



POLITECNICO
MILANO 1863

M.Sc. Computer Science and Engineering

Software Engineering 2 project

eMall - e-Mobility for All

Design Document

Christian Di Diego, Federica Di Filippo, Carmine Faino

08 January 2023

GitHub Repository:

<https://github.com/FedericaDiFilippo/DiDiegoDiFilippoFaino>

1.Introduction.....	3
1.1 Purpose	3
1.2 Scope	3
1.3 Definitions, Acronyms, Abbreviations	3
1.4 Revision history	3
1.4.1 Reference Documents.....	3
1.5 Document Structure.....	3
2. Architectural Design	4
2.1 Overview.....	4
2.2 Component view	5
2.3 Deployment view	6
2.4 Runtime view	7
2.5 Component Interfaces	17
2.6 Selected architectural styles and patterns	17
2.6.2 Three Tier architecture:	18
2.7 Other Design Decisions.....	18
2.7.1 External APIs:	18
2.7.2 Security Policy:.....	18
3.User Interface Design	19
4. Requirements Traceability	29
5. Implementation, Integration and Test Plan.....	30
6. Effort Spent.....	31

1.Introduction

1.1 Purpose

The Design Document provides a more technical overview of the Requirement Analysis and Specification Document (RASD) of the eMall application, describing the main architectural components, their communication interfaces, and their interactions. Furthermore, it shows some examples of User Interfaces.

1.2 Scope

This document will discuss the topic of electric mobility. The main parties involved are electric vehicle owners and CPOs, who interact through their employees, that are registered on the app. Users can view the prices and promotions offered by each CPO for the available charging stations. On the other hand, employees of the CPOs can adjust the energy prices at the stations they manage and post new offers on the app. In the context of the World and Machine paradigm, the machine refers to the part of the system being developed, while the environment is the world that interacts with the system.

1.3 Definitions, Acronyms, Abbreviations

CS - Charging Station

MVC – Model View Controller

1.4 Revision history

1.4.1 Reference Documents

- The specification document “Assignment RDD AY 2022-2023_v3”

1.5 Document Structure

This document is divided into six sections:

Section 1 introduces the problem and the main purposes of the eMall application.

Section 2 presents an overview of the project’s architecture and contains sequence diagrams to explain the interaction between Users and all the system’s components.

Section 3 shows some examples of the principal graphical interfaces for the application.

Section 4 explains which components of the architecture are necessary to satisfy the functional requirements presented in RASD.

Section 5 describes the integration and testing plan, specifying the order of component to be implemented and tested.

Section 6 explain the effort spent by the group in order to complete the project and shows a list of the activities done.

2. Architectural Design

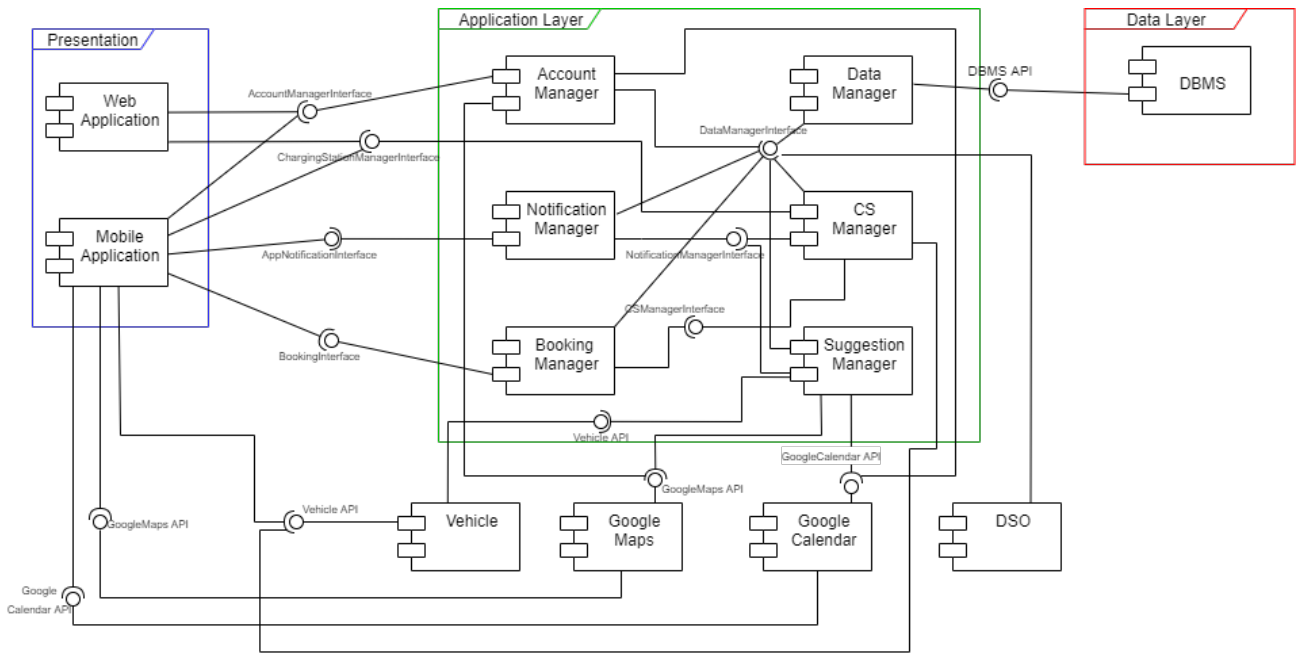
2.1 Overview



The application is developed through the Client-server architecture. To increase scalability and maintainability it is divided into three layers: Presentation Layer, Application Layer (which is divided into Web tier and Business tier) and Data Layer.

- Presentation Layer: it is the User interface and communication tier of the application, where the end User interacts with the application. This tier can be run both on smartphone (as an app) and on browser (as a web app), however the two interfaces present different functionalities.
- Application Layer: includes the execution of applications and the flow of the control program, i.e., the logic that deals with the validation of data and requests/responses in communication between processes of the various levels. It is divided in Web Tier, which contains the servlets and connects the Business Tier with the Presentation Layer, and Business Tier, which is responsible for handling User requests and generating responses.
- Data Layer: this layer contains the persistent data that are necessary for the system.

2.2 Component view



Presentation Layer:

This is the front-end layer, through which Users can interact with the system. As already said in RASD, the available functionalities depend on the type of User and on the device used. The interfaces of the mobile application and web application are similar, however because of the different functionalities available, the external interfaces needed may vary from one to the other.

Application Layer:

The Application Server realizes the business logic, its role is to compute all the needed data and coordinate the flow of information between the application layer and the data layer. It consists in the following elements:

- Account Manager: manages authentication and authorization of Users, allows Users to modify their data. It interacts with the data manager to check Users' data and to update the information stored.
- Notification Manager: there are different kinds of notifications that are handled by the notification manager. The most important ones are:
 - The notification from eMSP to Customer when the vehicle needs to be charged;
 - The notification from eMSP to Customer when the charge is completed;
 - The notification from eMSP to Admin when DSOs' energy cost or sources have changed.

-Booking Manager: manages bookings, in particular reservations and deletions. It interacts with the data manager to modify bookings status and to check the presence of possible overlaps. It even handles the generation of OTPs after a booking has been correctly completed.

-Charging Station Manager: It is divided into Batteries Manager, which handle the batteries status, and Socket Manager, which updates the booking schedule of a socket taking data from Database and checks the correctness of OTP and UserID inserted by Customers.

-Suggestion Manager: this manager is active only for Users that shared GPS and Calendar. It checks the battery status and dynamically calculates the right moments and places to charge the vehicle. It interacts with the notification manager to communicate this information to Customers.

-Data Manager: this component connects the Application Layer with the physical Database. it provides to the other application components an easier way to store and retrieve data from the Database.

External APIs:

-Google Maps: this external system communicates with the Account Manager to give information about Customer position, that can be used both for suggestions and to find near charging stations. It allows the suggestion manager to handle charges.

-Vehicle: this external system shares information about the status of the battery both with the socket manager and the suggestion manager. It even communicates to the socket the maximum amount of power absorbed.

-Google Calendar: this external application, if shared, informs the suggestion manager about the Customer schedule, so that it can plan the charges without wasting of time.

-DSO: allows the eMSP application to take all the needed information about energy cost and sources. This information is periodically updated in the Database.

2.3 Deployment view

The deployment diagram illustrates the necessary components for the system to function properly, apart from the Maps APIs, the Calendar APIs and the Vehicle APIs. The tiers in the diagram are as follows:

Tier 1 is the client application, that make requests to the server, which processes the requests and returns a response. Clients can be a web browser or a downloaded application. The first communicates with the web tier, while the latter interacts directly with the application layer.

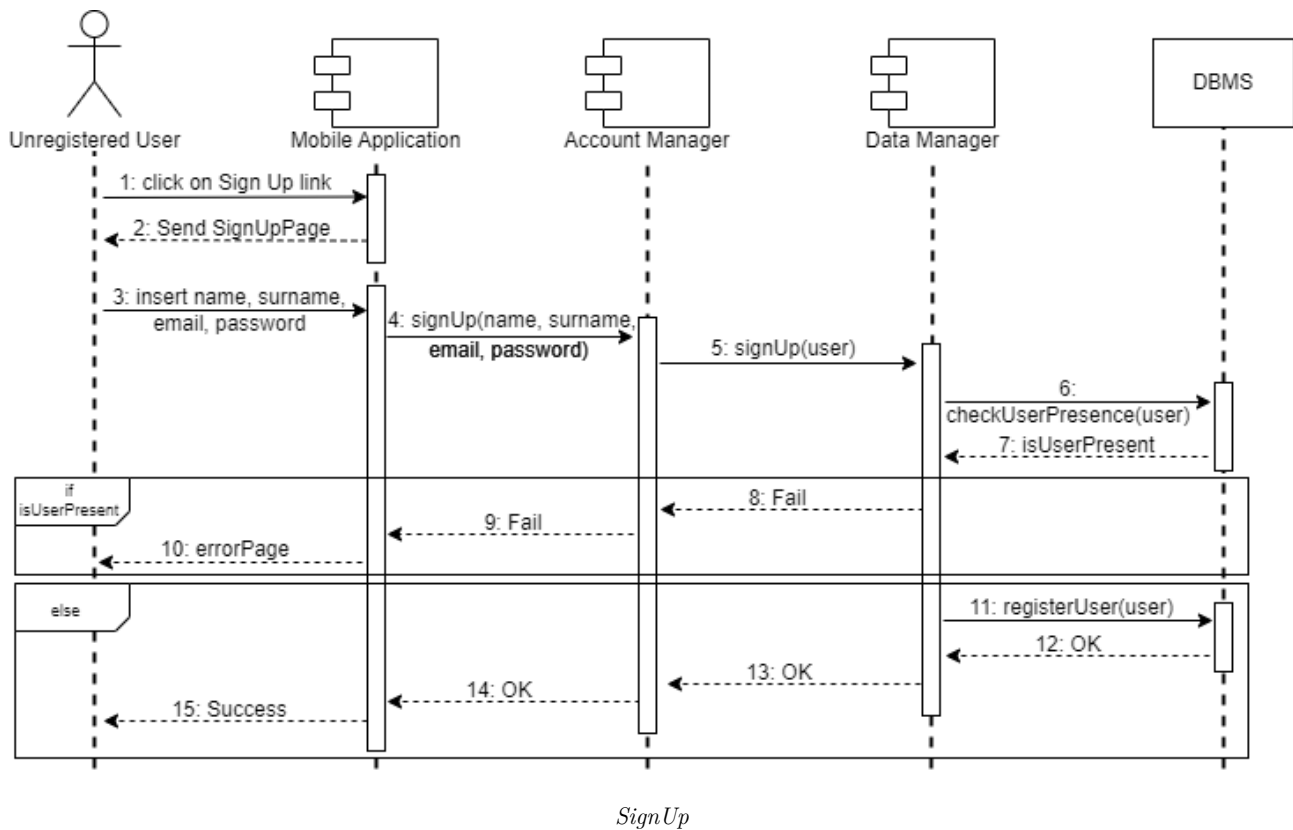
Tier 2 consists in the web server, that allows communication between the client and the application server. It handles HTTPS requests and responses.

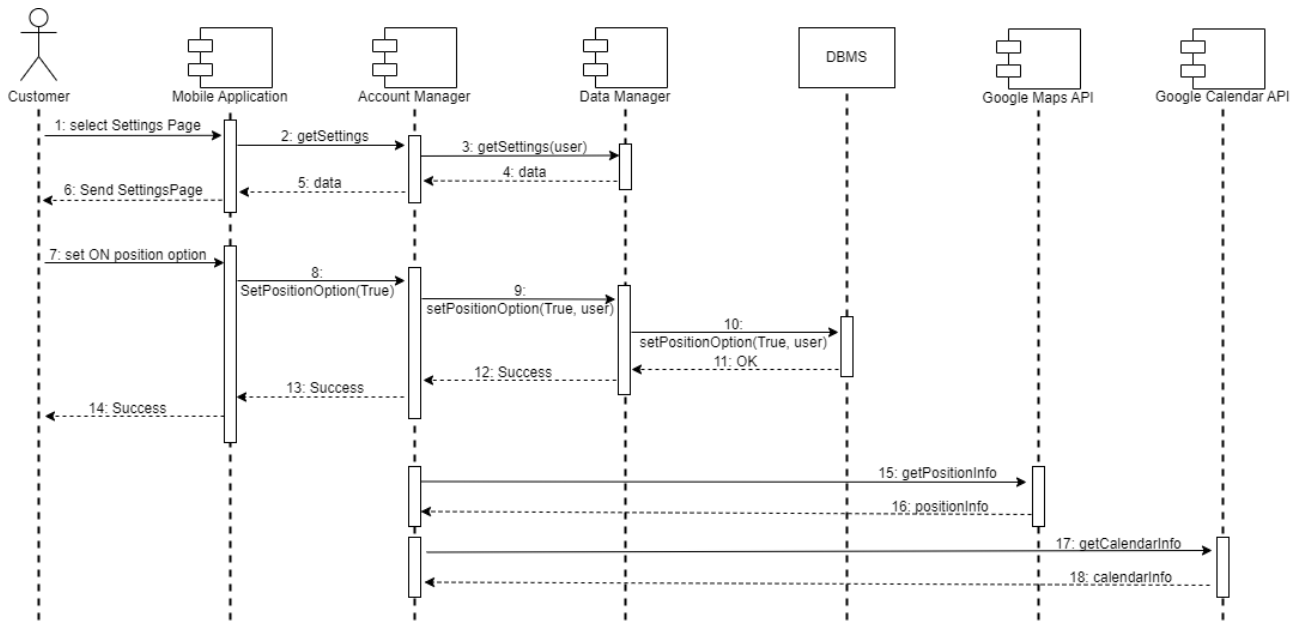
Tier 3: This tier contains the application servers, which run the main functions of the system. The entire application layer is mapped to this tier, which communicates with the client tier through APIs that are used by the applications (for the mobile app). The application servers also communicate with the data tier through a DBMS gateway.

Tier 4: This tier consists of DBMS servers that store the data and perform actions on it based on instructions from the application servers.

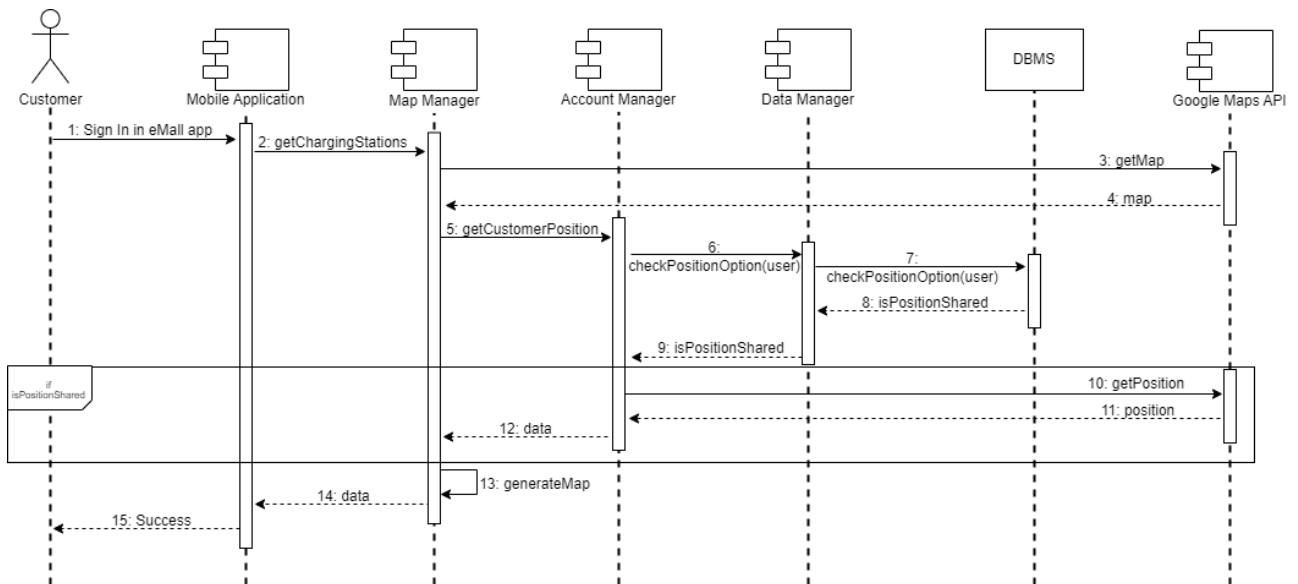
2.4 Runtime view

The following diagrams show the Runtime view from the mobile and web app’s perspective. The first one illustrates the main functionalities available for Customer, while the latter illustrate the ones available for Admin. Even if some functionalities work on both the devices, only one is reported for simplicity.

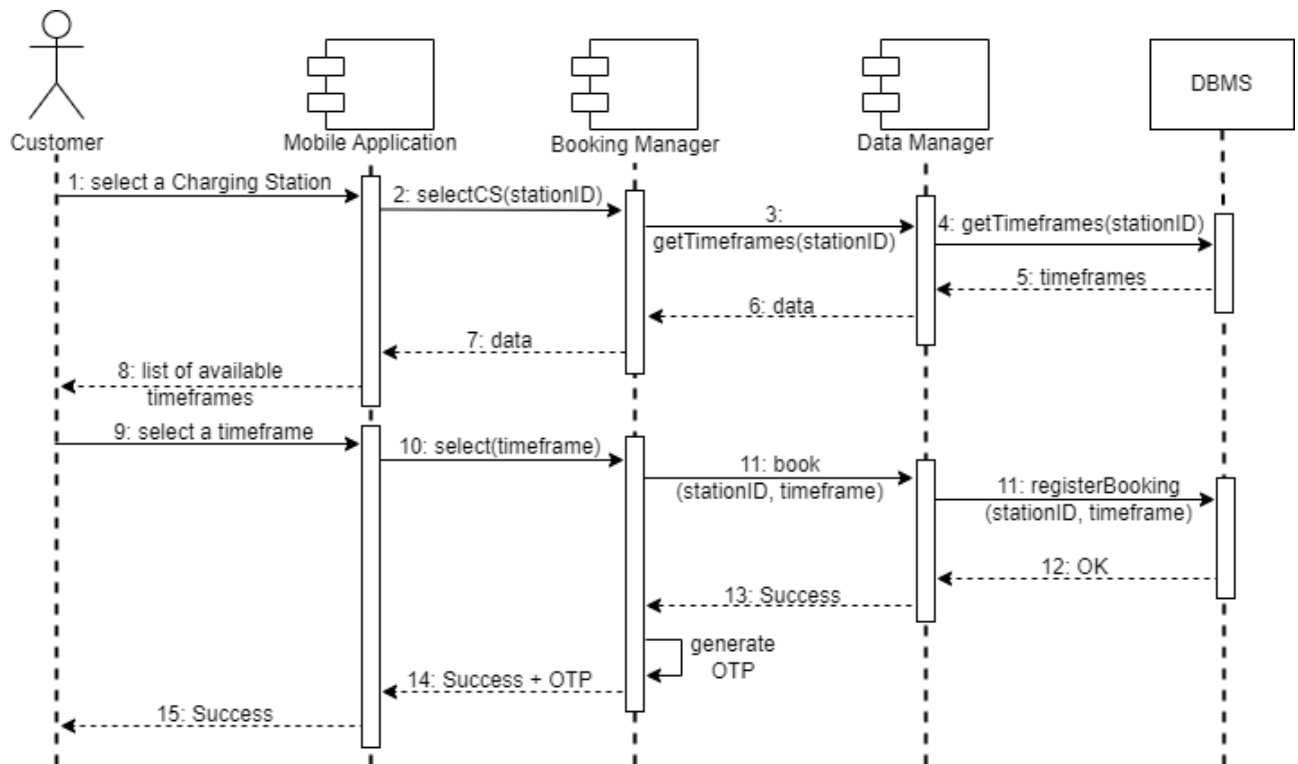




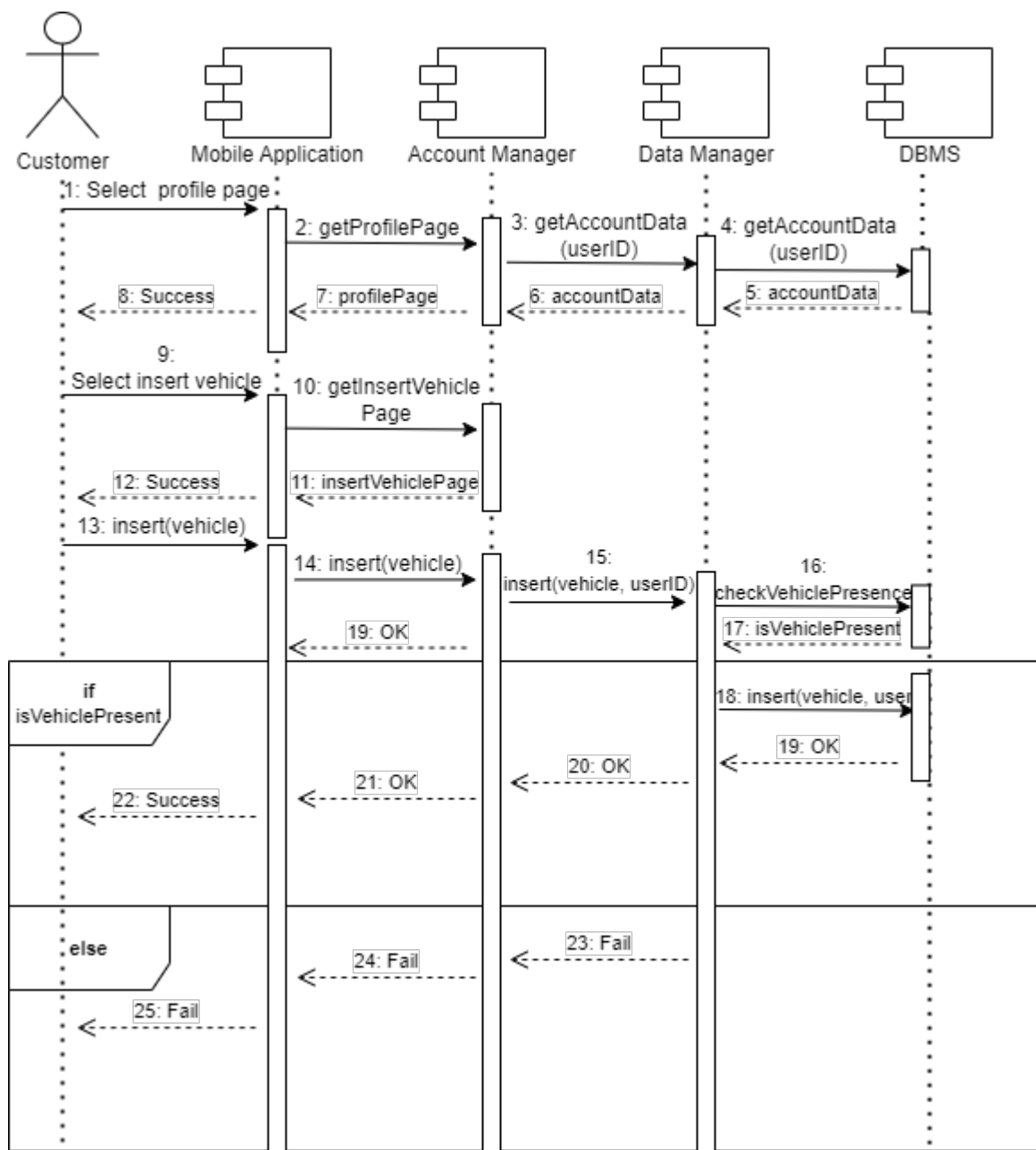
Modify Position Options



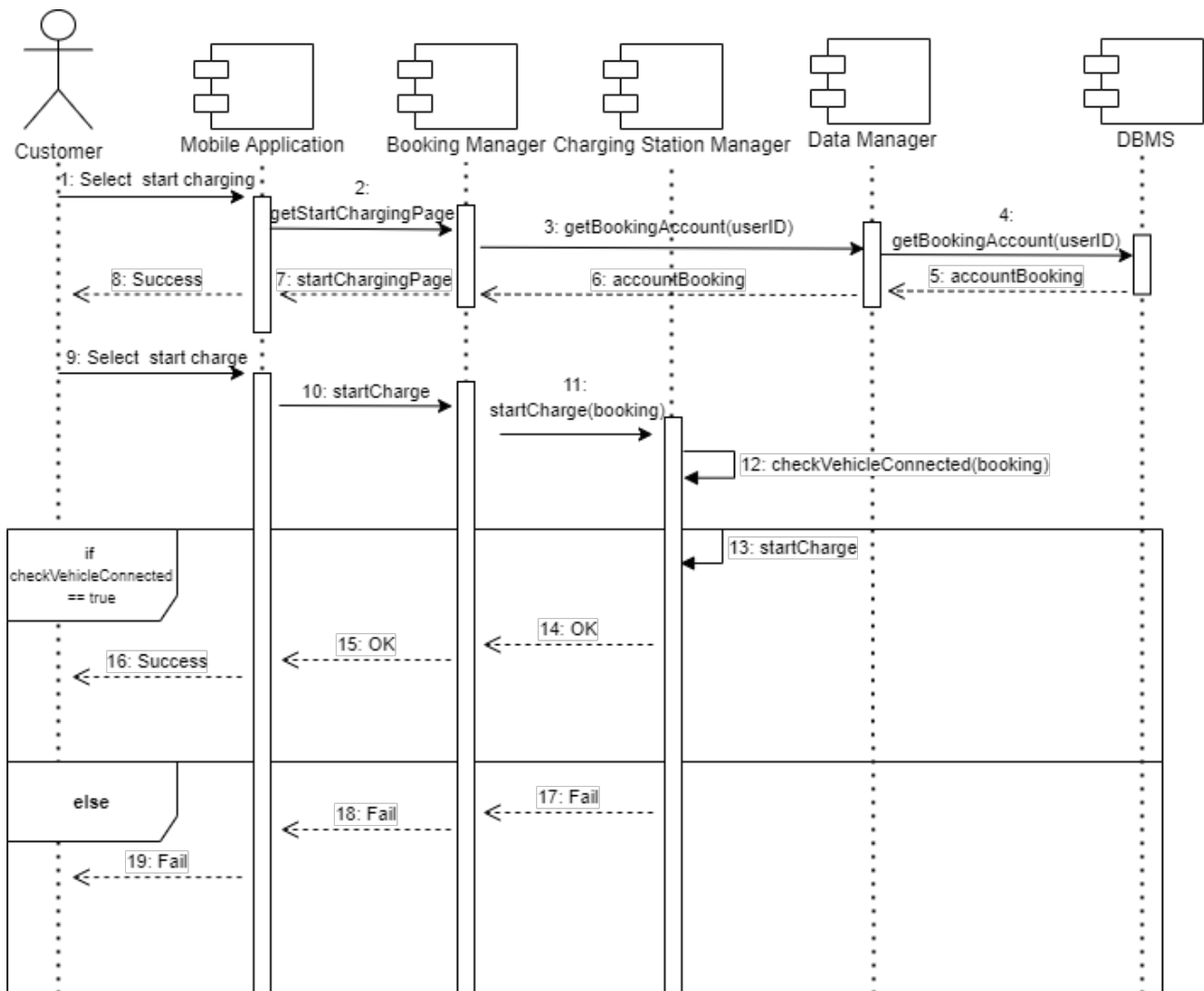
HomePage



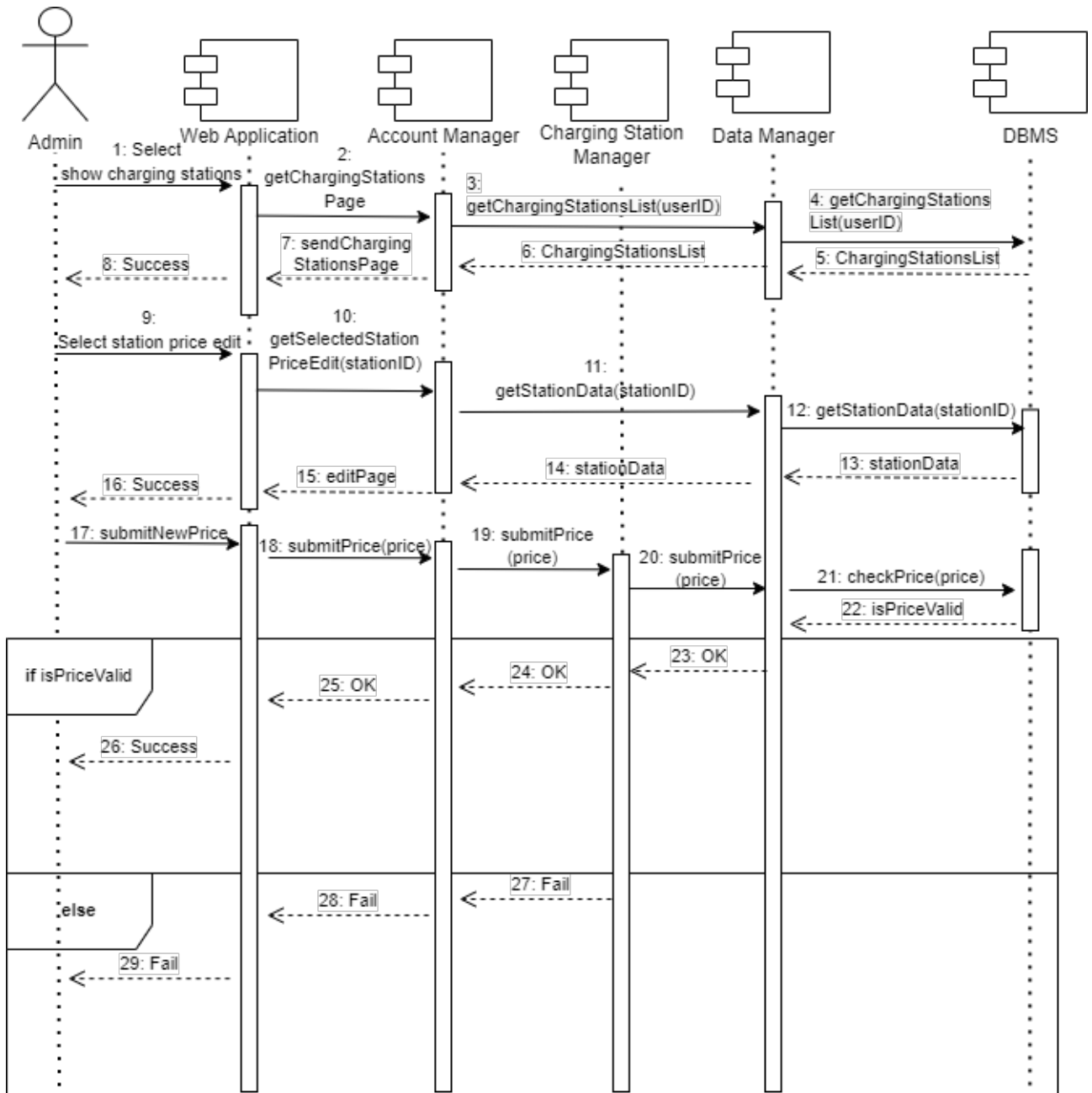
Booking



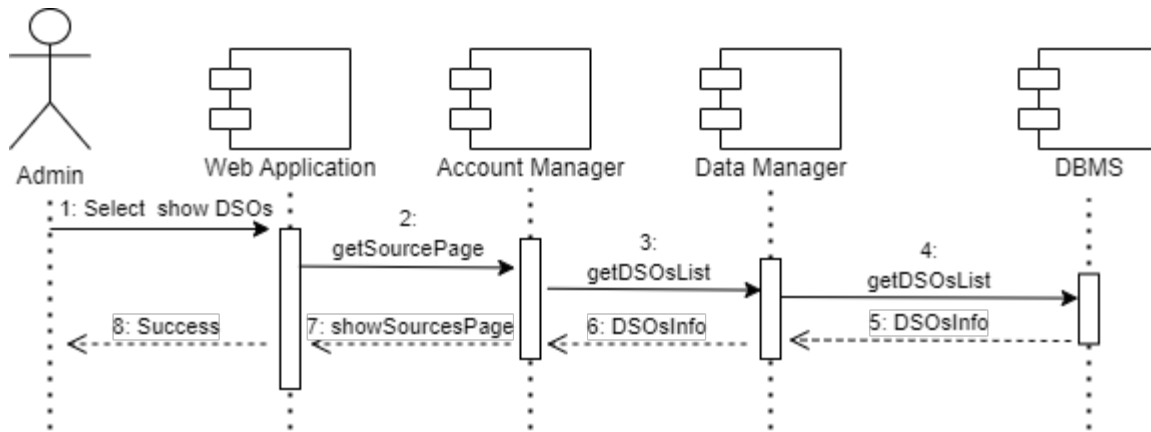
Vehicle's information insertion



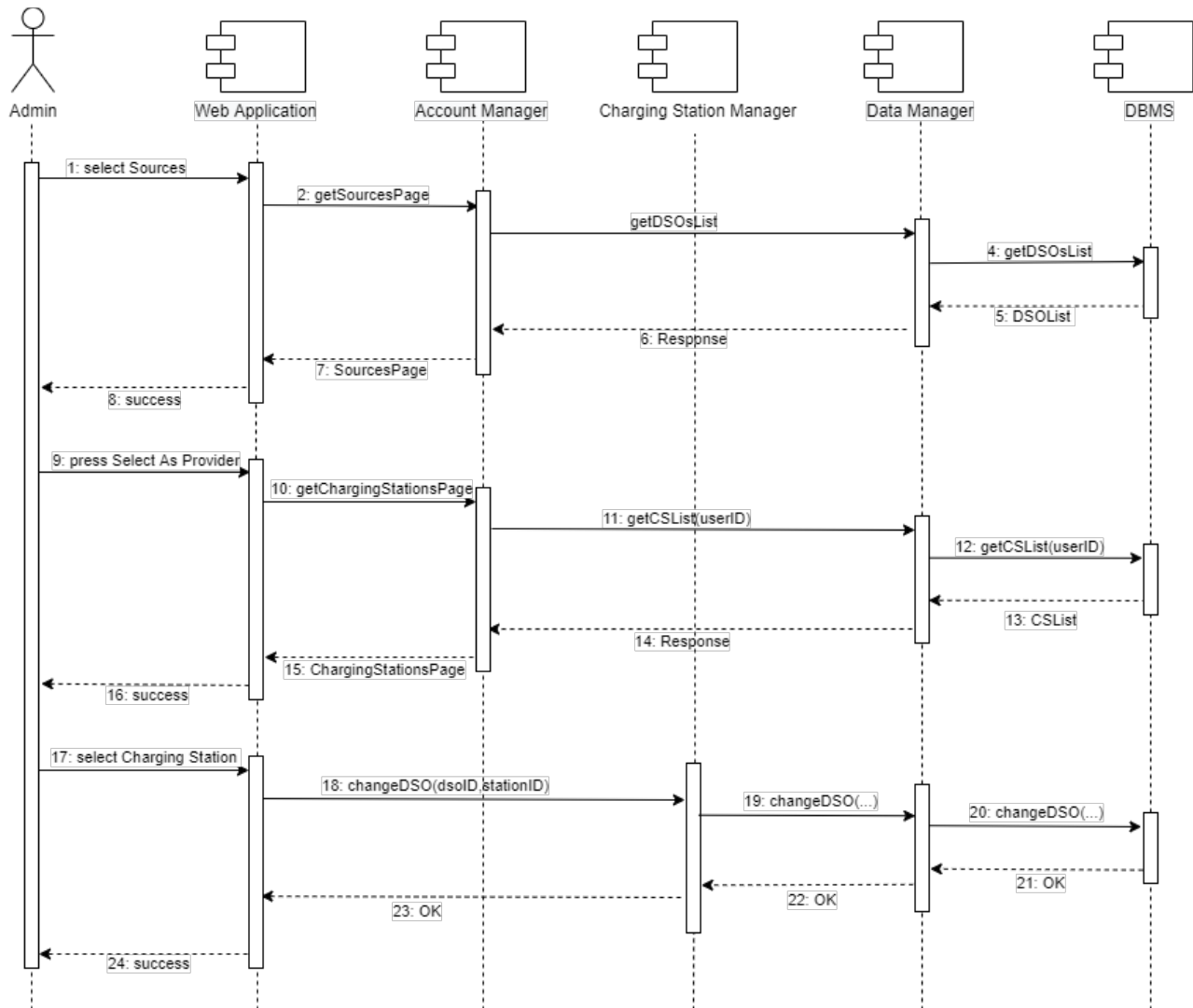
Start a charge



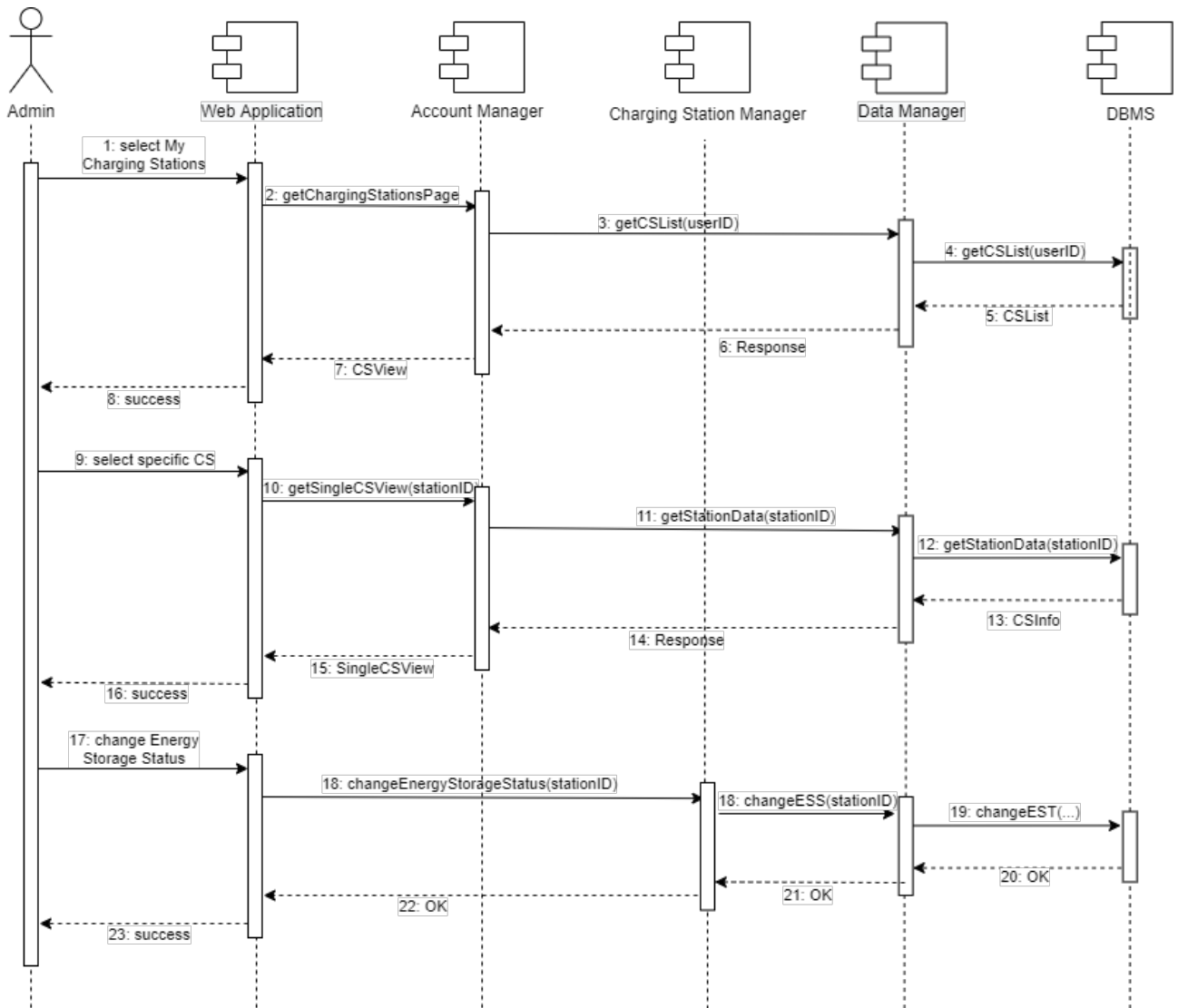
Admin publishes special offers



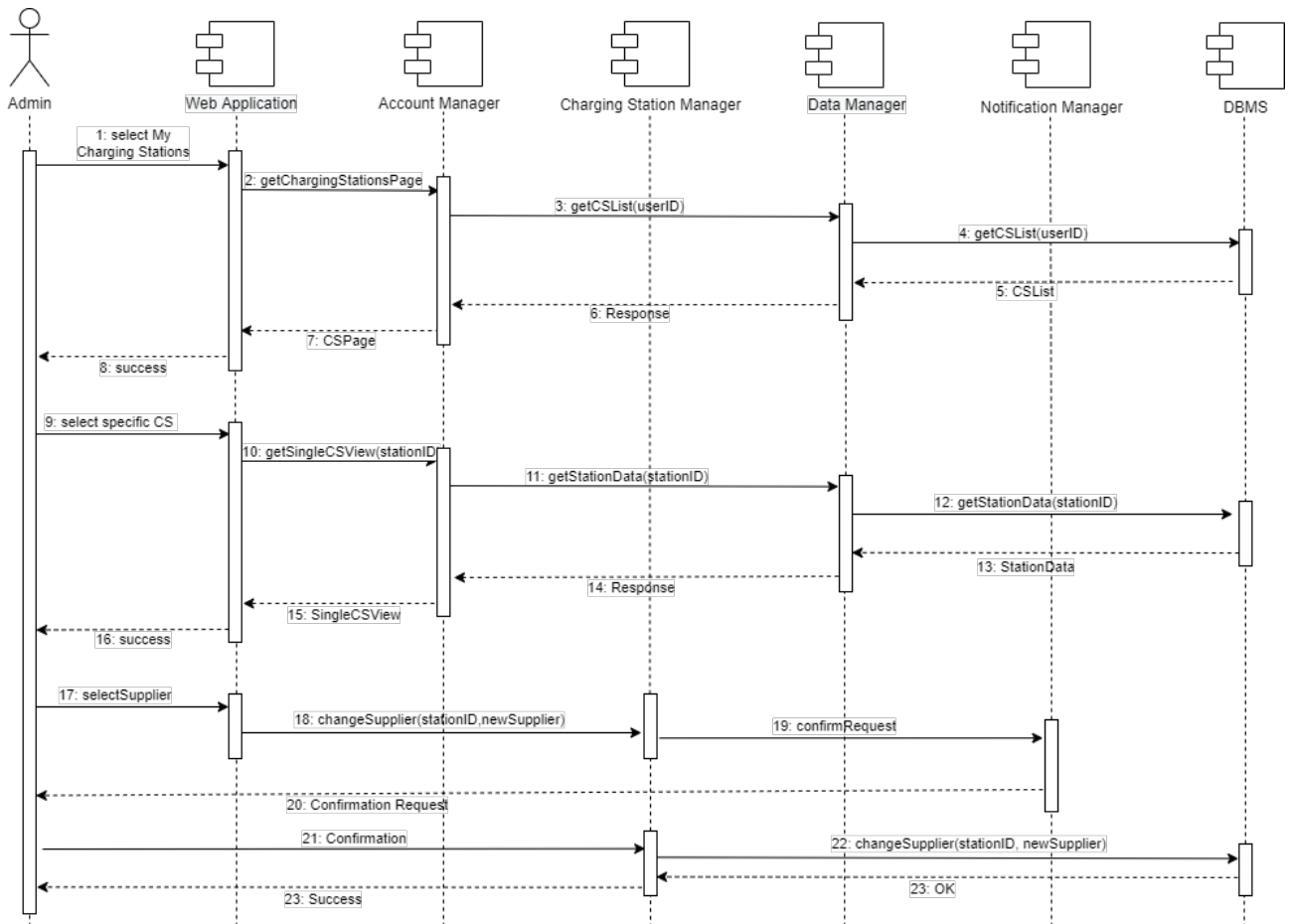
Admin checks energy prices and sources



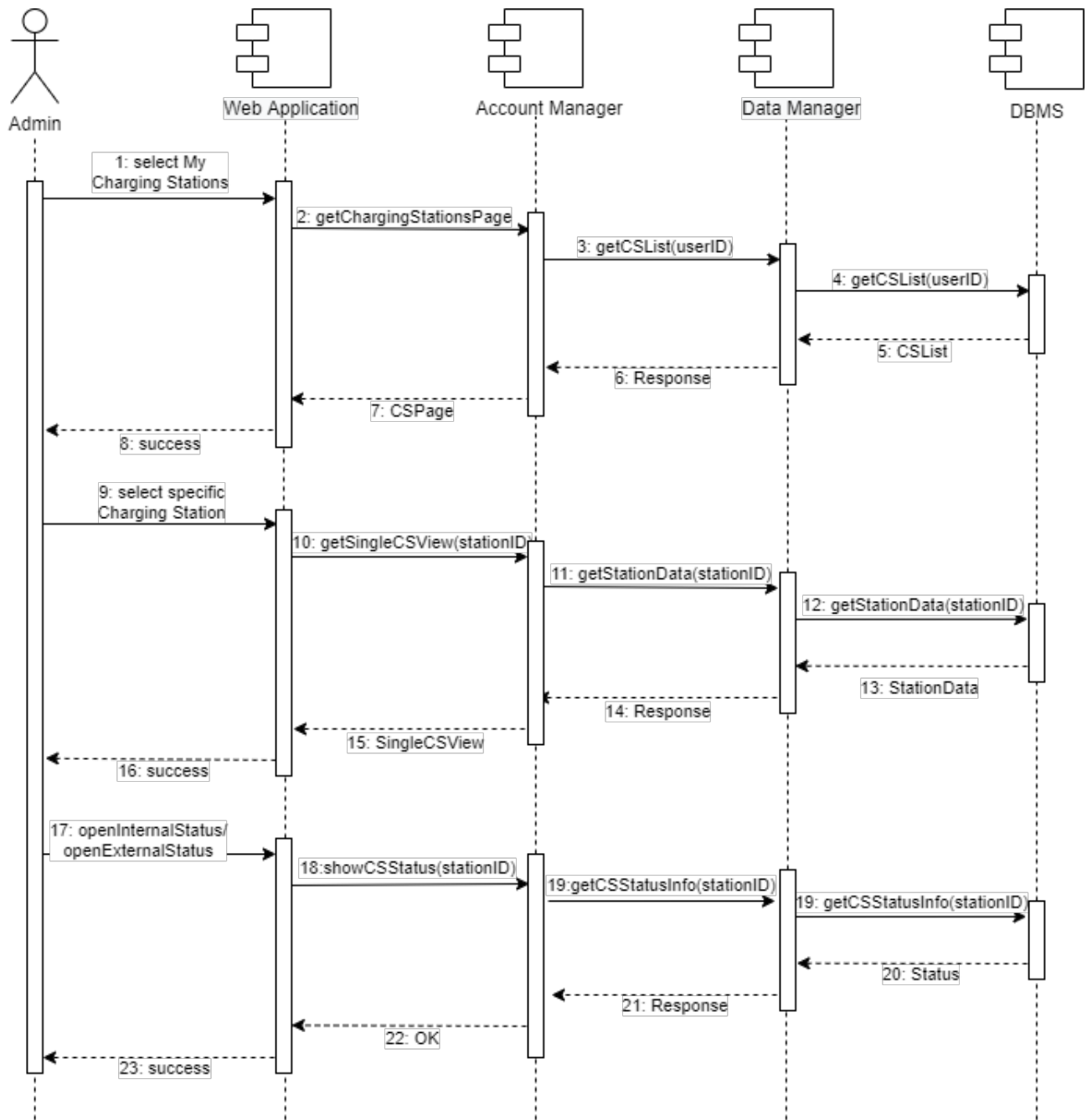
Energy provider selection



Store energy option

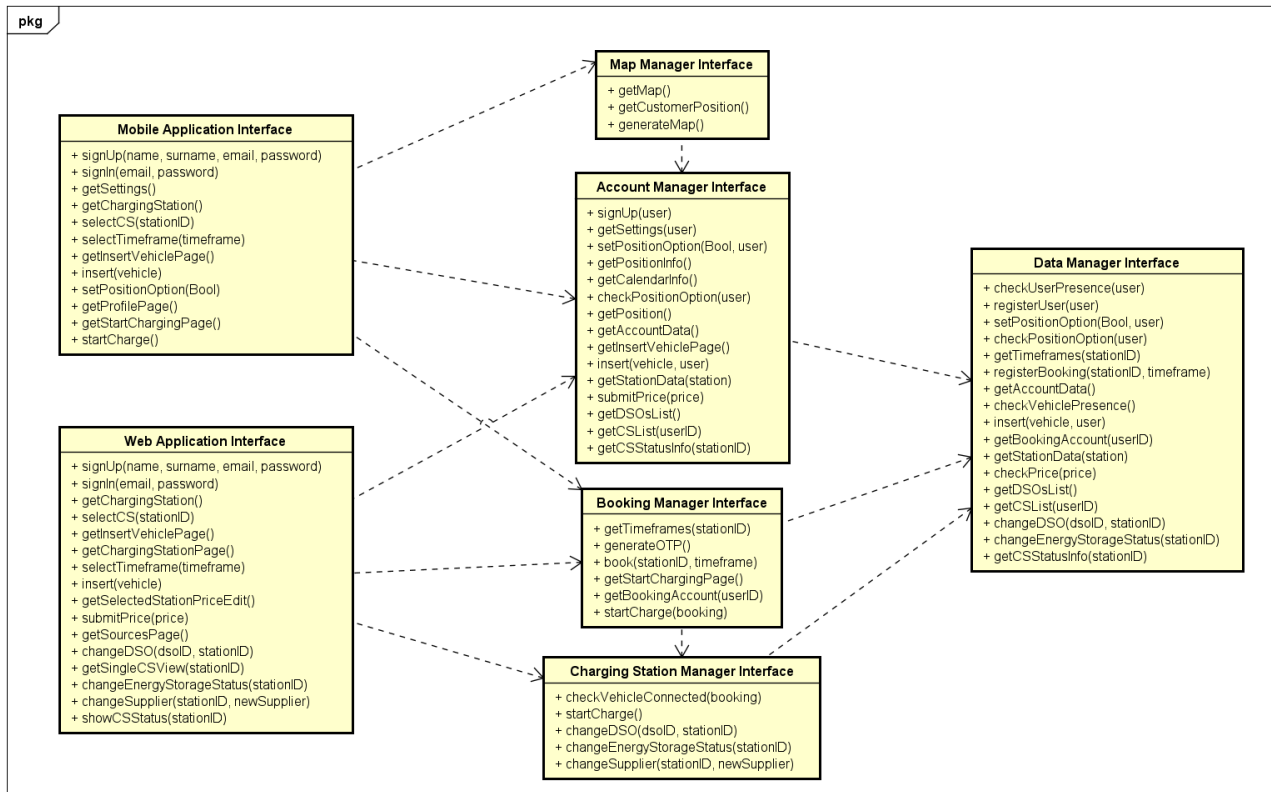


Charging process option



Charging Station status

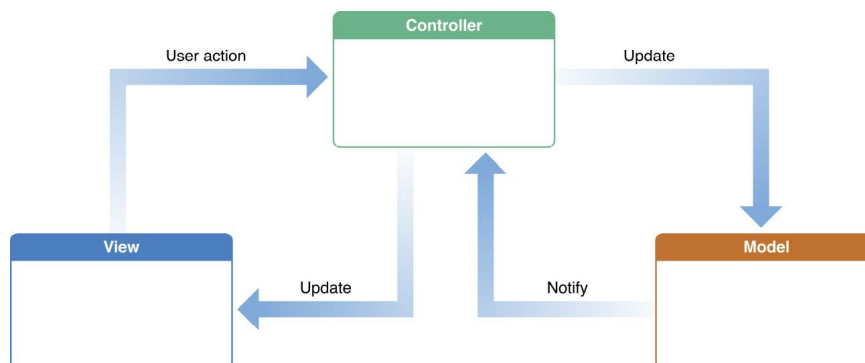
2.5 Component Interfaces



2.6 Selected architectural styles and patterns

2.6.1 Model-view-controller:

MVC (Model-View-Controller) is a software architectural pattern that separates an application into three distinct layers: the model, the view, and the controller. The model represents the data and logic of the application, the view represents the User interface, and the controller mediates the interaction between the model and the view. The system uses MVC pattern both in the web applications and in the mobile app. The main advantage of using MVC is that it allows developers to build applications that are more maintainable, extensible, and testable.



2.6.2 Three Tier architecture:

As already explained in section 2.3, Three-tier architecture is a software design pattern that divides an application into three distinct layers: the presentation layer, the business logic layer, and the data access layer. The presentation layer represents the User interface of the application and is responsible for displaying data to the User and handling User input. The business logic layer contains the core logic of the application and is responsible for performing tasks such as data validation and business rule processing. The data access layer is responsible both for interacting with the database or other data storage systems and for retrieving and storing data for the application. The main advantage of using a three-tier architecture is that it allows a separation of concepts, which makes the application easier to develop, maintain, and scale. It also allows a better reuse of code and improved performance, as the different layers can be optimized and scaled independently.

2.7 Other Design Decisions

2.7.1 External APIs:

The System uses the Google Maps External API to generate the map of the charging station, that is shown in the main view of the application. The data coming from this API are combined with the ones coming from the Calendar External API to calculate where to charge vehicles to avoid wasting of time during the daily schedule.

The Vehicle API (e.g. SmartCar) easily retrieve vehicle data and communicates to the application server the remaining lifespan of the battery. It also knows whether the vehicles are currently charging or not and informs the socket manager about the time left to complete the charge.

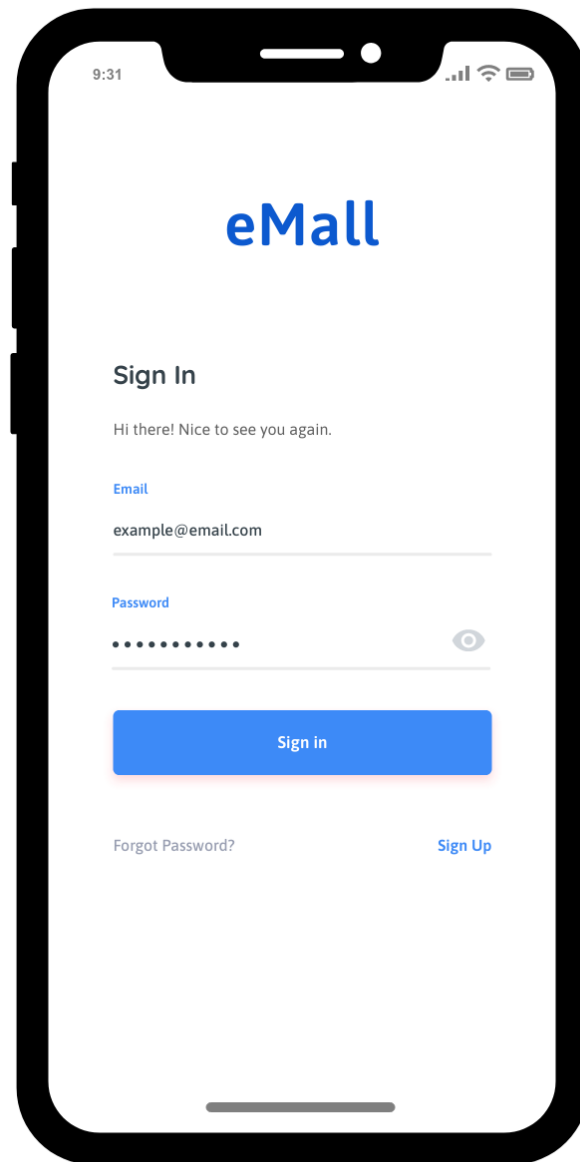
The DSO APIs share with the application server all the information about cost and sources of energy. Through this API DSO are informed about when to start or stop the energy supply of a charging station.

2.7.2 Security Policy:

All User data are encrypted with hash functions to ensure the security and privacy of User accounts and protect against unauthorized access. Encrypted communication between the client and server ensures the security of the connection. When using the web interface, the connection is established using the HTTPS protocol.

3. User Interface Design

- Sign in: Through this interface, a User can log in to the app.



Sign in

- Signup: This interface allows Customers to create a new account.

9:31

Sign Up

Name

Surname

Email

Password

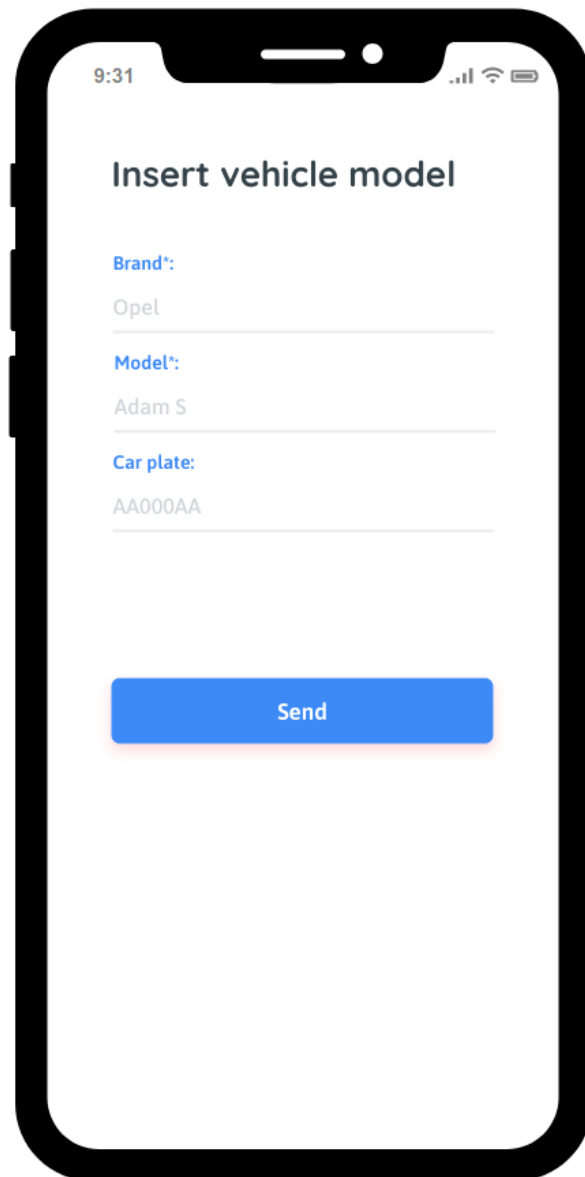
☒ I agree to the [Terms of Services](#) and [Privacy Policy](#).

Continue

Have an Account? [Sign In](#)

Sign up

- Insert vehicle: This interface allows Customers to add a new vehicle to their account. The 'brand' and 'model' fields are required, but the car plate is optional because some vehicles, such as electric bikes, may not have a car plate.



The image shows a mobile application interface for inserting a vehicle model. The screen is titled "Insert vehicle model" and features three input fields: "Brand:" with the text "Opel", "Model:" with the text "Adam S", and "Car plate:" with the text "AA000AA". A blue "Send" button is located at the bottom of the form. The interface is displayed on a smartphone screen with a black border and a white background. The status bar at the top shows the time 9:31, signal strength, and battery level.

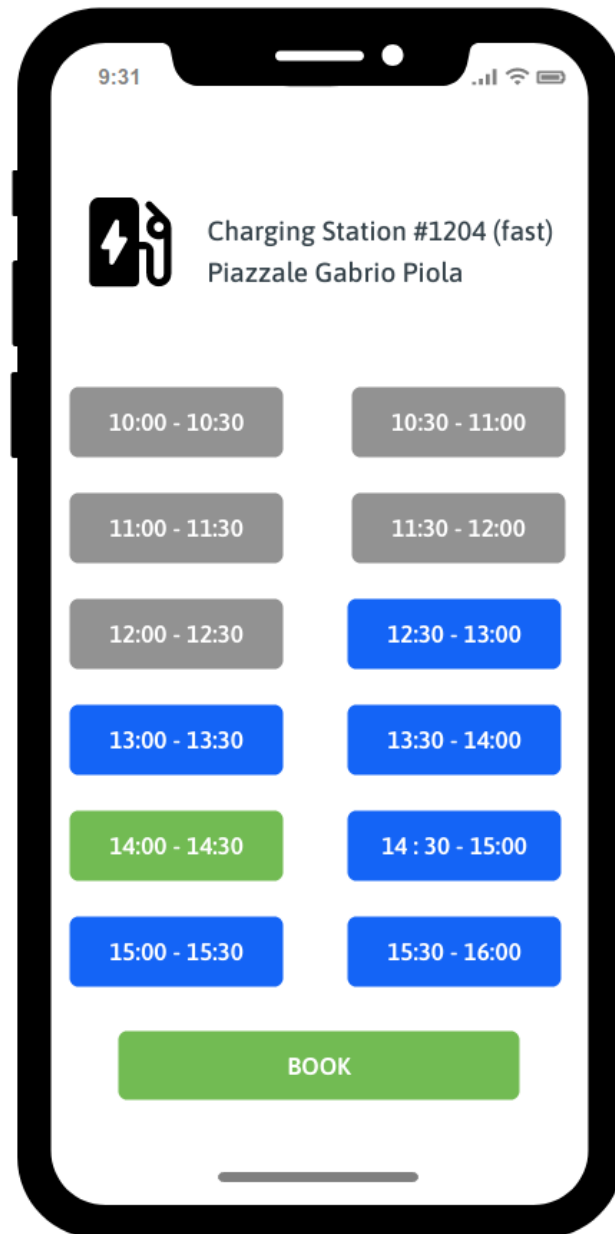
Vehicle's insertion

- Charging Stations map: This interface enables Customers to view the locations of charging stations on a map.



Charging Stations map

- Booking: The interface allows the Customer to book a charge by selecting a time slot. Time slots that are not available are displayed in grey, while selectable time slots are displayed in blue. The selected time slot is displayed in green.



Booking

- Starting charge: When the time of the booked time slot arrives, the Customer can start the charge by inserting the OTP code into the socket, physically connecting it to the car, and pressing the large button on this interface.



Starting charge

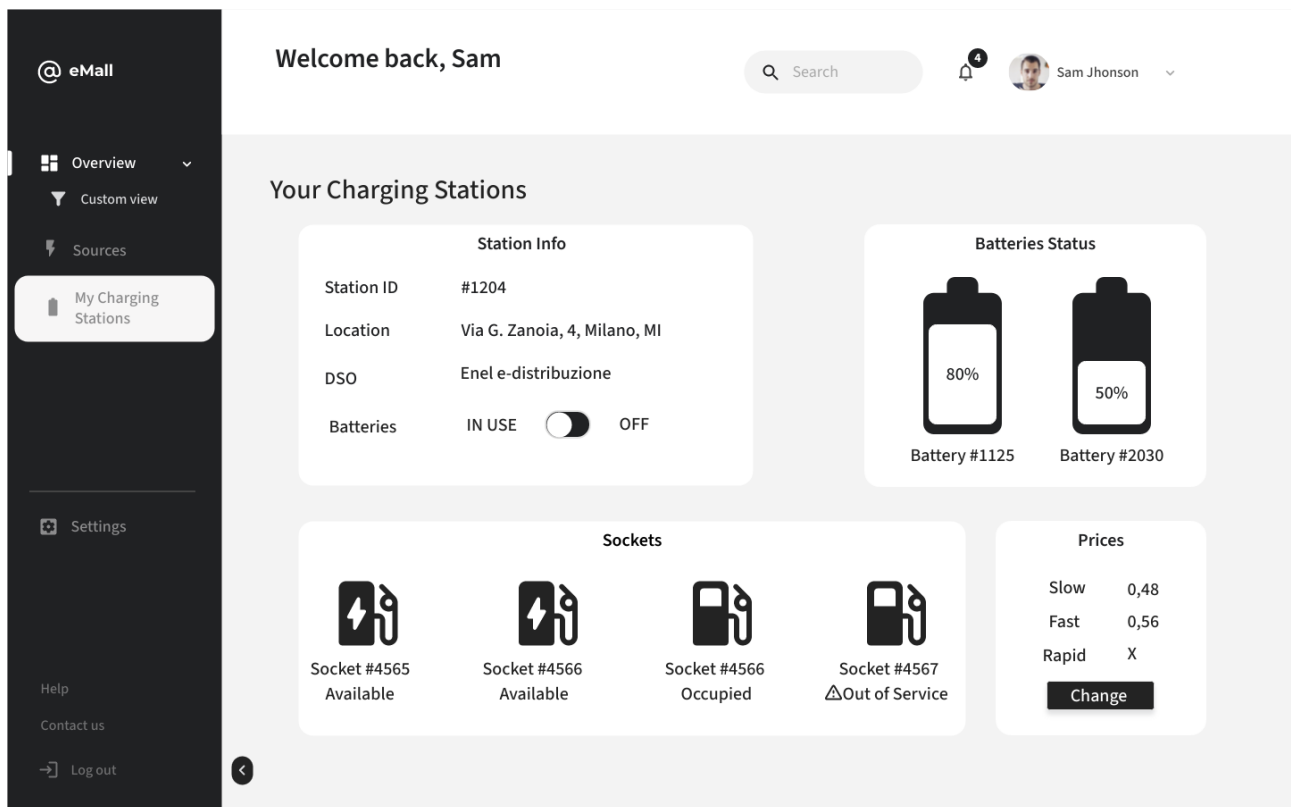
- Charging stations list: Using this interface, an Admin can view the charging stations owned by his company and select one to view more detailed information.

The screenshot shows a web application interface for 'eMail'. The top navigation bar includes a search bar and a user profile for 'Sam Jhonson'. The sidebar on the left contains several menu items: 'Overview', 'Custom view', 'Sources', 'My Charging Stations' (highlighted), 'Settings', 'Help', 'Contact us', and 'Log out'. The main content area is titled 'Your Charging Stations' and features a table with the following data:

Station ID	Location	DSO	
#1204	Via G. Zanoia, 4, Milano, MI	DB Energy	<button>OPEN</button>
#1020	Via F. Juvara, 15, Milano, MI	Enel e-distribuzione	<button>OPEN</button>

Charging Stations page

- Station detail: This interface displays the details of a charging station to the Admin. By clicking on the 'Change' button, an interface for changing the price will be displayed.



Single Charging Station view

- Price modification: This interface enables the Administrator to change the price of the charging station.

The screenshot shows a web application interface for an administrator. On the left is a dark sidebar with navigation links: 'Overview', 'Custom view', 'Sources', 'My Charging Stations' (highlighted), 'Settings', 'Help', 'Contact us', and 'Log out'. The main content area has a header with 'Welcome back, Matthew', a search bar, and a user profile for 'Matthew Parker'. Below the header, a modal titled 'Station #1204 - Edit price' is displayed. It contains a 'Current price' field with the value '0,30 €/KWh' and an 'Insert new price (€/KWh)' input field. A 'Save' button is located at the bottom right of the modal.

Station #1204 - Edit price

Current price

0,30 €/KWh

Insert new price (€/KWh)

Save

Prices modification

- Sources page: Using this interface, an Administrator can view the prices of the DSOs and their energy sources.

Welcome back, Sam

Search

4

Sam Jhonson

Distribution System Operators

DSO Name	Solar	Hydro	Wind	Fossil	Energy Cost (€/kWh)	
Enel e-distribuzione	20%	20%	35%	25%	0,59	<button>SELECT AS PROVIDER</button>
DB Energy	15%	40%	10%	35%	0,66	<button>SELECT AS PROVIDER</button>

Settings

Help

Contact us

Log out

Sources page

4. Requirements Traceability

The following table explains how the functional requirements defined in RASD are fulfilled by the system's components.

	Web Application	Mobile Application	Account Manager	Notification Manager	Booking Manager	Maps Manager	CS Manager	Suggestion Manager	Data Manager	BDMS
FR1	✓	✓	✓						✓	✓
FR2	✓	✓	✓						✓	✓
FR3	✓	✓	✓						✓	✓
FR4	✓	✓	✓						✓	✓
FR5		✓	✓			✓			✓	✓
FR6	✓	✓					✓		✓	✓
FR7	✓	✓	✓			✓			✓	✓
FR8	✓	✓			✓				✓	✓
FR9	✓	✓			✓				✓	✓
FR10		✓	✓						✓	✓
FR11		✓		✓						
FR12		✓		✓		✓		✓	✓	✓
FR13		✓		✓			✓			
FR14	✓	✓	✓						✓	✓
FR15	✓	✓	✓						✓	✓
FR16	✓	✓	✓						✓	✓
FR17	✓		✓				✓		✓	✓
FR18	✓		✓				✓		✓	✓
FR19	✓		✓				✓		✓	✓
FR20	✓		✓				✓		✓	✓
FR21	✓		✓						✓	✓
FR22	✓		✓						✓	✓

5. Implementation, Integration and Test Plan

The project should be divided as follows:

- Client: it includes both web browser and mobile application, with different functionalities;
- Application server: contains all the controller;
- Database: where all the data are stored;
- External Services: that handle some fundamental features for the application.

We will be implementing these elements using a bottom-up approach to avoid creating incomplete structures that would be harder to implement and test. The module for the application server is the most important because it is crucial for the S2B development and also tends to be more difficult to test. The table below lists the functionalities outlined in the RASD document and indicates the level of importance for the Customer and the difficulty of implementing each one.

Functionality	Importance for Customer	Complexity
Sign Up and Sign In	High	Low
Booking	High	Low
Charging	High	Medium
Battery Monitoring	Low	High
Change of Provider	Medium	Medium
Switch between Suppliers	Low	Low

Sign Up and Sign In: these two functionalities are strictly related, and both are handled by the Account Manager, so they could be tested together. Admins are already registered in the application, so we add them manually in the Database. These already present account could be used for the Login Test. This test is needed to check if accounts are correctly stored in the Database and if there are problems while showing User data. Login must be tested to check if registered Users' data are correctly retrieved from the Database.

An example could be that when an Admin requires the Charging Stations Page, he must get only the Charging Stations info related to his account.

Booking: this functionality is managed by the Booking Manager. Tests must check if any error occurs while storing booking info in the Database. In fact, there can't be different bookings for the same time slot and socket or multiple bookings for the same vehicle and time slot.

Charging: it is handled by the Charging Station Manager. Test must be implemented to verify that a charge can start only if a Customer inserts the correct UserID and types the right OTP.

They must check even if the notification is correctly sent as soon as the charge finishes. The integration between Charging Station manager and Data Manager must be tested too.

Battery Monitoring: this special function is managed by Suggestion Manager and Notification Manager and is available only for Customers. Tests are necessary to check that when the vehicle's charge is low a notification is correctly received. If a Customer has shared the position and the calendar, there must be tests that check the correctness of the itinerary proposed by Suggestion Manager.

Change of Provider and Switch Between Supplier: both the functionalities are handled by the Charging Station Manager. In this case test must verify if after every change in the charging station data are correctly updated in the Database and if information is correctly shared with DSOs.

6. Effort Spent

Task	Time Spent (h)
Discussion about the Assignment	1
Introduction	3
Overview and component view	5
Deployment view and runtime view	10
Interface and architectures	5
User interface design	6
Requirements traceability	1
Implementation, Integration and Test Plan	3
Revision	5
Total	39