DD assignment 1

Filippo Cinfrignini, Davide Esposito

January 2023

Contents

1 Introduction

1.1 Purpose

The purpose of this design document is to support the development team in the realisation of the system to be. It covers all the information related to the development of the system to facilitate the developers' work. This document provides an overall description of the adopted system architecture with the reasons for such a choice. It contains also a breakdown of the various components and a description of their interactions. The document describes the implementation, integration and testing plans which are defined keeping into account priority, required effort, and impact of the components on the stakeholders.

1.2 Scope

The eMall software (e-Mobility for All) is a system that helps limit the carbon footprint caused by our urban and sub-urban mobility needs. The system's objective is to provide services to drivers and CPO operators that facilitate the charging of electric vehicles and the provision of energy to charging stations.
The system to be developed will allow owners of electric vehicles to book charging sockets available so that the owners can charge their vehicles. The system will also offer features to monitor the charging processes.
The system will also allow those responsible for charging stations to check the stations' status. And will also make available the features to manage the station's energy sources. This document will further expand on the RASD document inserted in the Reference Document section.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Acronyms

Acronyms	Description
PSP	Payment Service Provider
CPO	Charging Point Operators
DSO	Distribution System Operators
CPMS	Charging Point Management System

1.3.2 Abbreviations

Abbreviations	Description
Fn	the n-th feature

1.4 Revision history

- V1.1 - January 8th 2023

1.5 Reference Documents and Website Documents

[1] Assignment RDD AY 2022-2023_v3.pdf.

[2] Lectures slides of the Software Engineering II course on https://beep.metid.polimi.it/

[3] eMobility: https://solidstudio.io/

[4] eMobility / OCPI: https://evroaming.org/

[5] eMobilhttps://www.evconnect.com/

1.6 Document Structure

The **first chapter** is the introduction. It contains a brief description of the purpose and the scope of the Design Document. It contains also information useful to the reader such as definitions, acronyms and abbreviations. It provides documents linked to this Design Document in the subsection reference documents. It concludes with an overview of the document's structure.

The **second chapter** includes a detailed description of the system's architecture, including the high-level view of the elements, the software components of eMall, a description through runtime diagrams of various functionalities of the system and an in-depth explanation of the architectural pattern used.

In the **third chapter** are delivered the mockups of the application user interfaces, along with the links between them to help understand the flow between them.

In the **fourth chapter** are stated the connections between the requirements defined in the RASD and the components described in the first chapter. This is used as proof that the design decisions have been taken concerning the requirements, and therefore that the designed system can fulfil the goals.

The **fifth chapter** describes the process of implementation, integration and testing of the software. It contains the plan to which developers have to stick to produce the correct system properly.

2 Architectural Design

2.1 Overview

In this section is given an overview of the architectural elements that compose the system, their interaction and a description of the replication mechanism chosen for the system in order to make it distributed.

2.1.1 High level view

The eMall system-to-be is a distributed system with a three-tier architecture (Figure 1). It includes the Presentation layer for formatting the information to be shown, the Application layer which encapsulates the business logic of the application and the Data layer that manages the access to the stored data. Each layer can communicate only with the adjacent one, this means that the data layer and the presentation layer are never going to communicate directly.

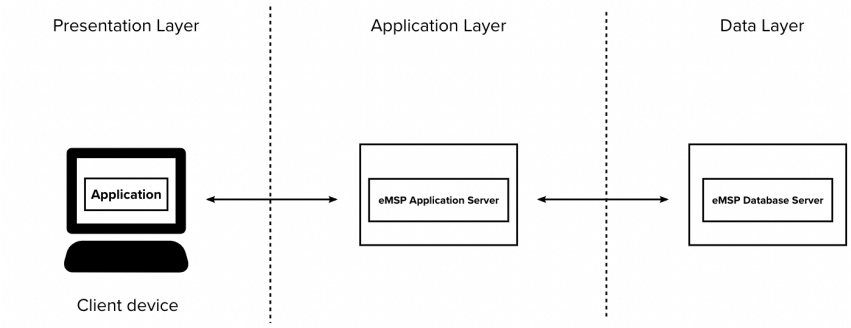

Figure 1: High level system architecture

System clients use a smartphone application, represented in Figure 2 by the presentation level, which communicates with the system application server. The latter is the main core of the entire system, it receives requests from clients, processes the information and returns the correct results.

To do this, the server must be able to communicate with all the CPOs within the eMall network. In particular, each CPO has its own IT infrastructure administered through the so-called Charge Point Management System (CPMS), which, through a peer-to-peer roaming architecture with OCPI protocol, will communicate with the eMall system. Furthermore, the system also manages communications with the various payment service providers (PSPs), i.e. private entities external to the eMall system for the management of the various payment transactions made by customers. This communication takes place through the different proprietary APIs made available by the different PSPs.

Finally, to provide the required feature set, the application server must be connected to a database from which to retrieve eMall user information.

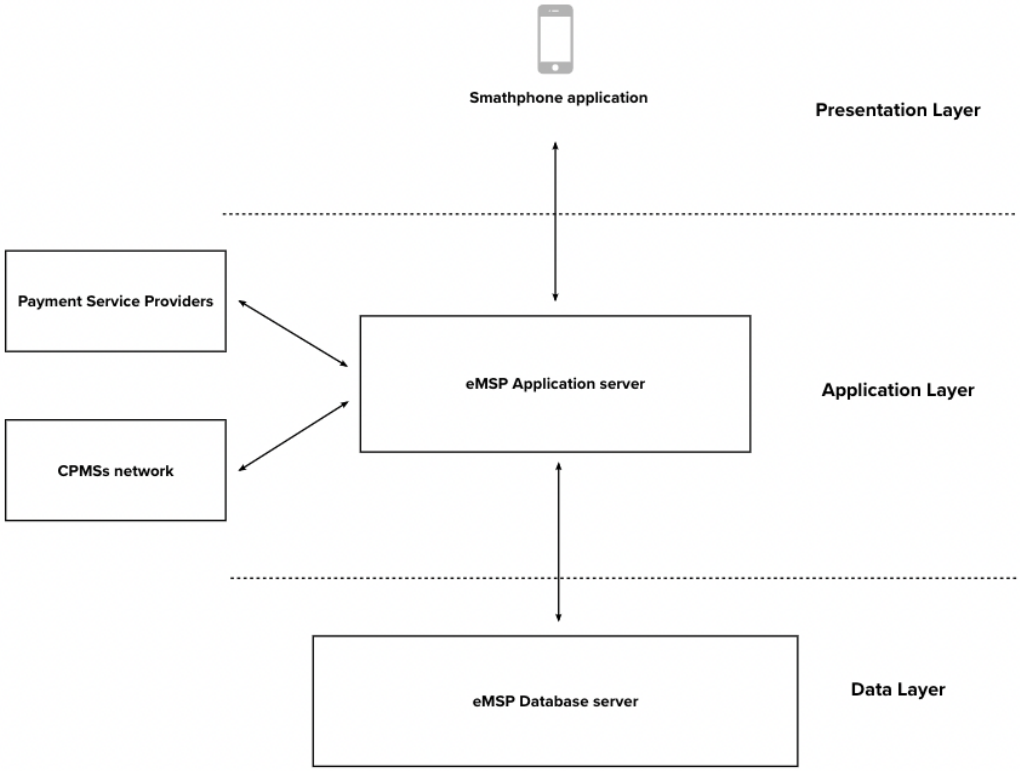

Figure 2: High level system architecture 2

2.1.2 Distributed view

Most of the elements represented in Figure 2 are going to be replicated in order to fulfill the performance and availability requirements. Figure 3 highlights the distributed elements of the architecture.

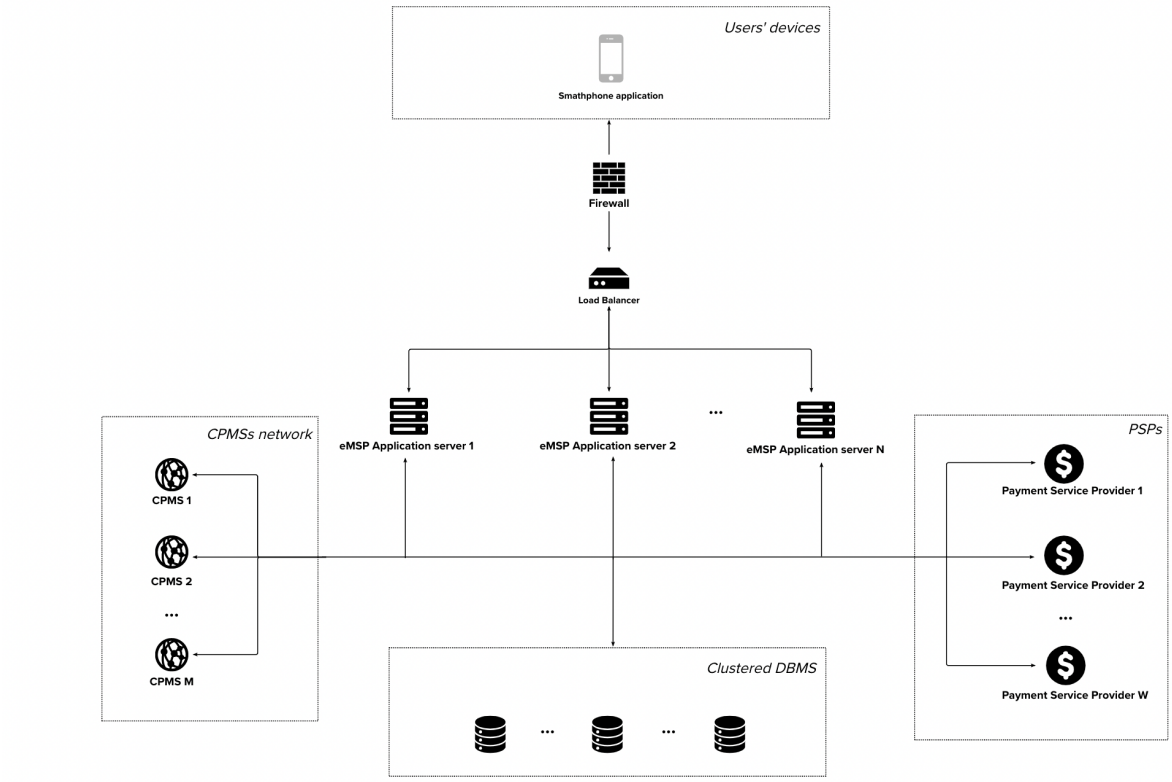

Figure 3: Distributed architecture

Load Balancer

The distributed architecture exploits a load balancer to dispatch the client requests to application servers. All the requests coming from the mobile application are captured by the load balancer which redirects the request to the more suitable application server available to handle it, i.e. the server which minimizes the response time.

eMSP Application servers

Regarding the DBMS, it should be clustered. Any service that provides solid clustered DBMS can be adopted for eMall, as long as the consistency is guaranteed and the performances are sufficient to fulfill the queries in a reasonable time. Finally, all the packets incoming inside the eMall network are filtered by a firewall before being sent to the load balancer.

Clustered DBMS

Regarding the DBMS, it should be clustered. Any service that provides solid clustered DBMS can be adopted for eMall, as long as the consistency is guaranteed and the performances are sufficient to fulfill the queries in a reasonable time. Finally, all the packets incoming inside the eMall network are filtered by a firewall before being sent to the load balancer.

2.2 Component view

In Figure 4 is represented the component diagram of the system. All the components in yellow are elements of the eMSP application server. The CPMS component is colored in blue and it represents the element not belonging to the application server, yet still being part of the application layer. The element colored in red, instead, represents the components provided directly to users to interact with the system, i.e. the one that belong to the Presentation tier. The only element colored in green is the DBMS, which is the only one belonging to the data tier. This color criteria is going to be used also in other diagrams of the document for clarity.

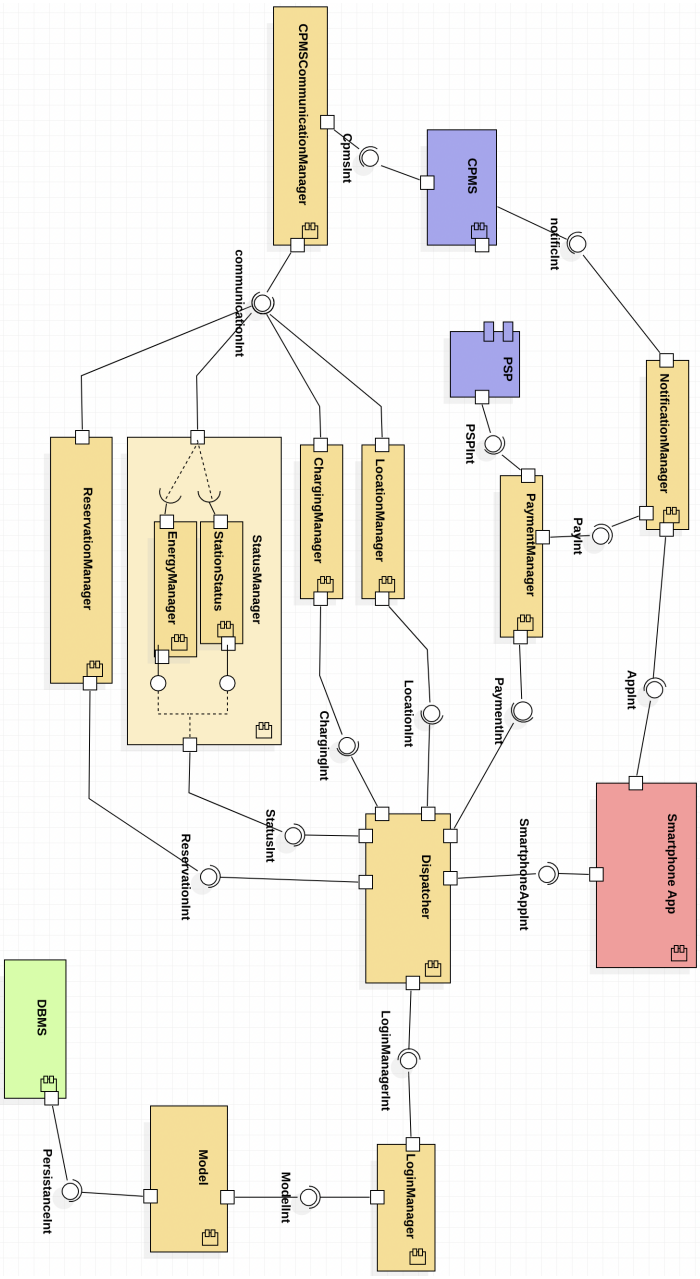

Figure 4: Component diagram

- The **Smartphone App** is the application for mobile devices provided to users for accessing the system via smartphone/tablet. It is required to be installed on compatible devices and its communication with the system involves directly the application server, without passing through other intermediary components, hence interacting directly with the Dispatcher.
- The **DBMS** component allows to create and manipulate the database that is necessary for the functioning of eMall. The db contains all the data related to the accounts of the users of the system and is the one resolving the queries received from the model component with the requested data.
- The **Dispatcher** receives all the requests from the Smartphone App component. Next, it forwards them to the competent component and, once the responses are ready, the dispatcher itself returns the answers to the clients.
- The **LoginManager** component allows eMall users to enter their account if the credentials they provide are correct. In general it has the role of granting the privacy of the customers' data by denying the access to unauthorized users.
- The **Model** component is used to recover and store data that are part of the system's persistent state (which in our case is stored on a database). Moreover, having seen that the only information kept in the database are those associated with user accounts and that (for simplicity) the registration and saving of accounts operations have not been managed, at this stage the model will be used only to retrieve user information.
- **ReservationManager** is the component designated, as the name suggests, to manage all the tasks related to the reservation process. In particular, it is interested in managing the requests to be sent to the CPMSCommunicationManager component for booking management. It's also the component involved in creating the verification code associated with reservations.
- The **VisitManager** component is responsible for the management of the charging processes carried out by the driver. In fact, it takes care of forwarding requests to start and finish the charging process.
- The **LocationManager** component is responsible for requesting the information (in particular associated with the location) of the stations. This information is essential so that the user can view the stations within the map and consult the associated information.
- **StatusManager** is the component designated, as the name suggests, to manage all the information related to the stations' status. In particular it manage the stations' energy status via the sub-component **EnergyManager** and the station's external status via the sub-component **StationStatus**.
- The **PaymentManager** component takes care directly to manage the payments from the drivers, interfacing directly with the Paymente Service Providers. In particular, it verifies the correctness of the payment methods and starts the transactions with the PSPs.
- The **CPMSCommunicationManager** it is one of the most important components of the whole system. It manages communication with CPMSs by forwarding the different requests received from the other components. Once a response has been obtained from the CPMS, it is responsible for forwarding it again to those components that made the request.
- **NotificationManager** manages the notifications it receives from the CPMS and forwards them to the other components of the system, in particular to the smartphone App component (for example when a booking expires or when the charging process ends).

2.3 Deployment view

In this chapter is described the deployment view for eMall (Figure 5). This view describes the execution environment of the system, together with the geographical distribution of the hardware components that executes the software composing the application. In the following graph the elements are highlighted according to colors defined in chapter 2.2.

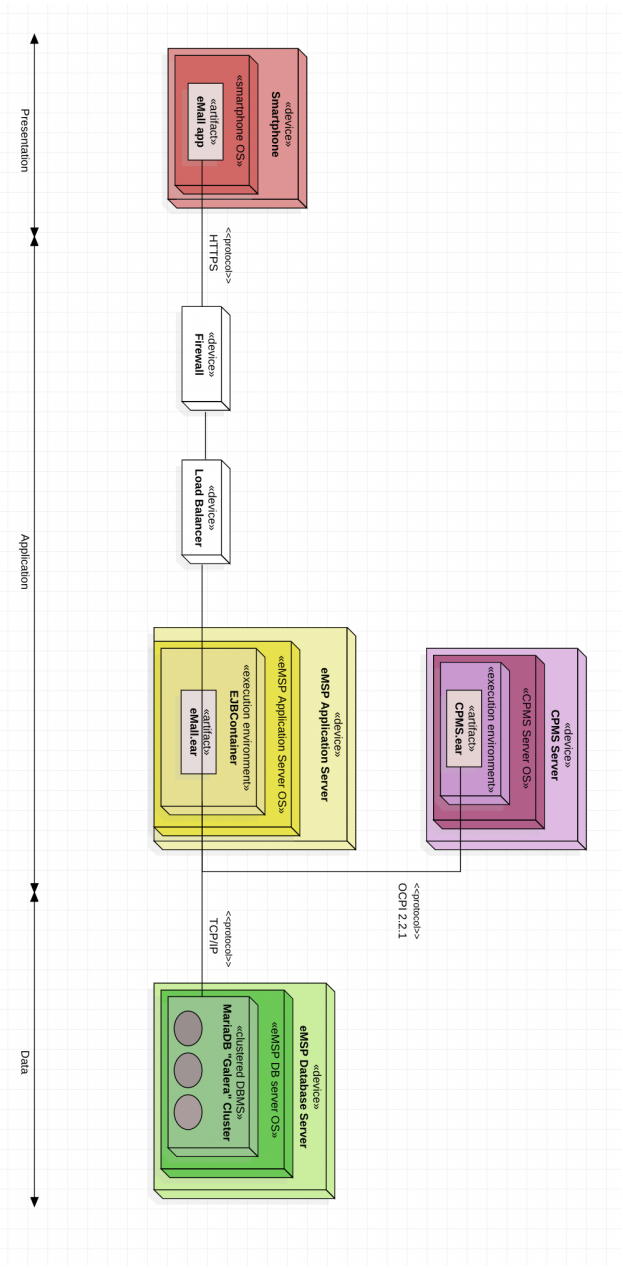

Figure 5: Deployment view

Further details about the elements in the graph are provided in the following.

Smartphone

The smartphone is a client on which it is possible to install the eMall smartphone application. The software can interact with the system via HTTPS requests, which are forwarded directly to the application server.

Load Balancer

A load balancer is a device which performs the load balancing. This is the process of distributing a set of requests over a group of resources such as application servers or database servers, with the intent of increasing performance, scalability and soundness of the system.

Firewall

The firewall is a device that monitors the packets incoming to the system, if a packet is potentially dangerous it is not forwarder. It is placed before the load balancer, in this way the only packets that enter the network are considered safe.

eMSP Application Server

The application server receives all the requests sent from smartphones which have the eMall application installed. It is the most important element of the business tier, in fact it is designated for data elaboration and management. It is the only element of the architecture that can communicate with the database via TCP/IP and with the CPMSs network (represented in Figure 5 by a single CPMS component) via the OCPI protocol (2.2.1 version). In addition, communication between the application server and external services (such as PSPs not shown in the figure) takes place through proprietary APIs provided by the same services.

CPMS Server

As we have seen, the emall system interacts with the different CPOs through its IT infrastructure (CPMS), which, in addition to being equipped with its own internal structure (servers, databases and connections to individual charging points), it provides an interface that allows our system to communicates via OCPI protocol.

eMSP Database Server

The database server represents a cluster of databases managed via the MariaDB Galera Cluster. This will guarantee virtually synchronous data replication on all the members of the cluster. Therefore, even if the system can be slowed down by the complexity of the replication mechanism, it will grant a very high availability and always provide up-to-date information to the application server. In this way no errors are possible due, eventually, to the unalignment between master and slaves typical of semi-sync/async clusters.

2.4 Runtime view

In this chapter are presented all the runtime views associated with the use cases described in the RASD relative to eMall. Runtime views show the interactions between the various components to carry out the functionalities offered by eMall.

[RV - 1] User-driver login to eMall

The following sequence diagram (Figure 6) shows the login procedure followed by a Driver using the mobile application.
When a driver enters its credentials, the application sends them to the Dispatcher module of the application server. From there the request is forwarded to the Login Manager module which compares the credentials provided to him with those retrieved from the system database. It is important to note that, if the login credentials are correct, the Dispatcher submits a request to the LocationManager module to retrieve the station locations through the CPMS to be delivered to the application in order to build the map.

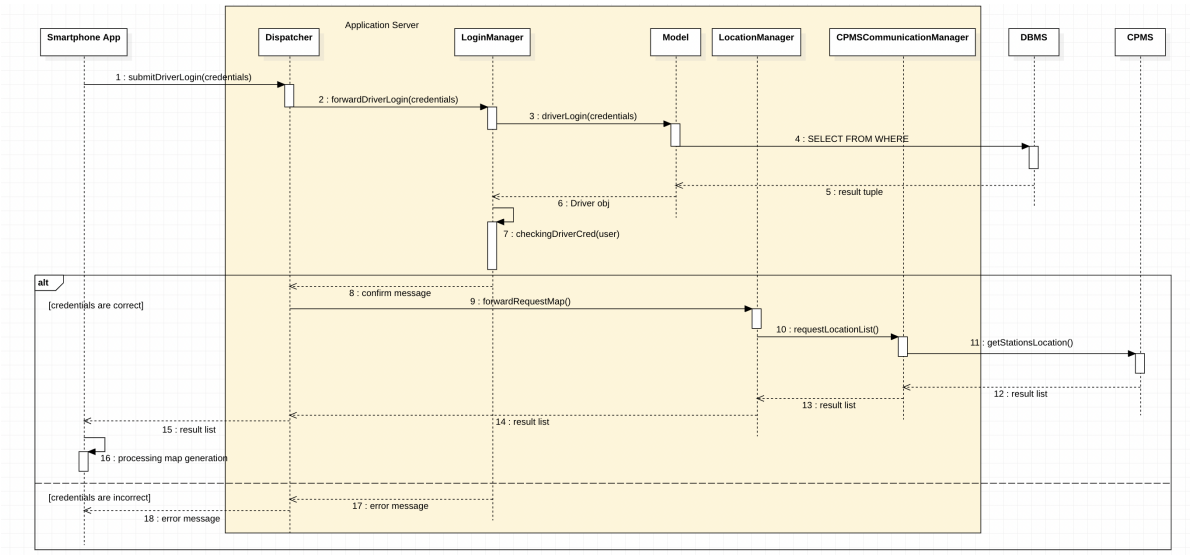

Figure 6: Driver login runtime view

[RV - 2] Driver consults the information of a station

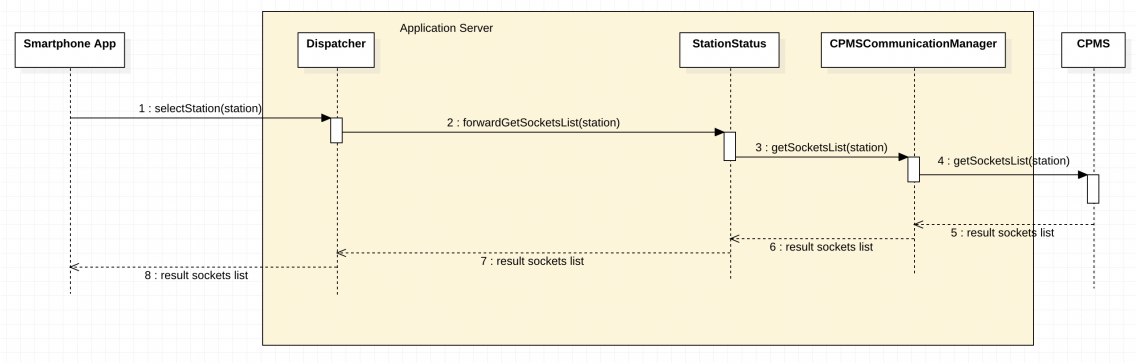

Figure 7: Get station information runtime view

When the driver selects a station from the map, the system sends a request to the CPMS, obtaining a list of socket objects that contain the information to the sockets available for that station (price, availability, type of socket, etc.). The application then extracts the information and shows it to the user via the application.

[RV - 3] Driver books a charge

The runtime view associated with booking a charging outlet (Figure 8) starts when the user, after selecting a station from the map, chooses an available charging outlet from the list. From the moment of the choice the system shows him the page for the insertion of the payment method. Once the correctness of the data entered has been verified (via a request to the payment service provider), the system forwards a request to the CPMS to obtain the booking information that will be subsequently confirmed by the user.

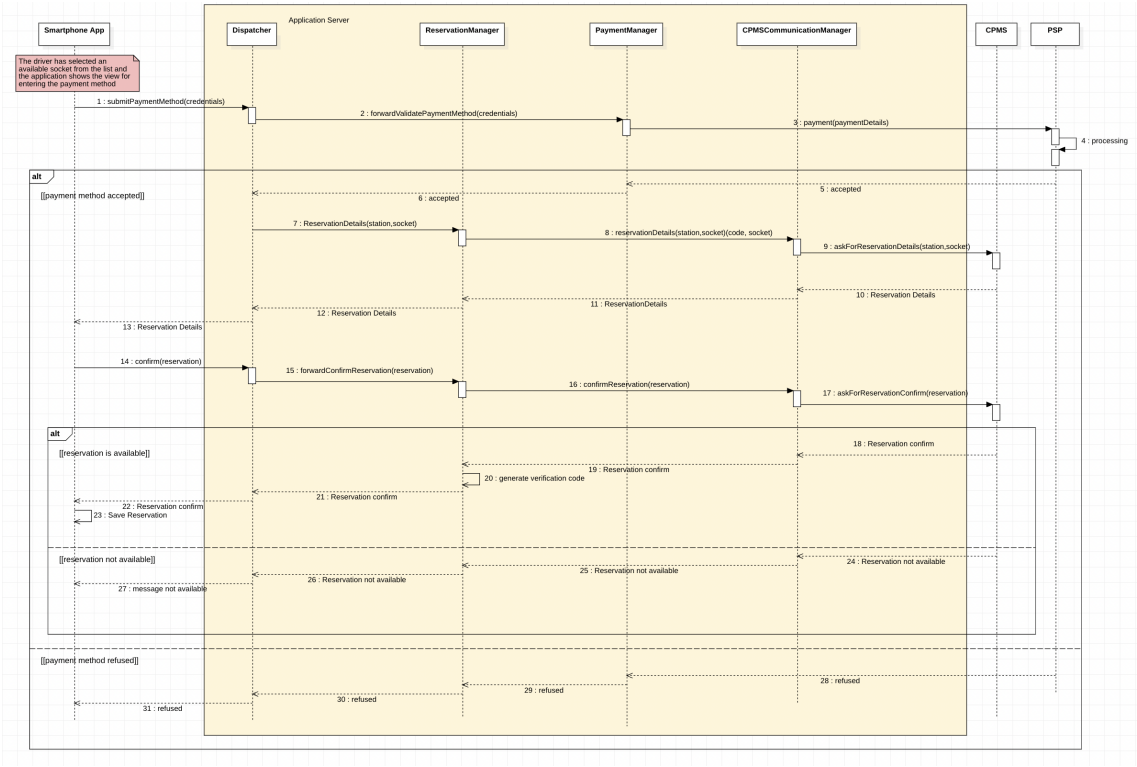

Figure 8: Book a charge runtime view

[RV - 4] Driver consult the details of a booking

Once a booking has been confirmed by the system, the application keeps the information associated with it locally, avoiding having to make the request to the DBMS every time. The whole operation requires no interaction between the mobile application and the different components of the eMall system. In fact for the visualization of such information the application will supply the various methods for the presentation of the 'MyReservations' view and the acquisition of the information associated with the reservation.

[RV - 5] Driver cancels a reservation

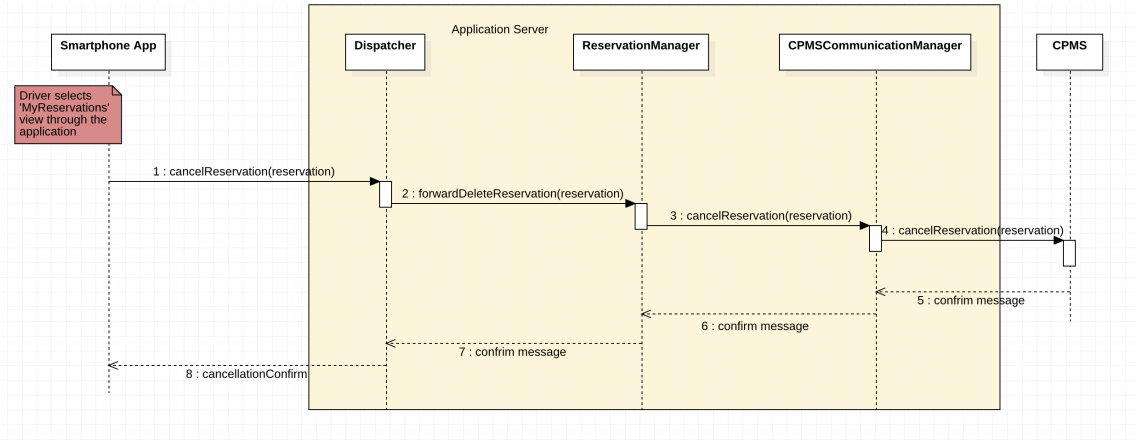

Figure 9: Cancel a reservation runtime view

[RV - 6] Driver starts a charging process in a station

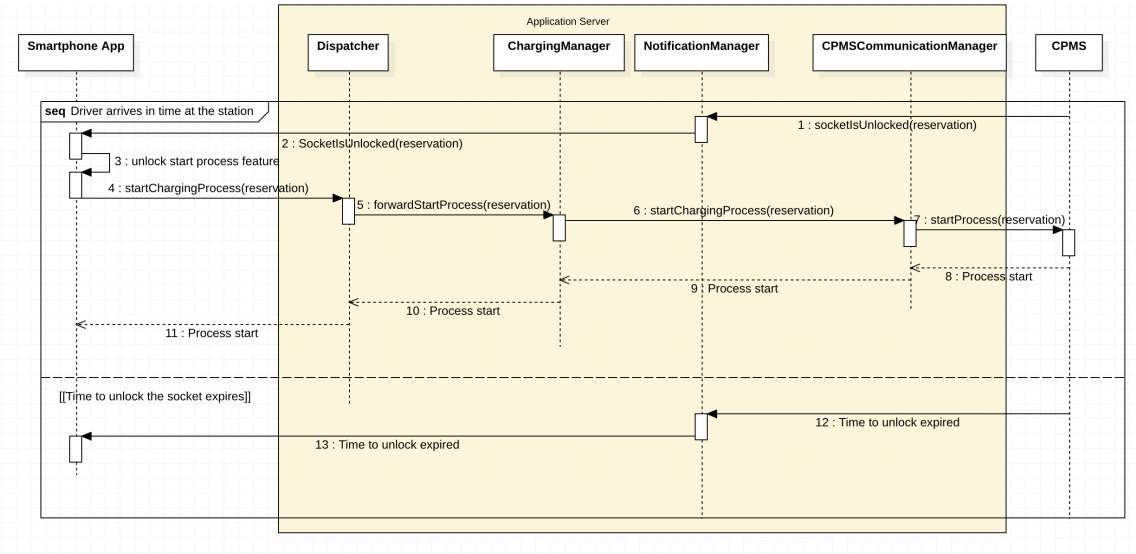

Figure 10: Start charging process runtime view

[RV - 7] Driver ends an ongoing charging process in a station

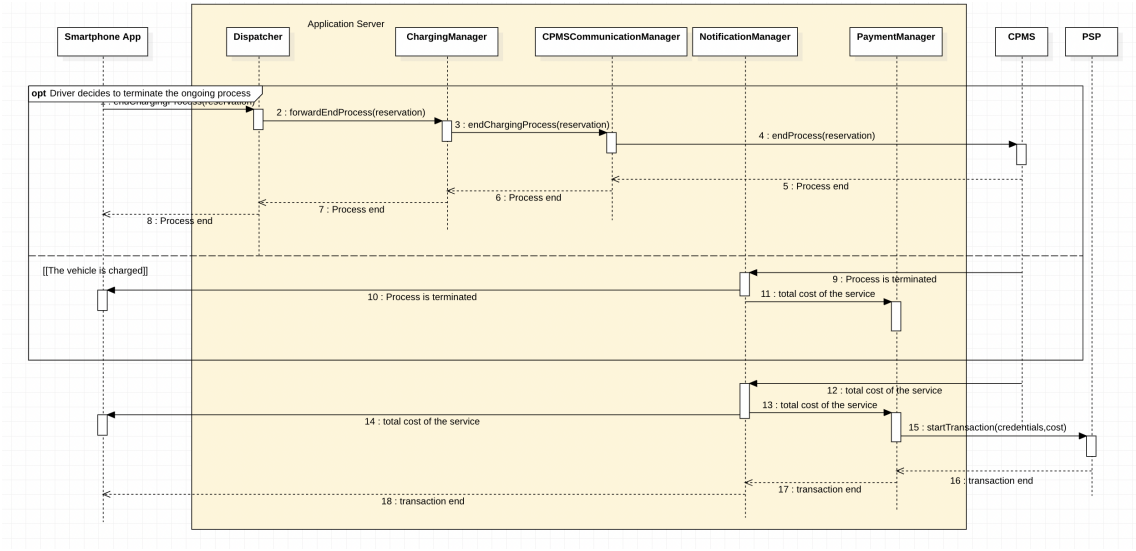

Figure 11: Stop charging process runtime view

[RV - 8] User-operator login to eMall

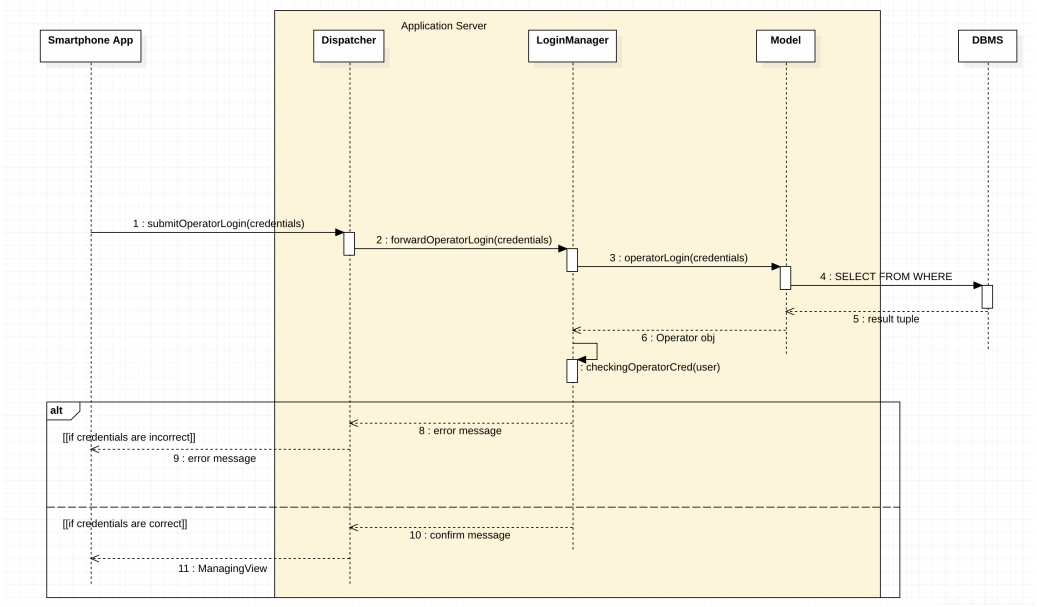

Figure 12: Operator login runtime view

[RV - 9] Operator consults the energy price of a DSO

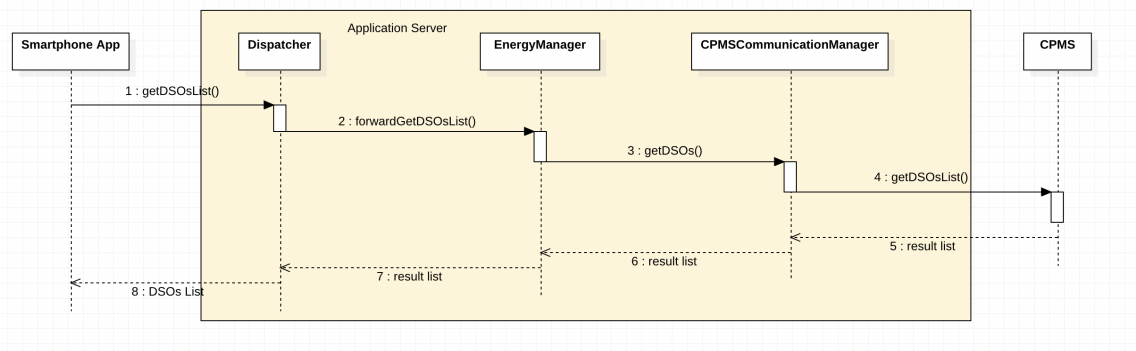

Figure 13: Get energy price from a DSO runtime view

[RV - 10] Operator accesses the management panel of a station

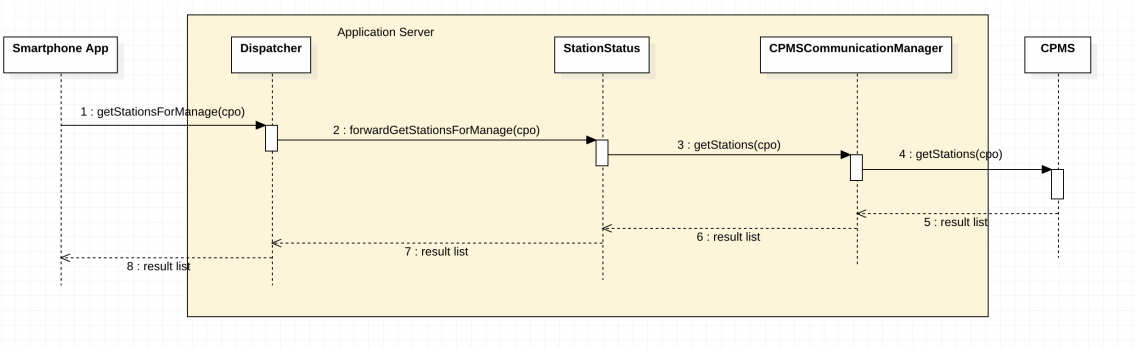

Figure 14: Select charging station runtime view

[RV - 11] Operator checks a socket status

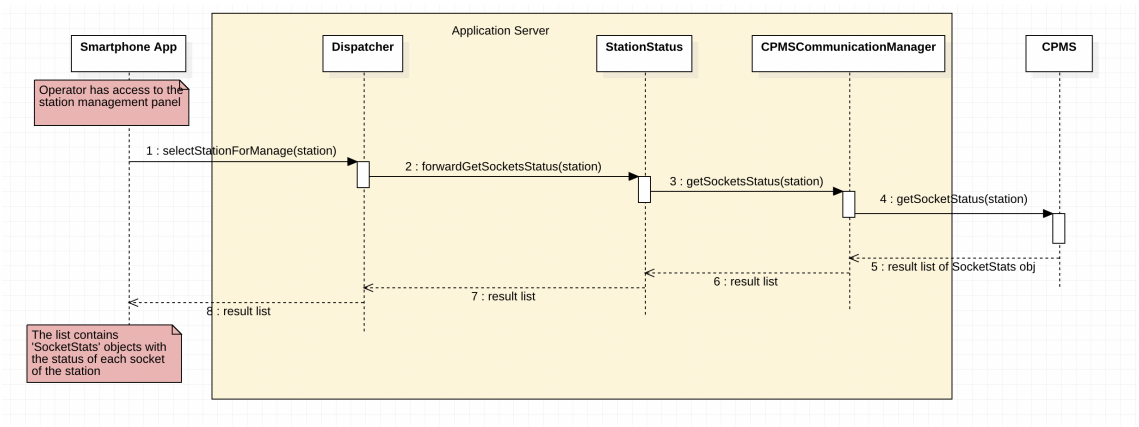

Figure 15: Get socket status runtime view

[RV - 12] Operator checks the batteries status of a station

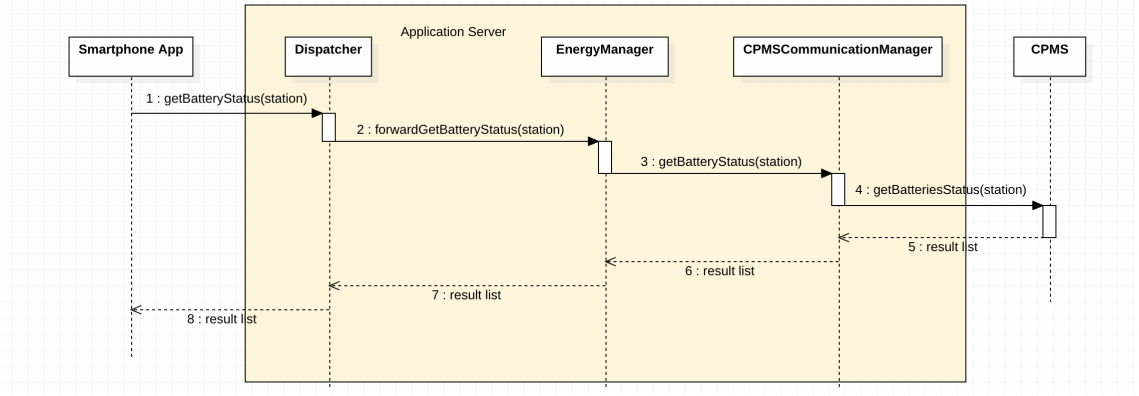

Figure 16: Get socket status runtime view

[RV - 13] Operator changes the energy source settings of a station

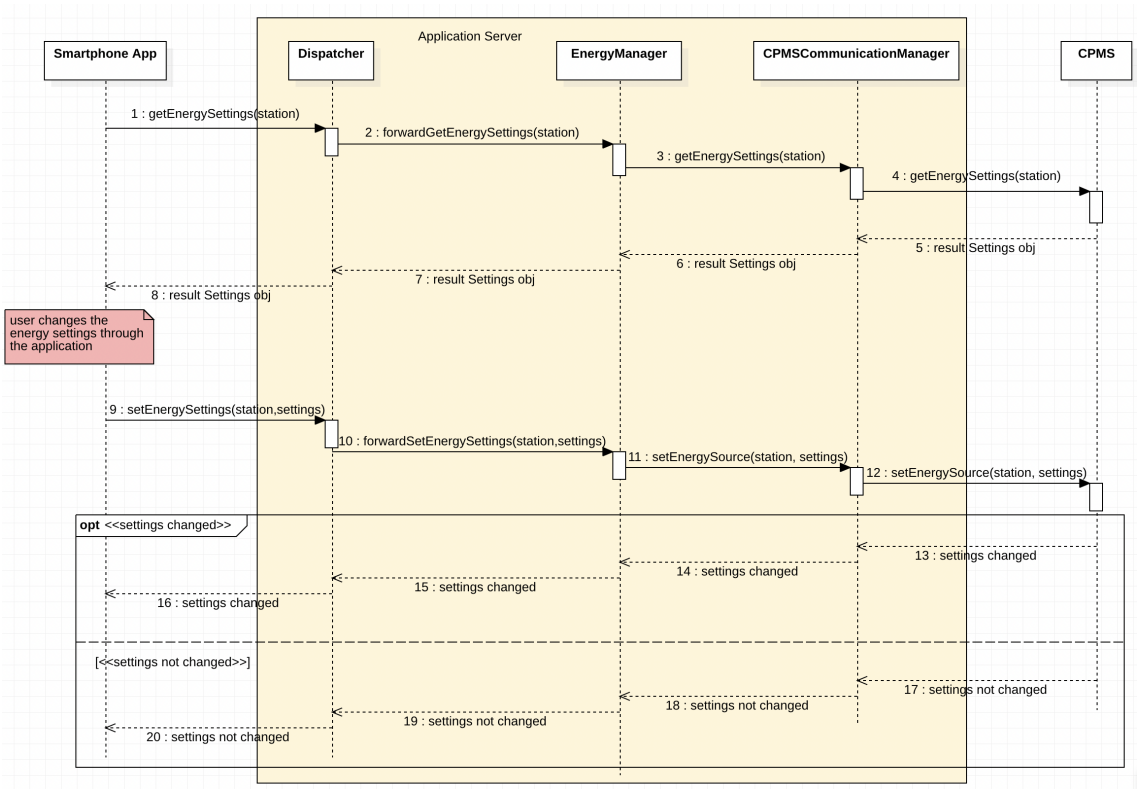

Figure 17: Manage energy source runtime view

2.5 Component interfaces

In this chapter is given a description of all the elements present in the component diagram in terms of the functions they offer.

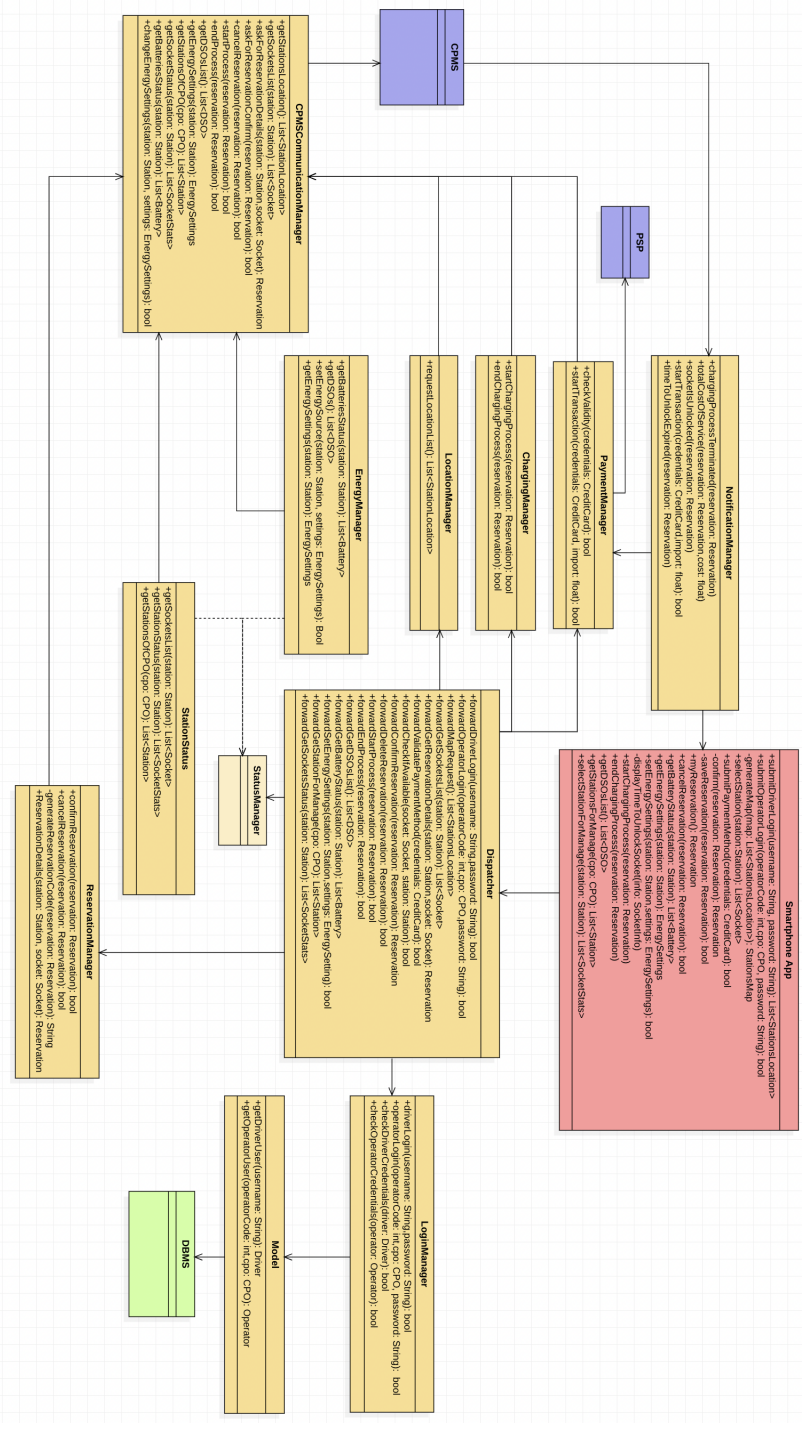

Figure 18: Component interfaces diagram

In figure 18 is represented the component interfaces diagram. Each component's color follows the

schema defined previously. It is possible to notice, written inside each component, all the functions that it offers (public functions, identified by a "+") and the function that it uses locally (private functions, identified by a "-"). The function names of the individual components (and their parameters) are the same as those used within the runtime views. However, in some cases the function names or parameters may differ from those presented in the runtime views, since in Figure 18 is represented the generic signature of the function, while in the diagrams is just represented their usage.

To understand the concepts underlying the classes in Figure 18, a brief explanation of their meaning is given below.

Reservation

Class that keeps information associated with reservations. In particular, it is used by the system both to forward booking requests made by drivers, and to communicate the information required by users to continue with the booking. (the user gets the general report of a booking before confirming it)

Station

The Station class maintains station information. In particular, it manages both the information visible to users and necessary for the system to forward requests to the CPMS.

CreditCard

Class that represents a payment method and contains the credentials provided by a user at the booking stage. Used by the system to verify the validity of payment methods and manage payments with PSPs

EnergySettings

The EnergySettings class keeps the information acquired by the CPMS about the settings of the power sources of a station that will then be shown to the operator via the mobile application. It is also used to forward to the CPMS the settings that the operator intends to implement to a station.

Socket - SocketStats

Classes that retain information associated with charging station jacks. In particular, the Socket Class manages the information required and necessary for the correct operation of the application for the driver user. Instead, the SocketStats class manages the information used by the user operator.

StationsMap

Is the class that includes the objects requested and provided by the map service used by the mobile application to display the map.

Battery

Class that will represent the battery entity and keep all the information displayed by the operator for the control of the energy status of the batteries of the stations.

DSO

The class DSO represents the entity of the Distribution System Operators and maintains the information on the costs of the energy demanded from the operators.

CPO

representative class of the Charging Point Operators (CPOs).

Driver - Operator

Classes that contain account information about eMall users (drivers and operators).

StationLocation

StationLocation is the class that manages information about stations and their locations retrieved by the CPMS and used by the mobile application to generate the map.

2.6 Selected architectural styles and patterns

2.6.1 3-tier Architecture

eMall will be built over a 3-tier architecture which provides many benefits thanks to the modularization of the system in three independent layers (or tiers):

1. **Presentation tier:** Top-level tier including the user interface. Its main function is to rearrange the received data from the application tier and present them to the user in a more intuitive and comprehensible manner.
2. **Business Logic tier:** It includes the business logic of the application, that is the logic according to which the application takes decisions and performs calculations. Moreover, it passes and processes data between the two surrounding layers.
3. **Data tier:** It includes the Database system and provides an API, to the application tier, for accessing and managing the data stored in the DB.

Adopting this type of architecture guarantees an higher flexibility by allowing at first to develop and then to update a specific part of the system apart from the others. Besides, a middle tier between client and data server ensures a better protection of the stored data. In fact the information is accessed through the application layer instead of being accessed directly by the client.

2.6.2 Facade Pattern

The facade pattern is used in the implementation of the Dispatcher component so to hide the complexity of the application server to the client applications to which is provided a simpler interface. This solution ulteriorly increases the decoupling of the various components of our system; in particular between the client applications and application servers.

2.7 Other design decisions

2.7.1 Smartphone App local data

The task of managing notifications to customers is assigned to the Smartphone App component. It is important to remember that the smartphone stores a copy of the booking after its confirmation. This information is necessary to avoid asking the system each time the user wants to consult it (including the time limit to show up at the station and unlock the charging socket). This will drastically reduce the number of requests made to the application server and will not impact the application itself, having to keep the information associated with only one reservation at a time.

2.7.2 OCPI protocol and Roaming peer-2-peer

Aspect not to be overlooked is the communication protocol used between the application server layer of the system and the different CPMS that are part of the eMall network.
The OCPI protocol, in particular with the latest version 2.2.1, allows the eRoaming via peer-to-peer connection, where both eMobility Service Providers (eMSPs) and Charge Point Operators (CPOs) can correlate immediately via OCPI. Separate interfaces are designed for eMSPs and CPOs. OCPI provides many features and many benefits, such as billing, tariff information, charging point status and platform monitoring. But roaming capability is very important, as it prevents users from being tied to a single charging network. The choice of design in the use of peer-to-peer communication is linked to the fact that the system intended to design communication between a single eMSP and several CPOs.

3 User Interface Design

This chapter provides an overview of the emall user interfaces. More precisely, in the following are presented the models of the UI of the mobile application both compared to the driver and operator user. The same models have been proposed as already presented in Chapter 3.1 of the RASD document, since they are considered to be quite complete on the idea of operation and presentation of the mobile application eMall.

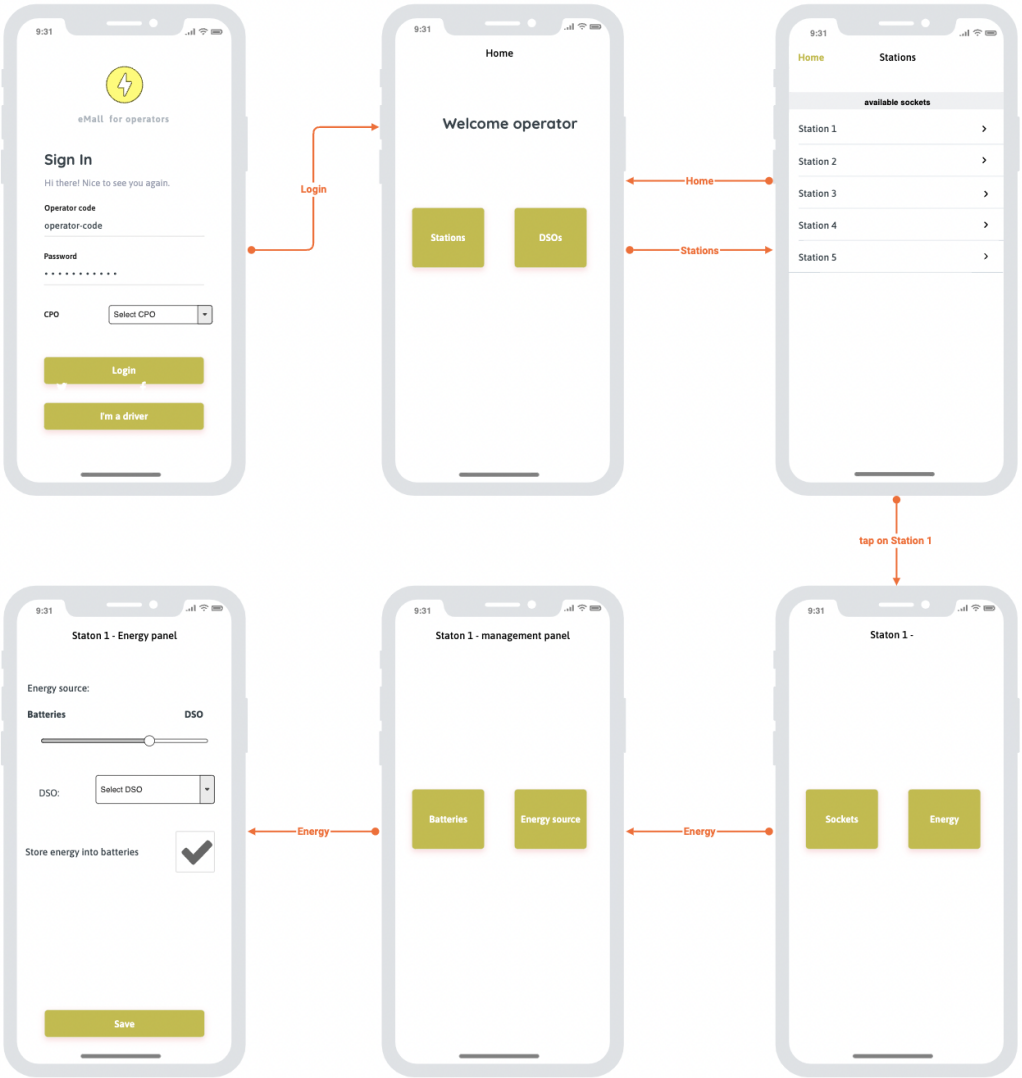

Figure 19: Operator UI mobile app

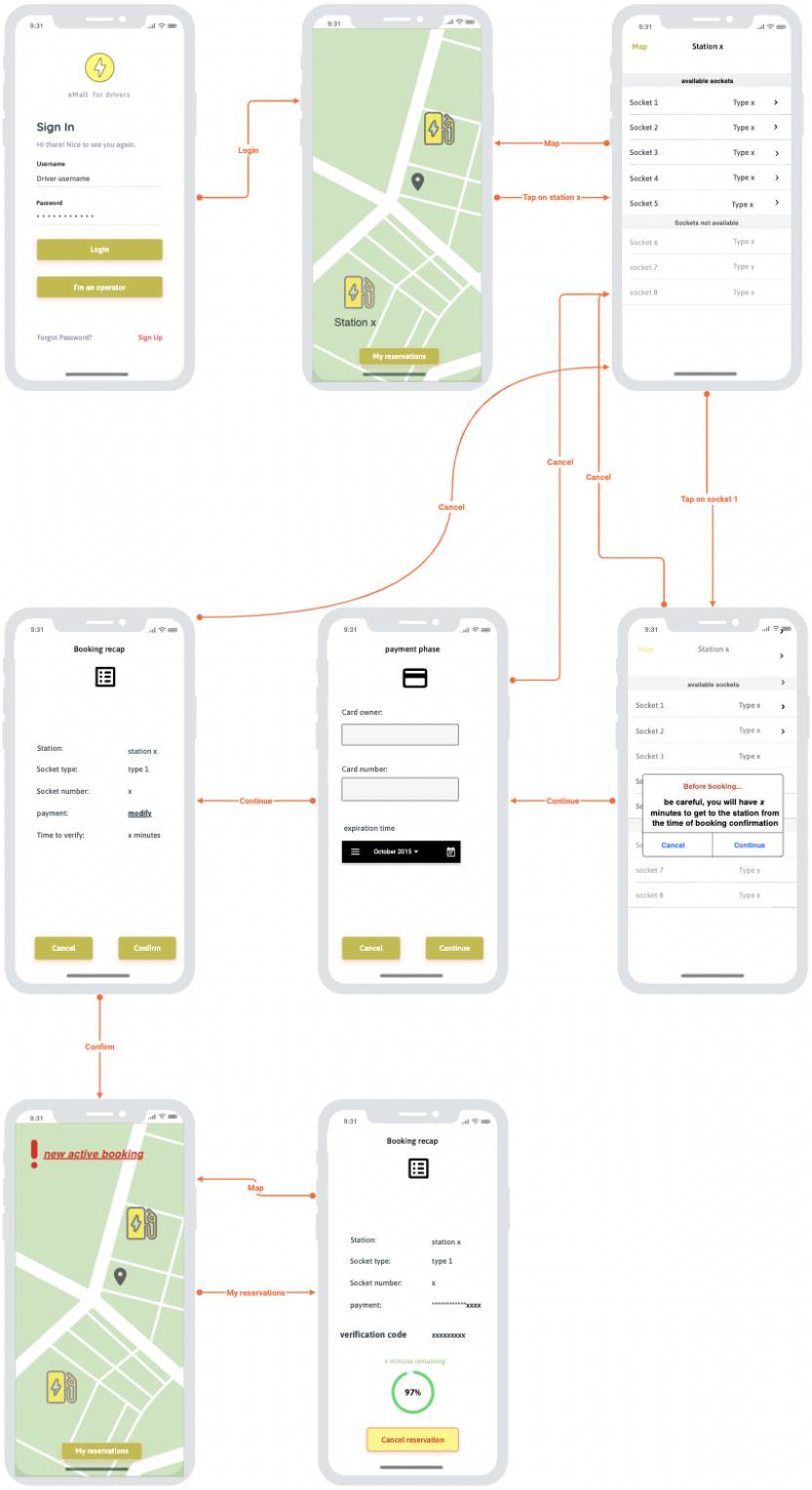

Figure 20: Driver UI mobile app

4 Requirements Traceability

In this section of the document it is explained, for each requirement defined in the RASD, what design elements are involved for its fulfillment. In the following a series of tables maps such design elements to the respective requirement.

Requirements	[R1] Allow drivers to log in [R13] Allow drivers to log in
Components	• Smartphone App • Application Server: – Dispatcher – LoginManager – Model • DBMS

Requirements	[R2] Allow drivers to select a charging station from a map
Components	• Smartphone App • Application Server: – Dispatcher – LocationManager – CPMSCommunicationManager • CPMS (external component)

note: were considered those components of the system that allow to retrieve information on the location of available stations and the subsequent construction of the map.

Requirements	[R3] Allow drivers to check a charging socket availability [R4] Allow drivers to check the socket types provided by a selected charging station [R5] Allow drivers to consult the charging costs of a specific charging station [R14] Allow operators to view the list of stations owned by the CPO they work for [R15] Allow operators to consult the number of charging sockets occupied [R16] Allow operators to monitor the estimated time until a busy socket becomes
Components	• Smartphone App • Application Server: – Dispatcher – StationStatus – CPMSCommunicationManager • CPMS (external component)

Requirements	[R6] Allow drivers to make one booking at a time
Components	• Smartphone App • Application Server: – Dispatcher – ReservationManager – PaymentManager – CPMSCommunicationManager • CPMS (external component) • PSP (external component)

note: the entire booking operation of a socket was considered (including verification of the correctness of the payment method).

Requirements	[R7] Allow drivers to view details of a booking that has not yet expired
Components	• Smartphone App

Requirements	[R8] Allow drivers to cancels a reservation that has not yet expired
Components	• Smartphone App • Application Server: – Dispatcher – ReservationManager – CPMSCommunicationManager • CPMS (external component)

Requirements	[R10] Allow drivers to start the charging process for a vehicle [R11] Allow drivers to stop an ongoing charging process
Components	• Smartphone App • Application Server: – Dispatcher – ChargingManager – CPMSCommunicationManager • CPMS (external component)

Requirements	[R12] Allow drivers to pay at the end of the charging process
Components	• Application Server: – NotificationManager – PaymentManager • CPMS (external component)

Requirements	[R17] Allow operators to see the list of the available DSOs [R18] Allow operators to consult the current price of energy of a DSO [R19] Allow operators to select the energy source for a station (batteries, DSO or a mix thereof) [R20] Allow operators to decide whether to store or not energy in stations'
Components	• Application Server: – Dispatcher – EnergyManager – CPMSCommunicationManager • CPMS (external component)

Requirements	[R21] Notify drivers when their charging process is ended [R22] Notify drivers when the time requested to unlock the socket is expired [R23] Notify drivers the total cost of the service obtained
Components	• Application Server: – NotificationManager • CPMS (external component)

5 Implementation, Integration and Test Plan

In this chapter is described the order in which you plan to implement the subcomponents of your system and the order in which you plan to integrate such subcomponents and test the integration and the methods to validate and verify.

The aim of the tests implemented for the system is to discover the majority of the application's bugs before every release.

Integration and implementation are strictly correlated, hence it often happens that the integration order coincides with the implementation one, that's why although this first chapter, the next one is dedicated to the implementation strategy, it happens to keep into account the integration test plan when defining it.

Another relevant instrument helpful to implement and test the code is writing well-commented and documented code (eventually with the help of some tool, such as Javadoc).

5.1 Implementation

An implementation plan outlines the steps to take to bring out the system. The system will be implemented following a combination of the bottom-up and thread strategies, so to combine the advantages of both approaches. Using a thread approach allows to produce intermediate deliverables; these are also immediately evaluable by the stakeholders. This approach is very helpful when it comes to the validation of the realized system for the two qualities described above. Whereas exploiting a bottom-up strategy promotes an incremental integration. Incremental integration facilitates bug tracking thanks to the possibility of testing the intermediate results (the sub-systems resulting from the integration of components) as additional modules get integrated.

The thread strategy's goal is to identify the features offered by the system and the part of the components (that will also be referred to as sub-components, even though they do not necessarily match with design sub-components) responsible for delivering such features. For each function, different subcomponents cooperate in order to realize it. For this reason, it is necessary to define the order of their implementation.

The order follows from a bottom-up approach. The bottom-up approach also tackles the difficulty of the interaction and cooperation between components. This implementation strategy allows assigning the task of implementing different features to independent development teams that work in parallel. It remains important to spot eventual common components and avoid producing the same component or subcomponent twice.

5.1.1 Features Identification

The system's features to be implemented have been extracted directly from the system requirements. Some of them require the implementation of a single component, while others require the implementation of more than one component. It follows there a brief recap of the features of the system.

[F1] Acquire charging stations information
This is one the core feature of the system. It is a feature intended for drivers only, not for operators. It enables drivers to book a charging process. It requires that the driver is logged in the application.

[F2] Book a reservation
This is the second core feature of the system. It is a feature available to drivers. It is composed of multiple sub-features that can be developed separately.

[F3] Control the charging process
This is also a driver's feature. It enables drivers to start, to stop and monitor the charging status.

[F4] Manage station energy source
This feature is reserved to operators. It is crucial to the management of the stations, it enables operators to ensure that there is always energy available to charge the vehicles by acquiring energy from DSOs or using stations batteries.

[F5] Monitor station status
This is an operator feature. It is helpful to the operators because it enables them to ensure that they're plan for a station are working. This feature is providing operators a feedback on the stations they monitor.

5.1.2 Features Implementation Plan

[F1] Acquire charging stations information
The first step is to log into the application, so this sub-feature will be the first to be implemented. Then it be must be implemented the sub-system that will return to the driver the stations and their details. The components involved are the following: SmartphoneApp, LocationManager, Dispatcher, LoginManager, CPMSCommunicationsManager, Model. The components dependency chain impose the following developing order, considering that we are following a bottom-up approach: Model, LoginManager, Dispatcher, SmartphoneApp, CPMSCommunicationManager, LocationManager.

[F2] Book a reservation
This feature also require the driver to be logged in. The first step is to log into the application, but this sub-feature is already implemented. Then it be must be implemented the sub-system for ensuring that the card used by the user is valid. And again a sub-feature that will return to the driver the stations and their details. The components involved are the following: LocationManager, PaymentManager. The components dependency chain impose the following developing order, considering that we are following a bottom-up approach: LocationManager, PaymentManager.

[F3] Control the charging process
This feature also require the driver to be logged in. The first step is to log into the application, but this sub-feature is already implemented. It is necessary the sub-system for starting the charging process. And a sub-feature that notify the driver when the battery is full. The components involved are the following: ChargingManager, NotificationManager. The components dependency chain impose the following developing order, considering that we are following a bottom-up approach: ChargingManager, NotificationManager.

[F4] Manage station energy source
This feature also require the operator to be logged in. This sub-feature is already implemented. It is necessary the sub-feature that enable the operator to manage the station. The components involved are the following: StatusManager. The components dependency chain impose the following developing order, considering that we are following a bottom-up approach: ChargingManager, NotificationManager.

[F5] Monitor station status
This feature also require the operator to be logged in and requires the sub-feature that enable the operator to manage the station. There are no new components to be implemented.

5.2 Integration

The order of features development is the following: F1 $\rightarrow$ F2 $\rightarrow$ F4 $\rightarrow$ F3 $\rightarrow$ F5. It follows the criteria of implementing first the most critical functionality to be offered, and the components inside them are developed bottom-up. The components are developed taking particular care to write them as reusable as possible and not strictly related to the functionality implemented.

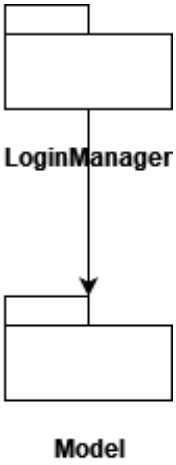

Figure 21: Model integration for F1

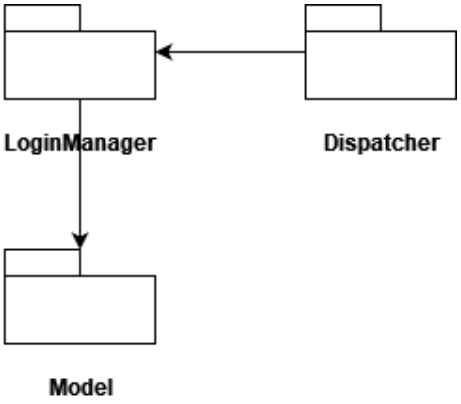

Figure 22: LoginManager integration for F1

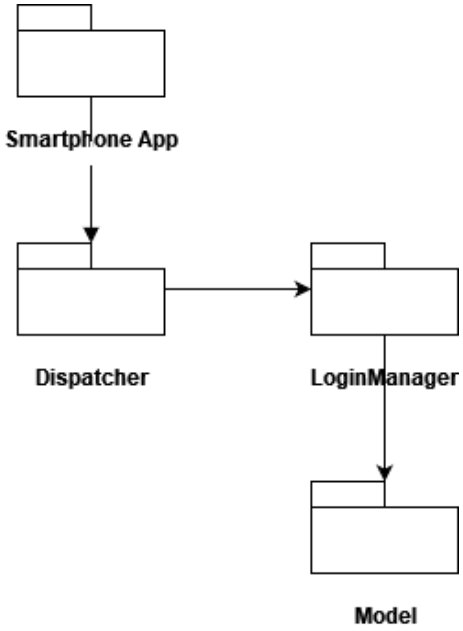

Figure 23: Dispatcher integration for F1

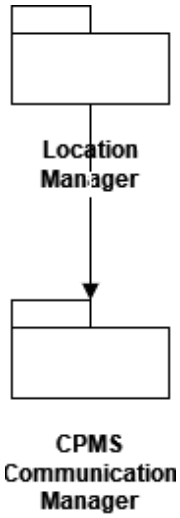

Figure 24: CPMSCommunicationManager and LocationManager integration for F1

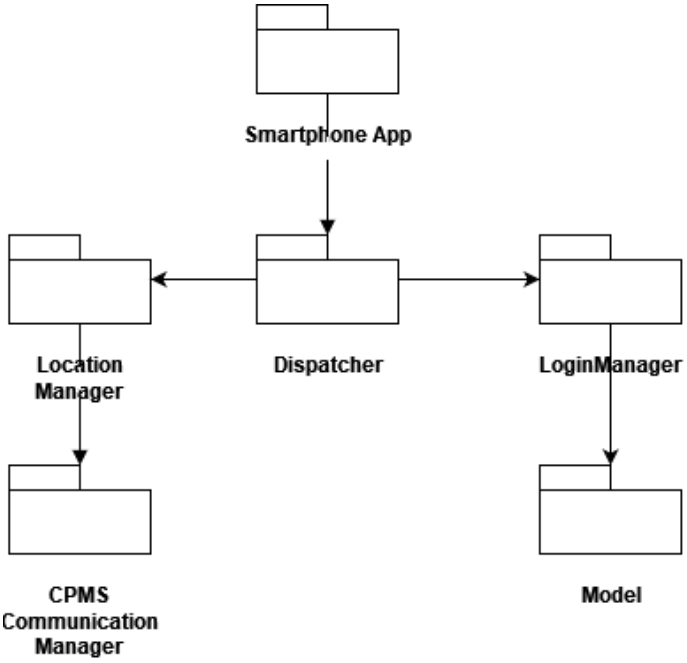

Figure 25: F1 (Acquire charging stations information) integration view

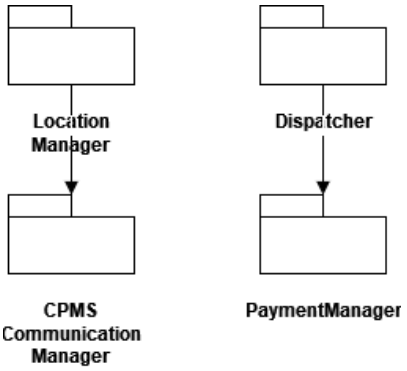

Figure 26: LocationManager and PaymentManager integration for F2

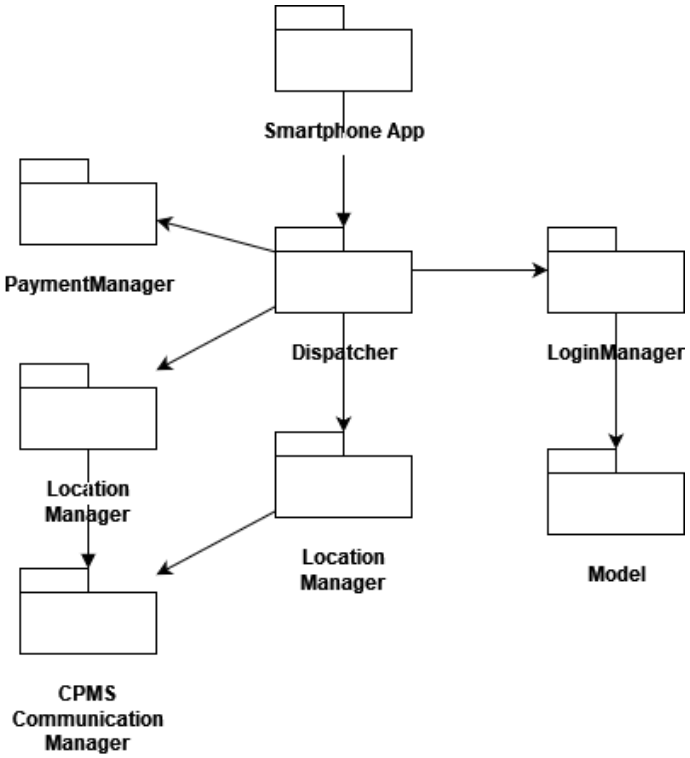

Figure 27: F2 (Book a reservation) integration view

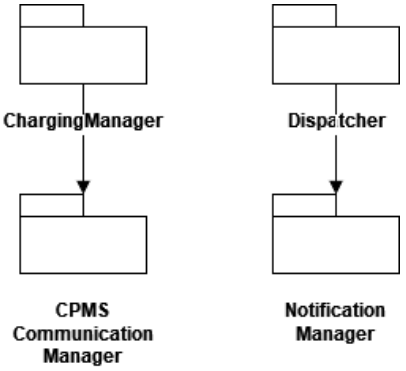

Figure 28: ChargingManager and NotificationManager integration for F3

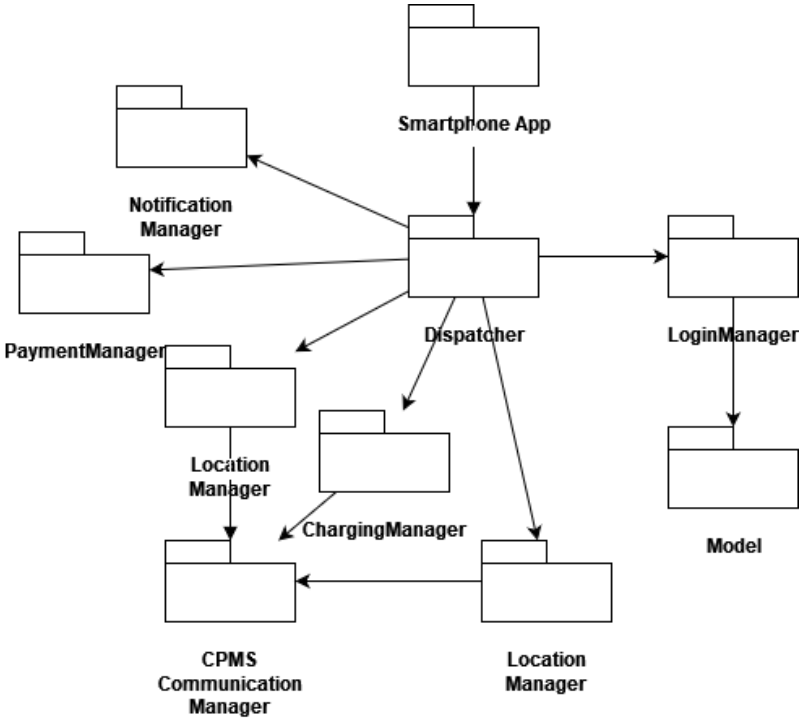

Figure 29: F3 (Control the charging process) integration view

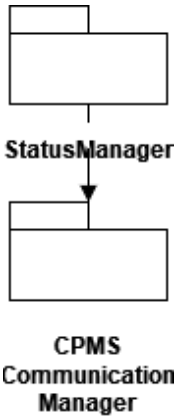

Figure 30: StatusManager integration for F4 F5

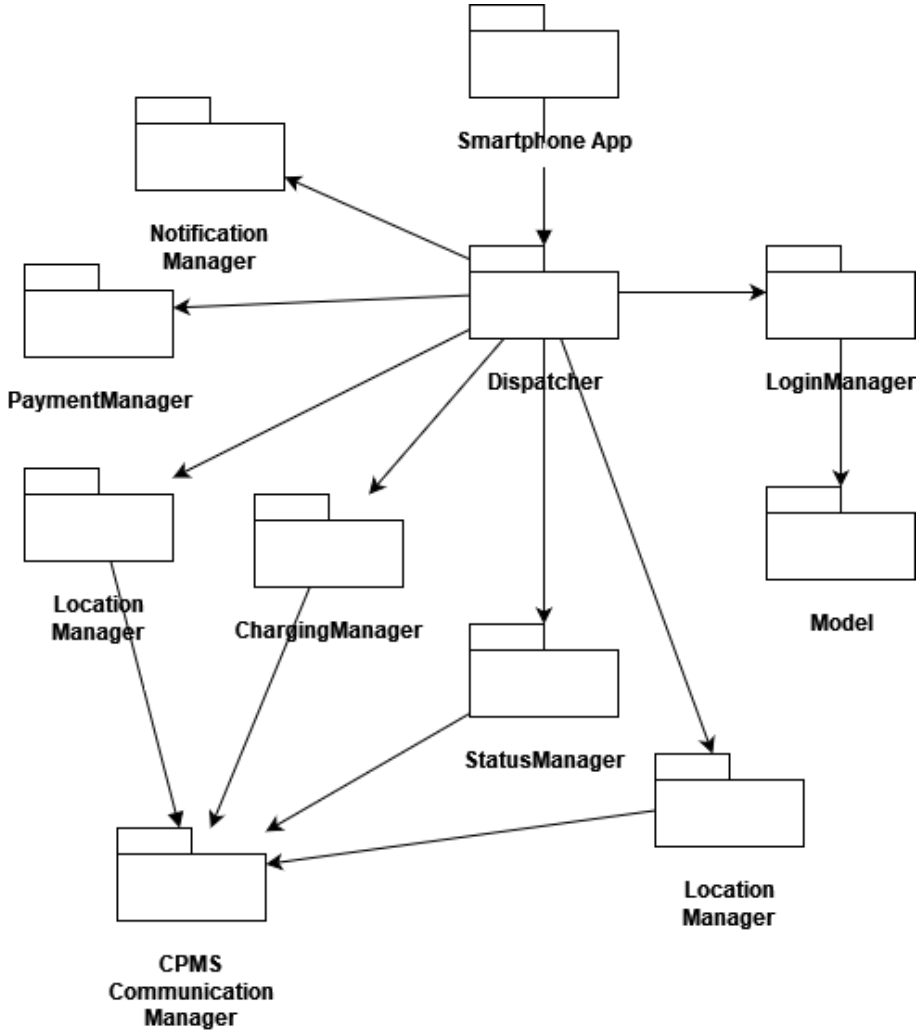

Figure 31: F4 and F5 (Manage station energy source, Monitor station status) integration view

5.3 Test Plan

5.3.1 System testing

The eMall system undergoes a series of testing activities, each one with its granularity and its scope. During the development phase, every single component or module needs to be tested on its own to check whether it is performing as expected. Since some components may not work in isolation, it will be needed to think and realize Driver and Stub components that simulate the behaviour of the surroundings modules (with both the expected and also unexpected conducts to check the component's robustness).

Once the testing of the single components has reached sufficient levels, it's time to integrate and then test the result of such integration. In our case, it is used the bottom-up strategy for integration and testing, as mentioned in the previous chapter. The following testing step is System testing. System Testing is conducted when all the system's features are completely integrated and it verifies compliance with both functional and non-functional requirements. A series of tests proposed for System Testing:

- **Functional testing**: to verify whether the system satisfies all the requirements specified in the relative Requirement Analysis and Specification Document (RASD). It is also possible to think of new features that might improve the user experience. An example could be to add new parameters that operators could monitor for the charging stations.
- **Performance testing**: the purpose of this testing phase is to spot bottlenecks and inefficiencies. These inefficiencies might affect heavily the system's performance.
- **Usability testing**: to establish how well users can utilize the system (Smartphone application) for accomplishing tasks. The system needs to be easy to use to avoid drivers and operators make mistakes. For these reasons, usability testing needs to be executed before delivering the system.
- **Load testing**: look for any bugs that may compromise the capacity of the system to handle a big number of requests. eMall must be ready to handle a big number of users simultaneously.
- **Stress testing**: tests the system's error handling capability especially when under a heavy load. eMall offers some critical functionalities such as monitoring the charging stations; if it fails bad consequences may occur.

5.3.2 Additional specifications on testing

The system has to be tested whenever new functions are added to check the presence of bugs. All the testing described above has to be done, to verify the correct behaviour of the system, during the implementation, to know if bugs are present.

During the development of the system, it is necessary to receive feedback from users and stakeholders. This should occur regularly, whenever a feature gets introduced. First, a description of the functionalities should be given to them, so that they can check if the system is going to be what they expect. Then, when available, some alpha versions of the system should be provided to them to receive feedback regarding eventual malfunctions. Doing so will contribute to validating the system during the creation process, and this is helpful. This way developers know if the system they are creating is correct, and can change some parts if its implementation does not reflect the customers' expectations.

Another vital testing that needs to be performed regularly is: check code quality via inspection. There are a lot of tools available online for doing so, GitHub is one of the most common.

6 Effort Spent

The time tables written below represent just an approximation of the time spent for the writing and discussions the team had for each specific chapter of this document. These times have not been measured while producing this document and are just based on the personal perception the team members have of the time spent.

Filippo Cinfrignini

Chapter	Effort (in hours)
1	3
2	14
3	10
4	13
5	15

Davide Esposito

Chapter	Effort (in hours)
1	1
2	23
3	12
4	16
5	7

7 References

[1] Assignment RDD AY 2022-2023_v3.pdf.

[2] Lectures slides of the Software Engineering II course on https://beep.metid.polimi.it/