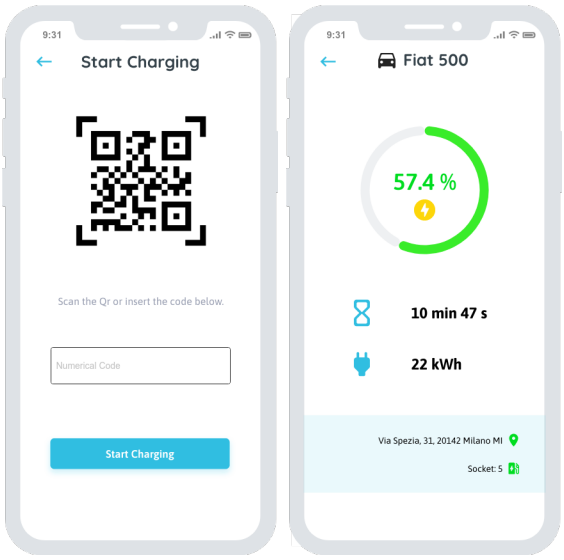


Figure 3.4: mobile *start charging* interface and *charging progress* interface

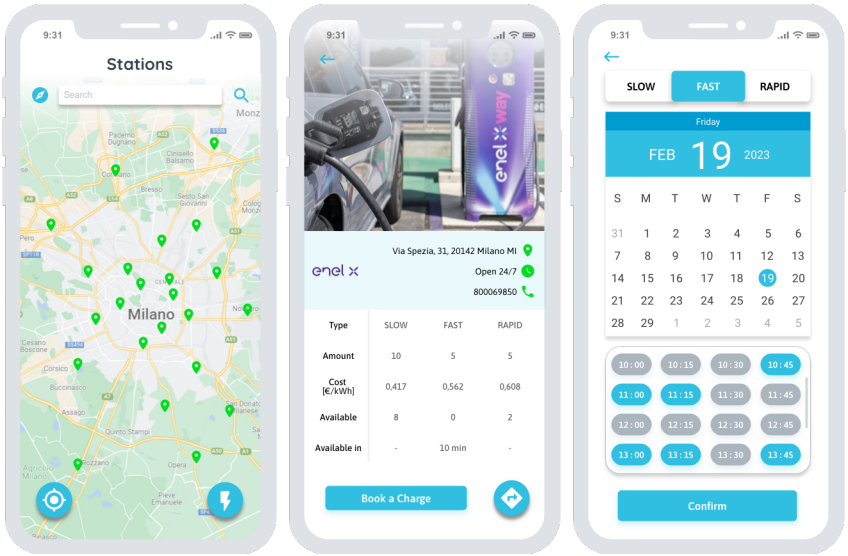


Figure 3.5: mobile *charging stations map* interface, *station details* interface and *booking* interface

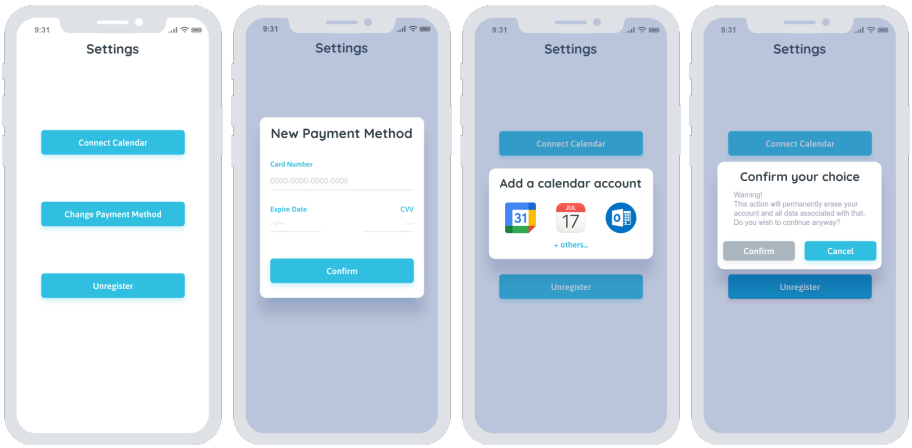


Figure 3.6: mobile *Settings page* interface with *add Calendar*, *change Payment Method* and *Unregister* interfaces

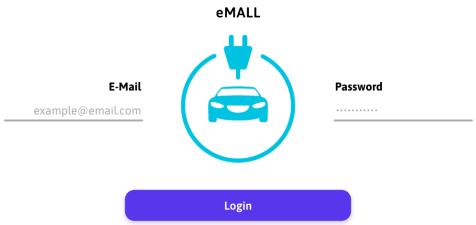


Figure 3.7: web *Login* interface

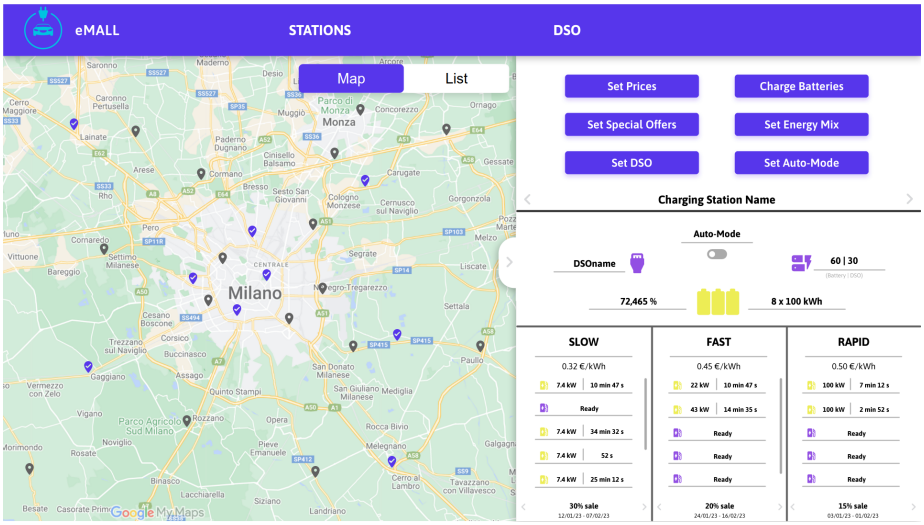


Figure 3.8: web *Stations Map View* interface

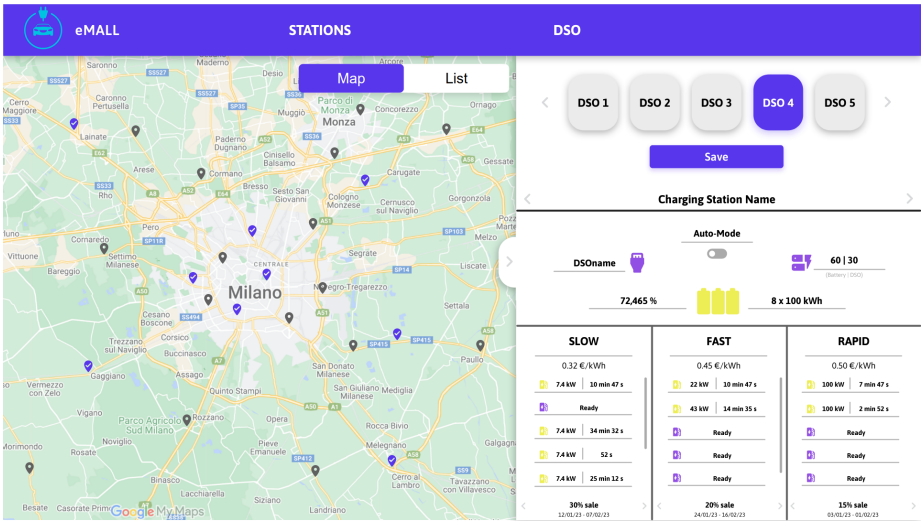


Figure 3.9: web *set DSO* interface

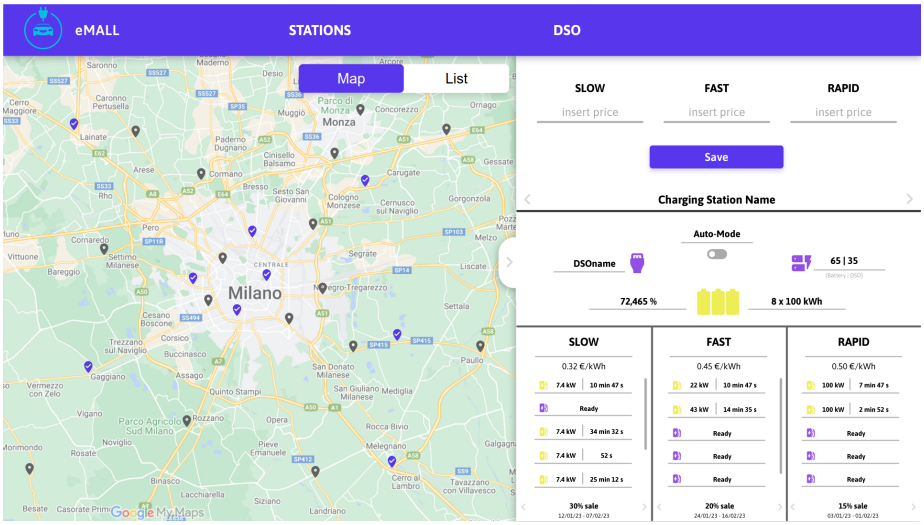


Figure 3.10: web *set Prices* interface

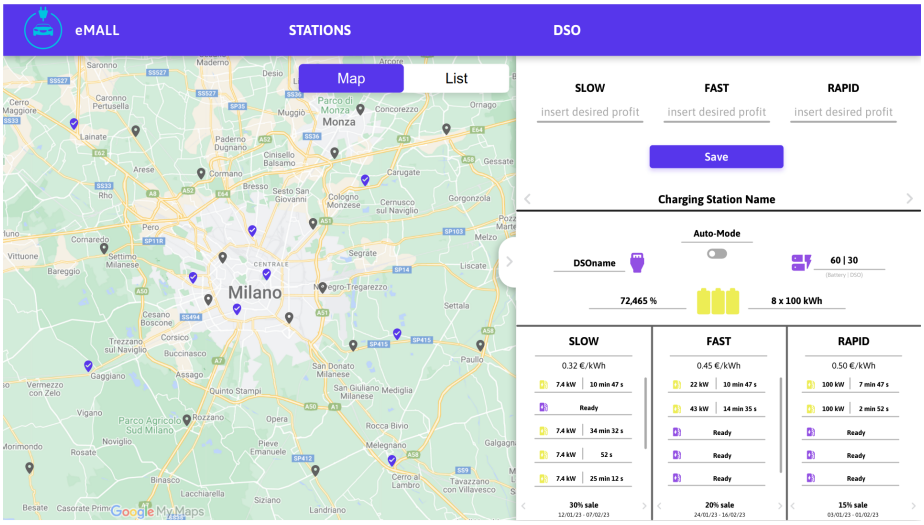


Figure 3.11: web *set Auto-Mode* interface

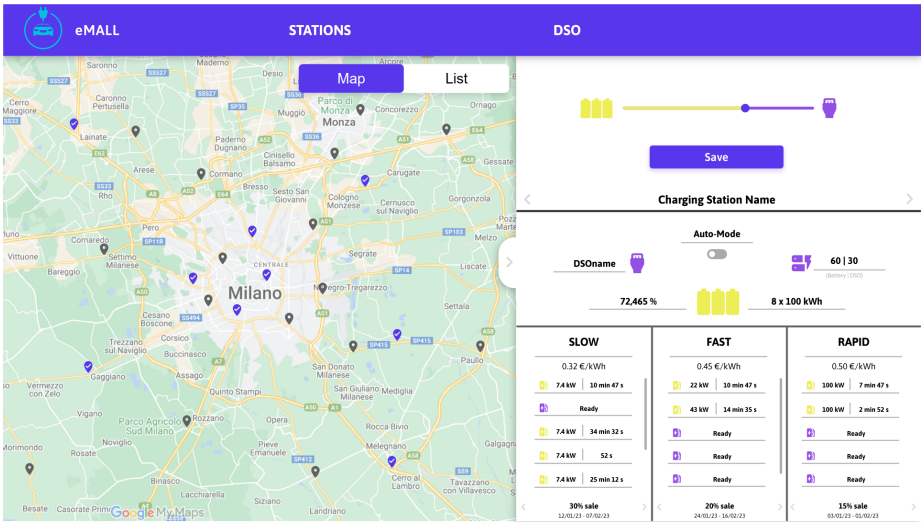


Figure 3.12: web *set Energy Mix* interface

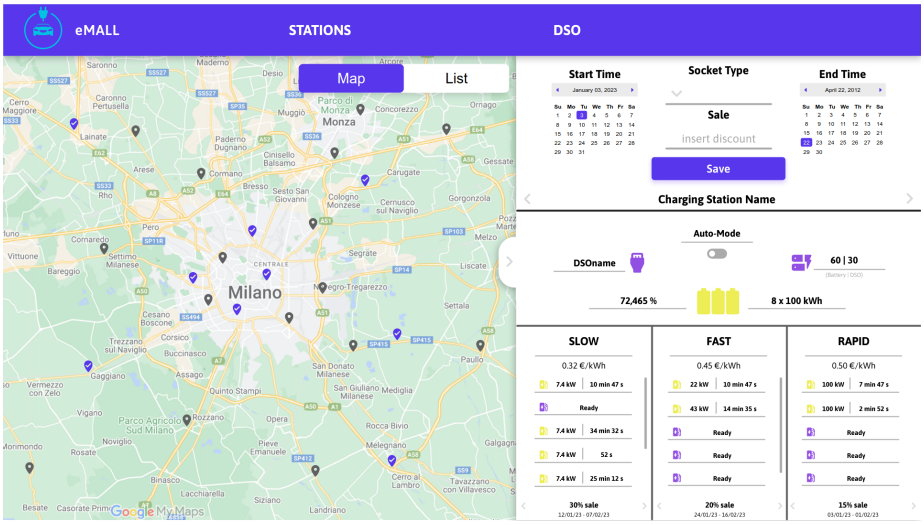


Figure 3.13: web *set Special Offer* interface

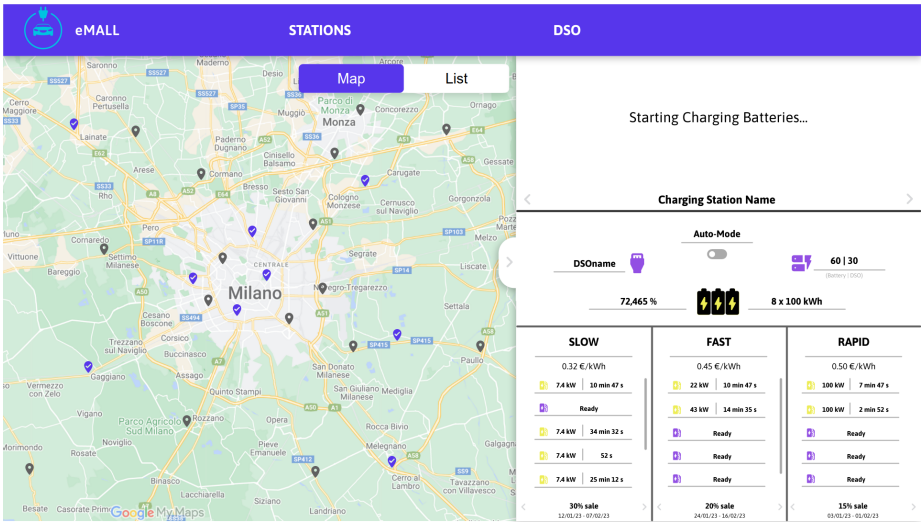


Figure 3.14: web *charge batteries* interface

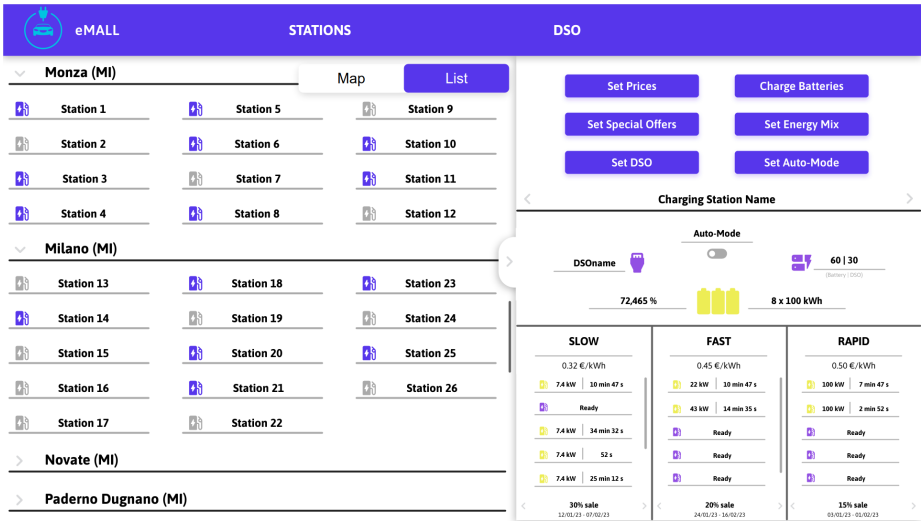


Figure 3.15: web *Stations List View* interface

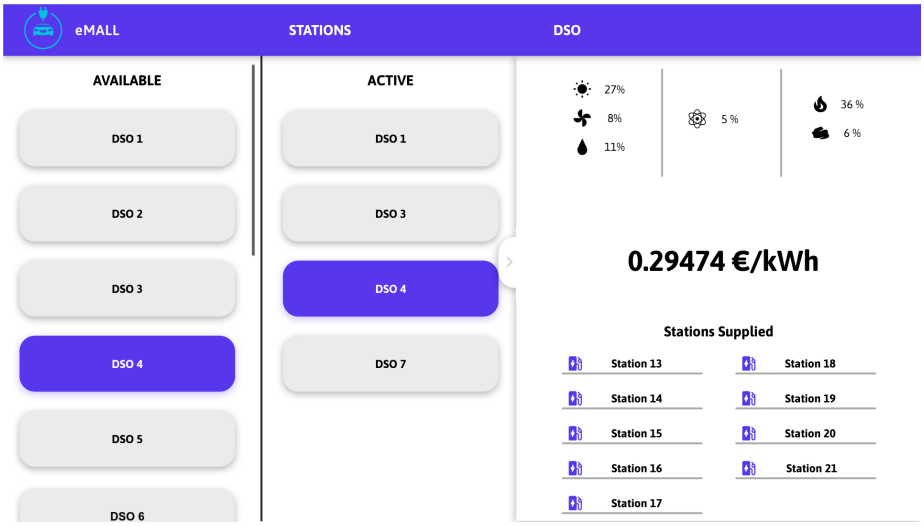


Figure 3.16: web *DSO List* interface

3.2 Mobile Application flow diagram

The following diagrams represent the end-user interface flow through the mobile application.

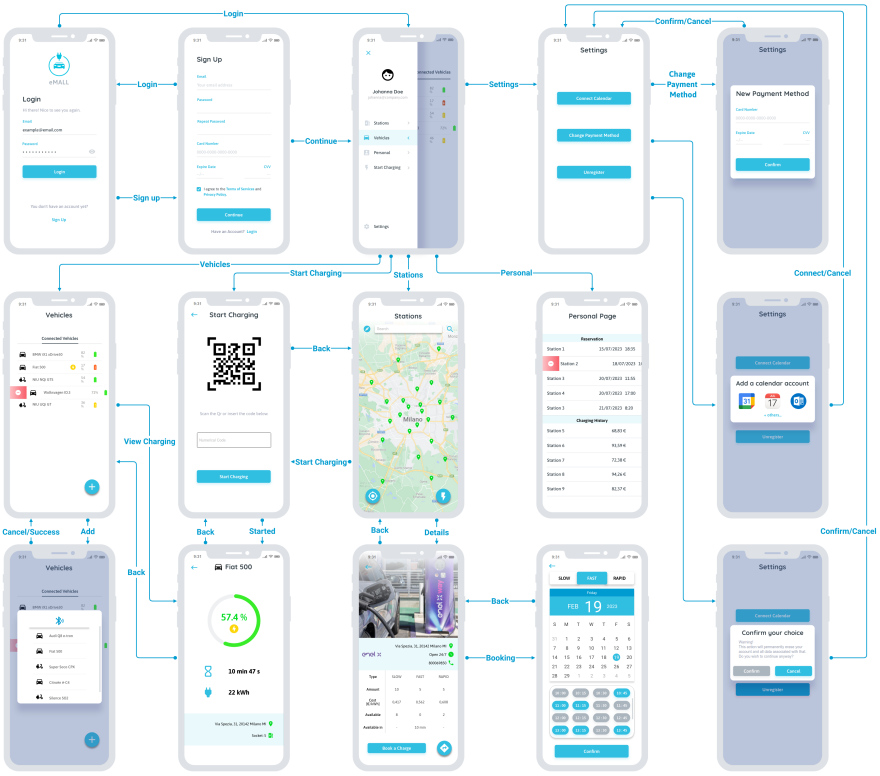


Figure 3.18: mobile flow diagram

4 Requirements Traceability

4.1 End-User

Requirement	Description
R1	The system shall allow an unregistered end-user to register **Mobile App; eMSS Gateway; User Data Manager; Payment Manager**
R2	The system shall allow a registered end-user to unregister **Mobile App; eMSS Gateway; User Data Manager**
R3	The system shall allow a registered user to login **Mobile App; eMSS Gateway; Authentication Manager; Notification System**
R4	The system shall display a navigable map of the user's surrounding area in the end-user's main page, representing charging stations at their location

	Mobile App; eMSS Gateway; Station Manager
R5	The end-user should be able to connect his vehicle(s) to the system from the vehicles page **Mobile App; eMSS Gateway; User Data Manager**
R6	The end-user should be able to disconnect his vehicle(s) from the system from the vehicles page **Mobile App; eMSS Gateway; User data manager**
R7	The end-user should be able to view the details of the external status of a specific charging station in a dedicated details page **Mobile App; eMSS Gateway; Stations manager**
R8	The end-user should be able to book a charge in a specific charging station for a certain (available) time frame from its details page **Mobile App; eMSS Gateway; Booking system; Stations Manager**
R9	The system shall send a phone notification and an email to the end-user 1 hour before his reservation starts, to remind him not to miss the appointment **Mobile App; eMSS Gateway; Booking system; Notification system**
R10	The end-user should be able to start the charging process at a specific charging socket from the stations page if and only if a vehicle is correctly plugged into that socket and the socket is not reserved for someone else **Mobile App; eMSS Gateway; Charge Manager**
R11	The system shall control the charging processes, starting them when requested, deciding the amount of power to use for each and monitoring them to know when they have ended **Mobile App; eMSS Gateway; Station Manager**
R12	The system shall send a phone notification and an email to the end-user whenever one of its current charging processes ends **Mobile App; eMSS Gateway; Notifications System; Charge manager**
R13	The system shall charge the cost of the service to the end-user, using the chosen payment method, whenever a charge ends **Mobile App; eMSS Gateway; Charge manager; Payments manager**
R14	The end-user should be able to view and cancel his current reservations from his personal page **Mobile App; eMSS Gateway; User data manager; Booking system**
R15	The end-user should be able to view the history of charges and payments from his personal page

	Mobile App; eMSS Gateway; Payment Manager; Charge manager
R16	The end-user should be able to change his payment method from the settings page **Mobile App; eMSS Gateway; User data manager; Payments manager**
R17	The end-user should be able to connect his calendar cloud service to the system (if supported) from the settings page **Mobile App; eMSS Gateway; User data manager**
R18	The end-user should be able to disconnect his calendar cloud service from the system from the settings page **Mobile App; eMSS Gateway; User data manager**
R19	The system shall periodically inspect the battery level of the end-user's vehicle(s), the end-user's position and calendar (if linked to the system) to send a phone notification and an email suggesting a charge if the battery level is under 30%. The suggestion must recommend a station near to the user, taking into account the distance from the user, the availability of charging sockets and any current special offers, and it must propose a time that does not overlap with any of the tasks in the user's calendar **Mobile App; eMSS Gateway; Notifications system; User data Manager; Suggestions system; Station Manager**

4.2 CPO

R20	The system shall display a navigable map in the CPO's main page representing proprietary charging stations at their location **Web app; CPMS gateway; Stations Manager**
R21	The CPO should be able to view the details of the external and internal status of a proprietary charging station (or group of stations) in a dedicated details section **Web app; CPMS gateway; Stations Manager**
R22	The CPO should be able to set the prices of the charging types (slow/-fast/rapid) of a specific proprietary station (or group of stations) from its details section **Web app; CPMS gateway; Stations Manager; DSO info manager**
R23	The CPO should be able to set special offers on a specific proprietary station (or group of stations) from its details section **Web app; CPMS gateway; Stations Manager**
R24	The CPO should be able to set the DSO of a specific proprietary station (or group of stations) from its details section

	Web app; CPMS gateway; Stations Manager; DSO info manager
R25	The CPO should be able to recharge the storage batteries of a specific proprietary station (or group of stations) from its details section **Web app; CPMS gateway; Stations Manager**
R26	The CPO should be able to decide the mix of energy sources for charging (storage batteries and DSO energy) in a specific proprietary station (or group of stations) from its details section **Web app; CPMS gateway; Stations Manager; DSO info manager**
R27	The CPO should be able to set a specific proprietary station (or group of stations) in auto-mode from its details section **Web app; CPMS gateway; Stations Manager**
R28	The CPO should be able to overwrite the auto-mode for a specific charging station (or group of stations) by manually setting prices, the DSO or the energy mix or by charging batteries **Web app; CPMS gateway; Stations Manager; DSO info manager**
R29	The CPO should be able to view the list of all available DSOs from the DSO page **Web app; CPMS gateway; Stations Manager; DSO info manager**
R30	The CPO should be able to view the list of all the DSOs supplying proprietary stations from the DSO page **Web app; CPMS gateway; Stations Manager; DSO info manager**
R31	The CPO should be able to view the details of a specific DSO in a dedicated details section **Web app; CPMS gateway; Stations Manager; DSO info manager**

5 Implementation, Integration and Test Plan

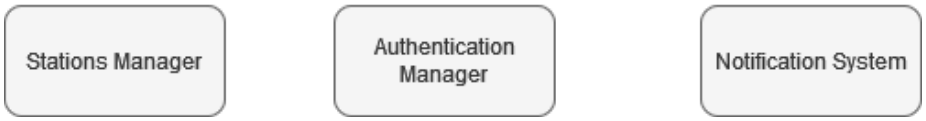
This section presents a series of steps to be taken in order for the implementation, integration and testing to occur properly. Using a bottom-up approach, the whole system will be built step by step and the testing phase can be done synchronously to the implementation one.

5.1 Integration

This section will describe the different integration steps to be taken, following a bottom-up approach. The icons represent the components and the lines represent the dependencies.

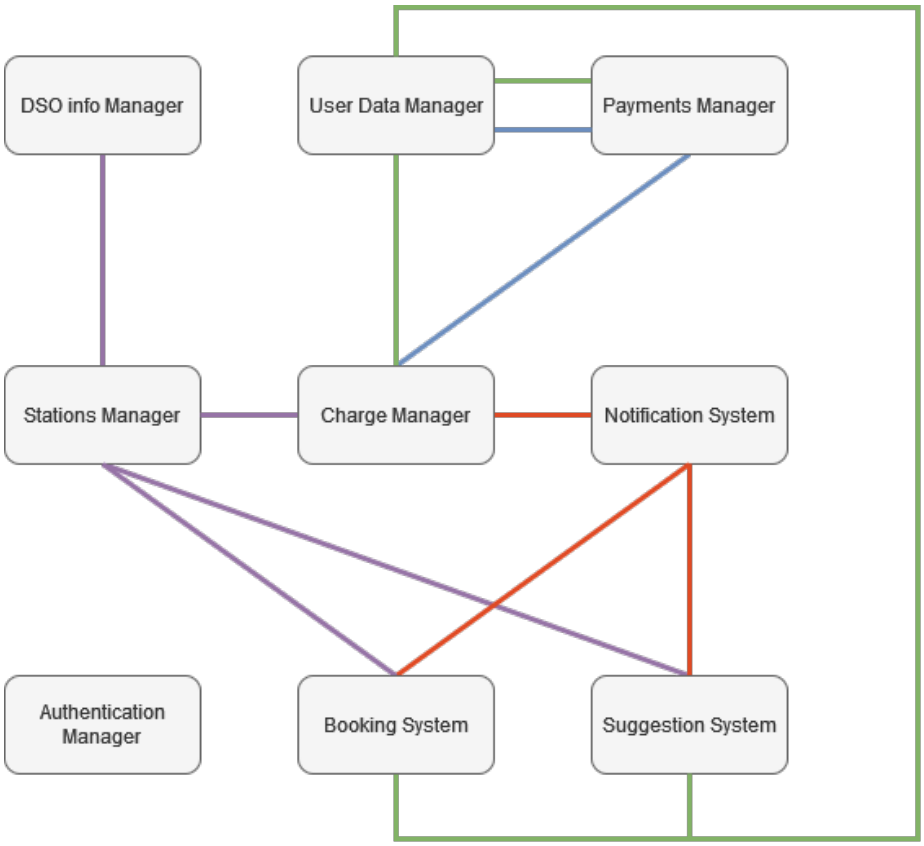
5.1.1 Integration step 1

First we can implement the independant components, the ones that doesn't have any dependency.



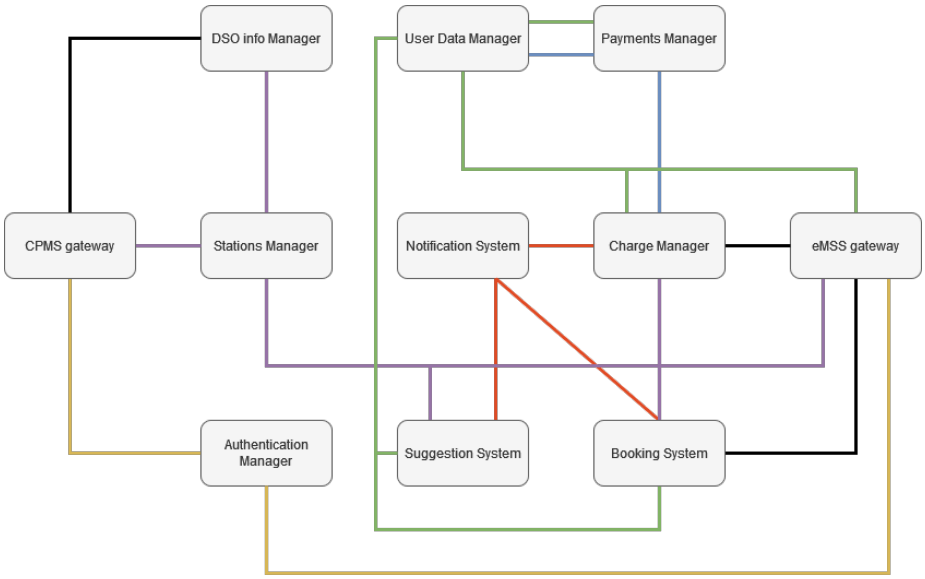
5.1.2 Integration step 2

With all the fundamental components integrated, we can proceed to implement other core components that depends on them. For example: Charge Manager and Booking System.



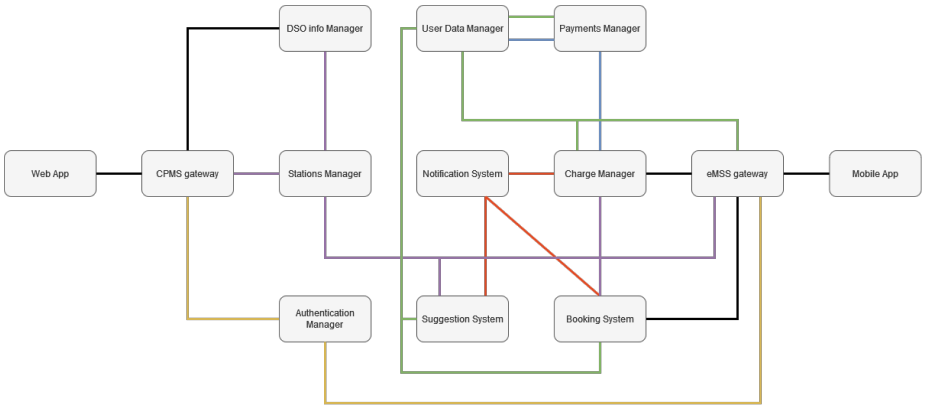
5.1.3 Integration step 3

Now that all the core components are integrated, we are able to integrate the two gateways



5.1.4 Integration step 4

In this step our system is already funcional and fully integrated. The final step consists in the integration of the Mobile App and Web app.



5.2 Testing

The testing should be synchronized with the development steps. Each component should be test individually before integrating and there should also be tests of every dependency to guarantee a correct integration, every connection between components should be properly tested before moving on. Another aspect is testing the system integration, for example testing the CPMS after step 2 to guarantee that the interactions between components are properly working.

6 Effort spent

6.1 Niccolò Nicolosi

Task	Hours
Architectural styles	0:30
Component view	10:30
Component interfaces	2
Deployment view	3
Revision and adjustments	19
Total	35

6.2 Francesco Negri

Task	Hours
Revision and adjustments	6
UI mockups	21
Sequence Diagrams	8
Mapping on Requirements	1
Total	36

6.3 Marcos Pietrucci

Task	Hours
Latex document setup	00:30
Introduction	00:30
Requirements mapping	1
System Overview	8
Sequence Diagrams	6
Integration	8
Revision and adjustments	7
Total	31:00