
Politecnico di Milano
School of Industrial and Information Engineering
Master of Science in Computer Science and Engineering

Software Engineering 2 Project

# eMall - eMSP system

## Design Document

**Authors:**
Federico Bono
Daniele Cipollone

Academic Year 2022/2023

Milano, 08/01/2023
Version 1.0

# Contents

# List of Figures

# 1 Introduction

## 1.1 Purpose

The purpose of this document is to describe and explain in details the design choices for the development and deployment of the eMSP system of eMall. The description is structured on multiple levels to give an overview of the system from multiple viewpoints. In the following pages you will find:

1. High level overview of the architecture
2. Overview of the components of the system
3. Deployment overview
4. Overview of the interactions between components
5. Overview of the interfaces offered by the various components
6. The UI of the mobile application used by the final user
7. The patterns and technologies used in the system

## 1.2 Scope

eMall App is a platform that helps the end users to plan the charging process, by getting information about Charging Points nearby, their costs and any special offer; book a charge in a specific point, control the charging process and get notified when the charge is completed. It also handles payments for the service.

In the e-Charging ecosystem, there are many different actors involved that we need to keep into consideration while collecting requirements and designing the system. The first information to consider is that Charging Points are owned and managed by Charging Point Operators (CPOs) and each CPO has its own IT infrastructure, managed via a Charge Point Management System (CPMS).

In order to comunicate with the various CPMS, the OCPI (Open Charge Point Interface) protocol is used.

To be able to process payments, the system will need to communicate with a Payment Service Provider (PSP) via the HTTP(s) protocol, using the propertary APIs offered by the provider.

Given that our system is not the producer of the data, and that there is the need for implementing different functionalities (e.g. payments) a three-tier architecture has been chosen, to separate the data layer (that mostly acts as a cache layer) and the business logic layer.

## 1.3 Definitions, Acronyms, Abbreviations

### 1.3.1 Definitions

### 1.3.2 Acronyms

## 1.4 Revision History

- v1.0 - 05 January 2023

## 1.5 Related Documents

- eMSP RASD (RASD_eMSP.pdf)
- OCPI specifications document (OCPI-2.2.1.pdf)

## 1.6 Document structure

The document is structured in six sections:

1. Description and introduction of the various design choices made during the design of the system. Descriptions are written at different levels of abstractions: from the general point of view to the detailed view of the single component.
2. User interfaces and design mockups.
3. The requirement traceability matrix is used to map each component to the requirement(s) that fulfils.
4. Implementation and test plans for the entire system
5. Total effort
6. References used

# 2 Architectural Design

## 2.1 Overview: high-level components and interactions

As anticipated in the previous chapter, the architecture selected for the design and development of the system is the three-tier architecture.

This architecture allow us to split the implementation into three layers:

1. Presentation: is the mobile application that will be used by the final users. It allows all the interactions with the system and will also be used as a communication endpoint for the notifications.
2. Application: is the backend of the system, all the business logic and the various connections between the system and the external services are implemented here.
3. Data: is the layer responsible to expose connectivity interfaces from the database. It will be a DBMS.

All the layers communicate in a linear way: the Presentation one interacts only with the Application layer, the same as the Data layer. With this architecture the presentation and the Data layers have no direct communication path, this allow to develop all the business logic only in the Application layer. Other advantages of using this architecture are that the various layers can be developed with different technologies and that they can be duplicated and differentiated (i.e. there can be multiple presentation layers that interact with the same application layer)

The main reasons behind the choice are:

- We are not the producer of the data
- We need to integrate different external services
- Separate the business logic from the data to:
  - Allow a parallel development with multiple teams specialized in the single tiers
  - Allow to use different technologies for the different tiers

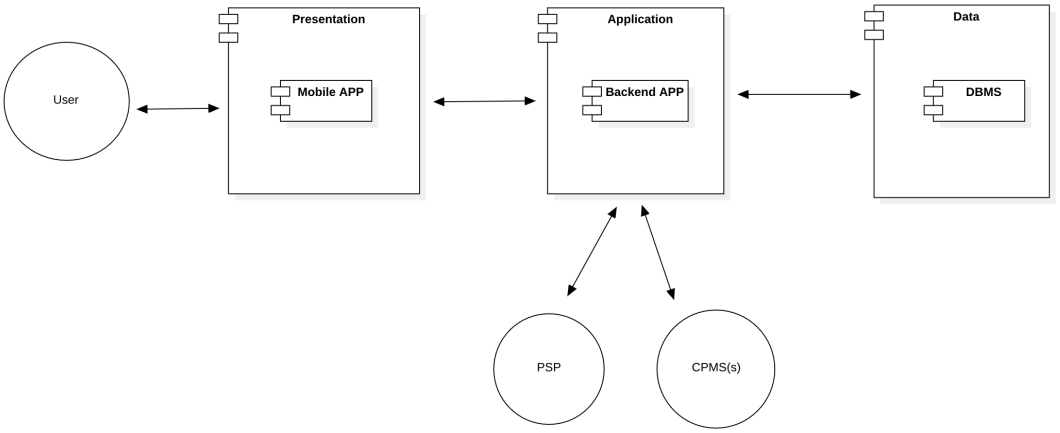


Figure 2.1: Overview of the chosen three-tier architecture with actors

## 2.2 Component view

The following schema shows all the main components and interfaces of the system. Later on you will find the details of each component.

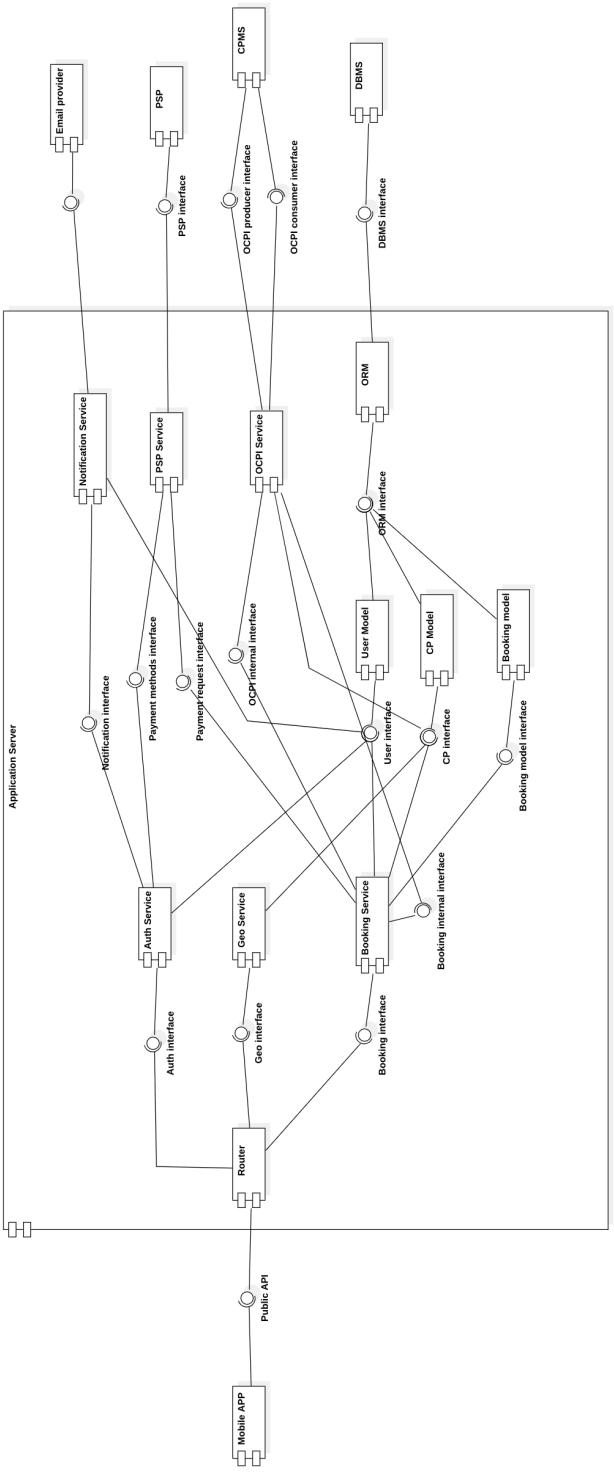


Figure 2.2: Component diagram of the system

- **Application server**: not a real "component" it shows the division between the backend, the frontend, the data layer and the external service. In the three-tier architecture it represents the Application layer.
- **Mobile APP**: mobile application for the system, used by the users.
- **Router**: handles all the requests directed to the Application APIs (i.e. OCPI PUSH endpoints are not handled here) that comes into the Application Server, uses the Auth Service to Authenticate and Authorize them. After that, if all the checks are passed, it will forward the request to the correct service that exposes the required route. It basically acts as a middleware, its presence is useful as it acts as a single point to develop all the authentication/authorization logic so that if it needs changes we can just update the router (and eventually the auth service) instead of updating all the services.
- **Auth Service**: is an internal service that handles authentication (Signup and login) and authorization (even if at the moment our system does not need that). It is mainly used internally but it exposes a couple of endpoints to the router for both signup and login.
- **Geo Service**: an highly reusable service, it uses a set of geographical points and exposes functionalities to list, filter and search those points based on their position. For our system is needed for the map functionality of the Mobile APP.
- **Booking Service**: it manages the booking for the application, it exposes some endpoints to the router for the booking functionalities of the Mobile APP and it is also used from other internal services.
- **Notification Service**: it manages all the notifications that the system needs to send to the user, both via email as via push services (e.g. Firebase Cloud Messaging)
- **PSP Service**: it manages the communication with the Payment Service Providers, it is useful as we can easily swap it with proprietary SDKs from the various providers.
- **OCPI Service**: it manages the communication with the CPMS, it exposes the endpoints for the PUSH part of the protocol (is needed to receive real-time updated from CPMS)
- **ORM**: a library that allows an easy to use mapping between Models and DB table, it handles queries and relationships. It is not worth it to develop an in-house solution, so we will use a library for that.
- **User Model, CP Model, Booking Model**: Models components that sit between services and ORM, useful to attach event listeners and other effects that need to run when updating the data.
- **Email Provider**: external service (or services) that exposes API to send emails
- **PSP**: external service that exposes API to process payment
- **CPMS**: Charging Point Management System that exposes OCPI compliant API
- **DBMS**: Database Management System

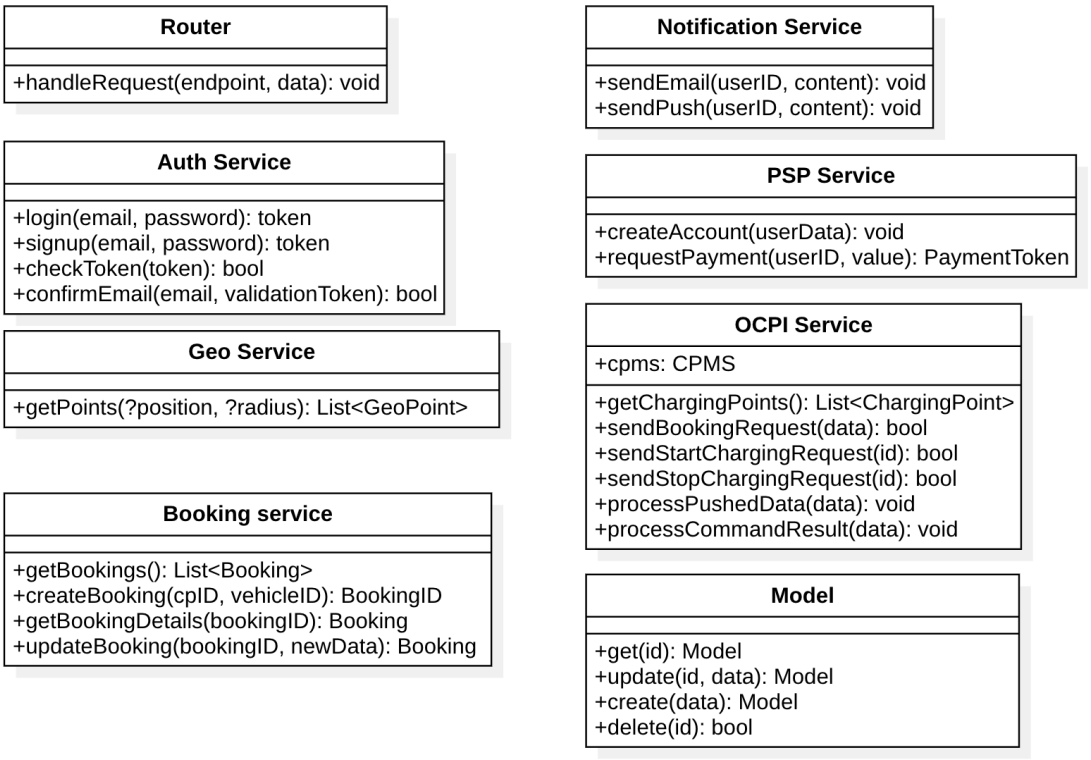


Figure 2.3: Class diagram for the main interfaces of the System

## 2.3 Deployment view

In this section we introduce a detailed view of the system and various component from a deployment cycle perspective. The following schemes highlight the environments and the tools to be used for the system.

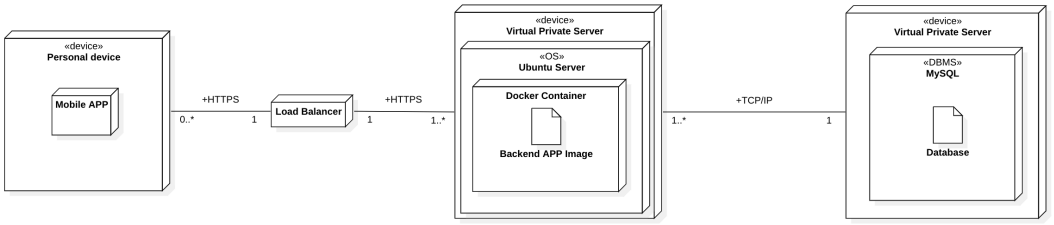


Figure 2.4: Deployment diagram for the System

- **Personal Device**: Mobile device used by users to run the Mobile APP to communicate with the system
- **Load Balancer**: A web server that balances incoming requests between multiple backend servers
- **Backend Server / Virtual Private Server**: Physical device where is installed the Docker Image of the Backend APP. Multiple replicas are possibile and preferable as they will increase both reliability and availability of the system. Keep in mind that without replicating the data layer, there will always be a bottleneck of performances due to the communication between application and data layers.
- **Database Server**: Host the database of the system. Multiple local replicas can improve read performances but also increase complexity.

## 2.4 Runtime view

In this section are shown the sequence diagrams for the various functionalities of the system. To simplify the description there is an initial generic sequence the summarize the routing and authorization part of the request handling. Also, as all the request go through the router, the actors that sends the request to the router are omitted.

### 2.4.1 Generic Routing and Authorization

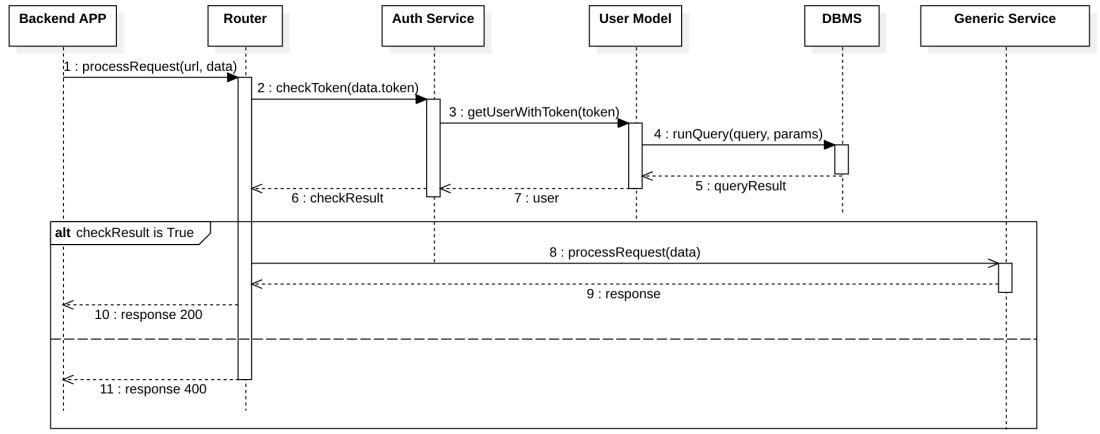


Figure 2.5: Generic Routing and Authorization flow for a generic request

When a request is received by the application web server, it goes through the Router component. It acts as an authorization middleware. As Authorization is needed, the whole Auth Service, Model and DBMS components are envolved. At the end, if authorization has been completed successfully, the request is forward to the dedicated service.

### 2.4.2 View Charging Point locations

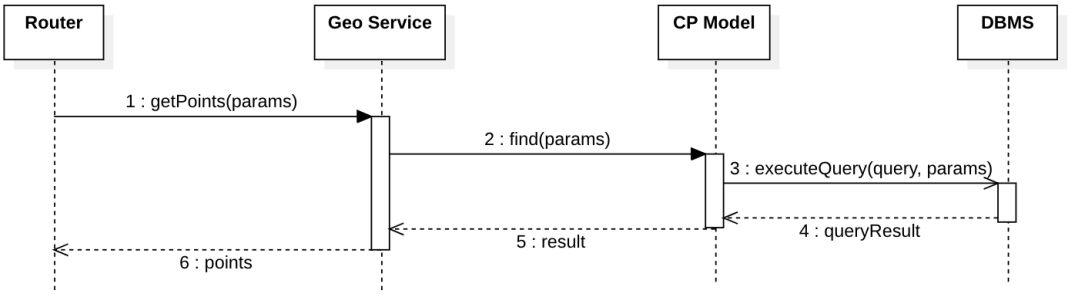


Figure 2.6: View Charging Point locations

### 2.4.3 Book a charging session

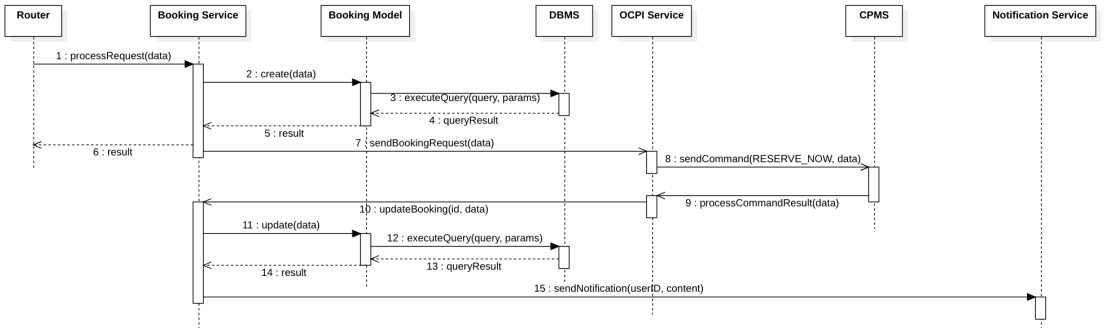


Figure 2.7: Book a charging session

### 2.4.4 Start a charging session

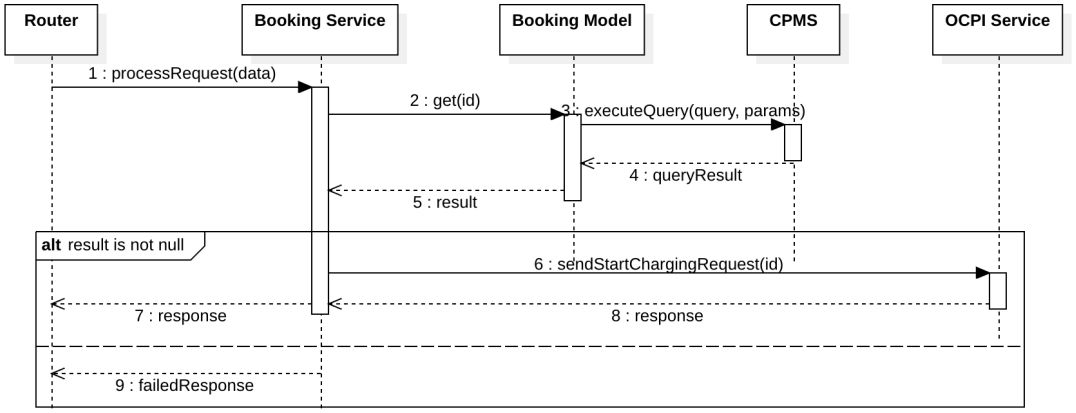


Figure 2.8: Start a charging session

### 2.4.5 Pay for a completed charging session

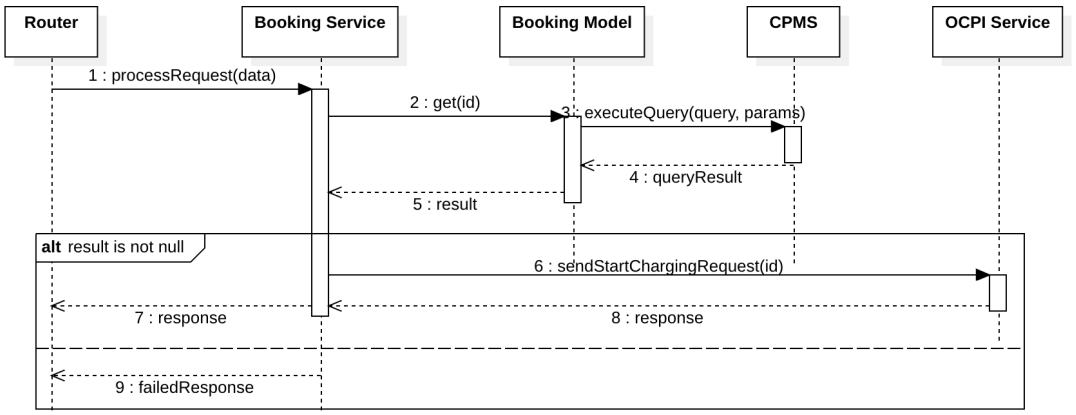


Figure 2.9: Pay for a completed charging session