eMall - e-Mobility for All

Implementation and Test Document

05/02/2023

Version 1

Authors:

Brugnano Matilde

Buttiglieri Giorgio Natale

Repository:

giorgio-natale/BrugnanoButtiglieri (github.com)

Academic Year 2022/2023

Software Engineering 2 Project

Table of Contents

1 INTRODUCTION

1.1 Purpose

This document aims at showing the implementation process of a minimum viable product (MVP) for the eMall project referred to by the RASD and DD documents.

In particular, it describes the functionalities of the software, the adopted development frameworks, their pros and cons, and the deployment processes we used. Moreover, it includes a description of the structure of the source code and the testing process we followed. Lastly we describe how to access the MVP and the effort spent for the implementation.

1.2 Acronyms

Term	Definition
eMSP	e-Mobility Service Providers: company offering a charging service to drivers of electric vehicles. Here, eMSP is intended as the software system used by the company to manage charging processes.
eMSP app	Mobile app through which drivers of electric vehicles can interact with the services offered by eMSP.
CPO	Charge Point Operator: company that owns and manages a network of charging stations.
CPMS	Charge Point Management System: software system that manages the charge point infrastructure of a CPO company through its connection with the charging equipment.
CPMS app	Web app through which CPO operators can interact with the services offered by CPMS.
MVP	Minimum viable product: version of a product with the minimum usable features to be tested by early customers

1.3 Scope

The process of implementation has been guided by the necessity of showing early customers the main usable features of this product, in order to let them provide feedback for future product development.

In the following section, we provide an overview of the implemented functionalities.

1.4 Revision History

- 05/02/2023: Version 1

2 IMPLEMENTED FUNCTIONALITIES

2.1 Selected functionalities

We selected a subset of the functionalities described in the RASD document that could put together a usable product, focusing on the primary purpose of eMall: managing bookings and handling and monitoring the charging process.

In particular, the main use cases implemented for eMSP are:

- User signs up
- Customer logs in
- Customer finds a charging station
- Customer books a charge, both in advance and on the fly
- Customer deletes a booking
- Customer starts charging their vehicle
- Customer monitors its bookings

The main use cases implemented for CPMS are instead:

- The delivery of energy to a vehicle stops due to full battery
- The delivery of energy to a vehicle stops due to booking timeframe ending
- CPO Employee logs into the CPMS
- CPO Employee visualizes the overview of a charging station.

2.2 Implemented components

Our MVP consists of:

- A mobile app that can be used to find charging stations nearby and handle bookings and charges [eMSP app]
- A web app that can be used by CPO Employees to monitor the status of charging stations [CPMS app]
- The microservices supporting the implemented functionalities:
    - CustomerService [eMSP]
    - BookingManagementService [eMSP]
    - EmployeeService [CPMS]
    - BookingService [CPMS]
    - ChargingManagementService [CPMS]
- A component that mocks some of the functionalities we did not include in the implementation (e.g. the configuration of charging stations)
- A component that mocks the physical charging points.

3 ADOPTED FRAMEWORKS AND TECHNOLOGIES

3.1 Back-end

3.1.1 Microservices

Microservices consist of a stateless application responsible for the application logic and a stateful application (datastore) responsible for storing and maintaining the state of the microservice.

We opted for Spring Boot Framework for the stateless application and PostgreSQL for the stateful application.

Our microservices also expose their REST API. We adopted OpenApi 3.0 standard to describe them and to help us with the code generation of the implementation.

Spring Boot

Spring is a popular framework for writing standalone applications that run on the Java Virtual Machine.

Spring Boot is a microframework based on the Spring Framework. In addition to the Spring Framework it provides opinionated default configurations that allow developers to quickly set up a functioning Spring application.

The main modules we picked from Spring are:

- Spring Web MVC
    - The Web module contains the main components used to handle HTTP requests. In addition, the Spring MVC implementation shipped with this module provides REST Web Services implementation following the Model-View-Controller pattern.
- Spring Data
    - The Data module contains the main components used to communicate with the persistence layer. We used JPA to handle persistence, in particular the Hibernate ORM implementation (default one).
- Spring for Apache Kafka
    - The Spring Kafka module contains the main components used to communicate with the Apache Kafka broker. It provides ready-to-use implementations of both consumer and producer clients and abstractions to simplify the interaction with the broker.
- Spring Cloud
    - Broad family of components useful to develop Spring applications that can run on the Cloud. They include service discovery, configuration manager, API gateway and much more. For this project we only used the circuit breaker sub-module (implementation of Resilience4j Circuit Breaker and Reply) since the underlying deployment infrastructure, Kubernetes, already handled most of the mentioned functionalities.

Pros

We opted for Spring Boot because it is an established framework, well-known by developers. The popularity of the framework has been a really important factor in our decision, given the fact that other tools we used (e.g. OpenApi code generator) already had a complete integration with the Spring Boot ecosystem.

Furthermore, by having an opinionated view of the Spring Framework, Spring Boot allows the developers to better manage their time focusing on the core logic instead of spending time writing configuration files.

Lastly, Spring Boot provides an easy way to deploy the application since it ships a single Jar package with a ready-to-use Web Server (Tomcat by default).

Cons

Having an opinionated view, Spring Boot makes it easier to stick with the default configuration, but at the same time it adds some complications when seeking specific and non-standard use cases. Since we stuck with the standard use cases, this con was deemed to be negligible.

Another possible con would be how easy it can be for the Jar to become unnecessarily heavy due to the significant number of dependencies included in the Spring Ecosystem. This can be avoided by making sure to only pull relevant dependencies for each project. In any case, this con is largely outweighed by the pros.

PostgreSQL

PostgreSQL is a popular object-relational database. It is open-source and lightweight.

Pros

PostgreSQL is a mature DBMS featuring good concurrent management strategies such as MVCC without read locks and parallel query plans.

Being lightweight, it is easy to deploy even with limited computing resources.

Another pro is that PostgreSQL is an object-relational database which allows for a more expressive model design, which in some cases can be beneficial. This can be done by using features such as table inheritance and complex data types.

Lastly, PostgreSQL is popular and well-established, so integration with other tools is seamless.

Cons

PostgreSQL is not natively distributed, meaning that it is more difficult to scale in case of excessive workload. Other solutions like MongoDB or Cassandra offer this functionality, but PostgreSQL, being relational, offers stronger consistency guarantees in critical use cases.

3.1.2 Message broker

Asynchronous communication between microservices has been handled using a message broker.

This approach makes it possible to lower the coupling between microservices: they do not need to be all simultaneously available to perform their tasks. This increases the overall availability of the system.

Apache Kafka

Apache Kafka is an open-source distributed event streaming platform. We used it as a message broker, specifically to handle state update propagation.

Pros

Apache Kafka is natively distributed and designed to be used in distributed systems, so it is a good fit for the architecture of this project.

An important requirement for eMall is to be highly available, especially for the components responsible for communicating with the charging points. Being highly available, Kafka helps our system to satisfy this requirement.

Microservices responsible for the charging process are quite sensitive to state changes. Kafka allows the persistence of sent messages for a configurable amount of time, therefore allowing said microservices to recover from failures.

Lastly, Kafka synergizes well with the Spring ecosystem, making it easier to interact with it.

Cons

Kafka is quite difficult to configure because it is designed to satisfy a broad type of use cases and requirements.

Moreover, it requires a significant amount of computing resources, especially RAM usage. For projects that do not have strong consistency requirements, solutions that are purely message brokers like RabbitMQ might be a better choice.

3.2 Front-end

This section provides an overview of the technologies and frameworks used for the development of the eMSP mobile app and the CPMS web app. Below, the key topics for front-end development are listed, describing the choices made for each of them.

3.2.1 Framework selection

For what concerns the core technologies to base the development on, we have selected React as a framework for the web app and React Native as one for the mobile app.

Pros

The two are very popular frameworks used for building modern and scalable web and mobile applications and, in particular, the use of React and React Native is useful because it allows for efficient code reuse and a streamlined development process.

The use of React provides indeed a flexible and efficient way to build and maintain web applications, and React Native allows us to leverage the benefits of React to develop high-performance mobile apps with a single codebase that can run on both iOS and Android platforms.

Moreover, they both offer fast performance and a large and active community, which integrates the use of the frameworks with many ready-to-use libraries and components.

In particular, our choice for the mobile app was made along with the choice of Expo, a tool for developing mobile apps with React Native. Among its several advantages, Expo allows developers to continuously work on mobile applications without having to release a new version of the app on the App Store or Google Play, provides cross-platform compatibility and includes a lot of built-in tools and services. For example, we made use of their support for push notifications.

Cons

Some cons for React may be found in the complexity of the framework, which leads to a steep learning curve and the difficulties onset as the application grows in size.

About React Native we can find cons in its dependence on native modules, which can complicate the development process, and limited performance with respect to its native equivalent.

3.2.2 Navigation and routing

For what concerns the navigation and routing, our choice was to use React Navigation for the mobile app and React Router for the web app.

React Navigation provides an easy and highly-customizable way to handle navigation between screens through the creation of tab bars and stack navigators, which offer a consistent navigation experience across different platforms.

On the other hand, React Router provides a way to map URL paths to components, allowing for immediate handling of dynamic and nested routing.

3.2.3 Data management

Data management benefited greatly from the use of Typescript, a strongly typed superset of JavaScript. Using TypeScript we could write more reliable and maintainable code avoiding errors with data access and manipulation.

In addition to this, React Query was used for both web and mobile applications to manage and fetch asynchronous data from APIs. It provides a way to cache and share data between components, reducing the amount of data that needs to be fetched multiple times, and offers features like loading state management and error handling.

In particular, the use of React Query allowed us to implement a dynamic polling algorithm to change the frequency of requests on the most critical status changes in the application data.

3.2.4 Form handling

We used Formik for both the web and mobile applications, which provides a simple and intuitive syntax for handling forms, reduces the amount of boilerplate code needed and offers built-in support for validation.

3.2.5 User interface

We have used React Native Paper library for the mobile app and React Bootstrap library for the web app, leveraging components offered by the Volt template for the latter.

The wide range of pre-built components offered by the two UI libraries helps create a consistent and attractive user interface while still maintaining the flexibility to customize them to our specific needs.

3.3 Deployment and CI/CD

The implementation of the MVP followed the CI/CD approach. We adopted processes that allowed new code changes to be automatically built, tested and merged to a shared repository.

As soon as a new commit is pushed to the main branch of the repository, an image of the updated component is built and published on a shared artifact registry.

Lastly, the new version of the component gets automatically deployed in the staging environment, substituting the old version.

3.3.1 Environment setup

To support the CI/CD activities, we used the services offered by Google Cloud Computing and Google Kubernetes Engine

- Two GCP VM instances that are nodes of a Kubernetes cluster
- A Jenkins namespace with a running Jenkins controller. The Jenkins controller is configured to listen to updates of the main branch of the repository of the project
- A namespace for the eMall project in which all components of the system are deployed.

3.3.2 Pipeline

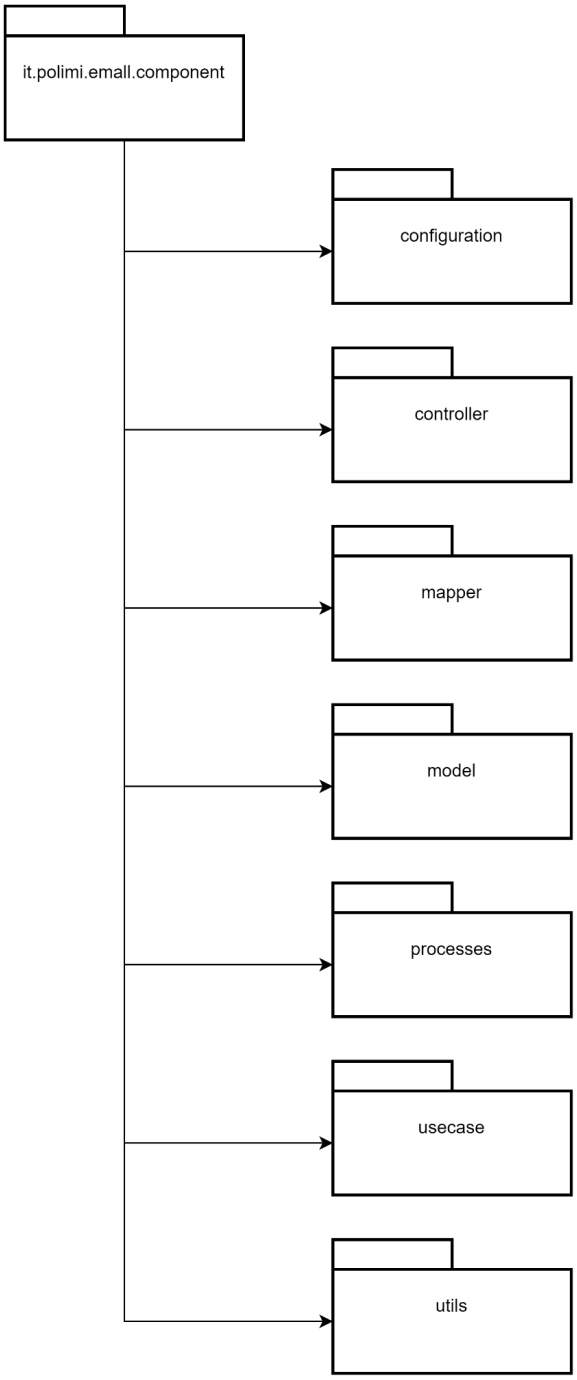
These are the steps of the CI/CD pipeline we used

1) When a commit is pushed to the main branch, the Jenkins controller gets triggered (via GitHub Hooks) and starts the deployment of the component that has been updated
2) The Jenkins controller spawns a new Jenkins Agent pod that clones the repository, builds the component being updated, run the tests and pushes the new image of the component to the artifact registry of the project
3) The Jenkins Agent then parses the Kubernetes manifest located in the folder /ci-cd of the component replacing the old image reference with the new pushed one
4) The Jenkins Agent applies the new manifests to the Kubernetes cluster and sends us a notification on Telegram about the success/failure of the pipeline
5) Kubernetes makes sure that the new version of the component is up & running in the staging environment.

4 SOURCE CODE STRUCTURE

4.1 Back-end structure

The structure of the code of all microservices is homogeneous and reflects the main structure described in the DD document, with the addition of some packages that contain supporting modules (mapper, processes, configuration, utils).



- Configuration
    - Contains configuration beans for

        - Kafka integration
        - Custom Jackson ObjectMapper
        - WebClient beans used to make HTTP requests to other microservices or external services
- Controller
    - Contains the definition of the controllers that implement the REST interfaces
- Mapper
    - Contains some utility functions used to translate between different representations of the entities. For example, it maps the model entities to the data transfer objects exposed through the REST API
- Model
    - Contains the model of the domain, encapsulating the core model entities, the managers and the repositories (for further details about these components, please refer to the DD document)
- Processes
    - Contains startup and periodic processes needed to keep the state of the microservice updated
- Usecase
    - Contains the implementation of the use cases the microservice takes part in
- Utils
    - Contains utility general functions

4.2 Front-end structure

The source code structure is consistent for both web and mobile applications. In particular, the first-level folders reported below reflect the key topics described in the paragraph 3.2 about front-end technologies:

- Api
    - Contains query configuration for API requests
    - Contains token handling for API requests
- Features
    - Divided in folders one for each implemented feature (or page), you can find:
        - Page/Component files
        - Related API file
- Router/Navigation
    - In the web application, it contains the mapping between pages and routes
    - In the mobile application, it contains the screen, stack and tab navigators definitions

In addition to this, you can find the Notification folders and the Authentication folders in the mobile app project in which the related managers are responsible for the related functionalities.

5 TESTING

Since we embraced a CI/CD approach for development, new features were always applied to the staging environment as soon as they were developed.

We performed end-to-end testing from the beginning every time a new feature reached the staging environment.

The main integration tests we performed are described below.

Id	Test-1
Name	CPO Employee logs in
Initial page	Web app, login page
Preconditions	The CPO Employee is not logged in
Steps	1) The user fills the form inserting "juliet@my-cpo.com" as email address and "password" as password 2) The user presses the button "Login"
Expected result	The user is now logged in and the charging station list page is shown to them

Id	Test-2
Name	Customer fails to login
Initial page	Mobile app, login page
Preconditions	The Customer is not logged in
Steps	1) The user fills the form with incorrect informations (email address and/or password) 2) The user presses the button "Login"
Expected result	An error message appears under the login form informing the user of the procedure failure

Id	Test-3

Name	User successfully signs up
Initial page	Mobile app, login page
Preconditions	- The User is not logged in - There is not the user test@mail.com in the system
Steps	1) The user presses the “Sign up” button at the bottom of the login page 2) The form to sign up is shown to the user 3) The user fills the form inserting name, surname, test@mail.com as the email address, test-password as password 4) The user presses the button “Sign up”
Expected result	The User (now a Customer) is presented with a map showing the charging stations in the area

Id	Test-4
Name	Customer can get info about nearby charging stations
Initial page	Mobile app, map page
Preconditions	- The Customer is logged in
Steps	1) The Customer taps the icon of a charging station on the map
Expected result	A new panel appears at the bottom of the page showing name, address, price and offer for the selected charging station

Id	Test-5
Name	Customer can land in the booking page
Initial page	Mobile app, map page
Preconditions	- The Customer is logged in

Steps	1) The Customer taps the icon of a charging station on the map 2) The Customer taps the button “Book” at the bottom of the page
Expected result	The Customer is presented with the page for booking a charge in advance

Id	Test-6
Name	Customer can create a booking in advance
Initial page	Mobile app, map page
Preconditions	- The Customer is logged in - There exists at least one tuple of charging point - socket available in the selected charging station for the inserted timeframe and socket type
Steps	1) The Customer taps the icon of a charging station on the map 2) The Customer taps the button “Book” at the bottom of the page 3) The Customer fills the form inserting the desired start date time for the charge, the desired duration and the socket type 4) The Customer taps the button “Submit booking” at the bottom of the page
Expected result	The Customer is presented with the page of the booking list, where he can see the newly created booking. For the newly created booking the page shows the booking code, inserted start date time, inserted duration, the tuple of assigned charging point code - socket code, the inserted socket type and the selected charging station address.

Id	Test-7
Name	Customer can create a booking on the fly
Initial page	Mobile app, map page

Preconditions	- The Customer is logged in - There exists at least one tuple of charging point - socket currently available in the selected charging station
Steps	1) The Customer taps the icon of a charging station on the map 2) The Customer taps the button “Book” at the bottom of the page 3) The Customer swipes right to the page where the form to submit a booking on the fly is shown 4) The Customer selects from the shown list the desired tuple of charging point - socket 5) The Customer taps the button “Submit booking” at the bottom of the page
Expected result	The Customer is presented with the page of the booking list, where he can see the newly created booking. For the newly created booking the page shows the booking code, current date time, the tuple of selected charging point code - socket code, the related socket type and the selected charging station address.

Id	Test-8
Name	Customer can activate a booking on the fly
Initial page	Mobile app, booking list page
Preconditions	- The Customer is logged in - The Customer has already created a booking on the fly, which has not been activated or deleted
Steps	1) The Customer scrolls the booking list until they find the created booking on the fly, whose background is highlighted/coloured 2) The Customer presses the button “Charge” next to the details of the created booking on the fly
Expected result	When loading is completed, the activated booking shows the left minutes time next to a battery icon, which represents the left time for the vehicle’s battery to be fully charged. The left minutes update regularly. In the web app, the related tuple of charging point - socket of the

	correspondent charging station is highlighted and the left minutes for the vehicle's battery to be fully charged is shown together with the power absorbed by the vehicle.

Id	Test-9
Name	Customer can activate a booking in advance
Initial page	Mobile app, booking list page
Preconditions	- The Customer is logged in - The Customer has already created a booking in advance, which has not been activated or deleted and which is related to the current date time (which means that the booking start date time + duration is before the current date time)
Steps	1) The Customer scrolls the booking list until they find the created booking in advance, whose background is highlighted/coloured 2) The Customer presses the button "Charge" next to the details of the created booking in advance
Expected result	When loading is completed, the activated booking shows the left minutes time next to a battery icon, if the bookings timeframe would end after the full charge of the vehicle's battery, or a timer icon, if the booking ends before the vehicle's battery could reach its full capacity. The left minutes update regularly.

Id	Test-10
Name	Customer can delete a booking in advance
Initial page	Mobile app, booking list page
Preconditions	- The Customer is logged in - The Customer has already created a booking in advance, which has not been deleted and whose start date time +

	duration is before the current date time (otherwise, the booking changes its status to “Expired”)
Steps	1) The Customer scrolls the booking list until they find the created booking in advance 2) The Customer presses the button “Delete” next to the details of the created booking in advance
Expected result	The deleted booking shows its new status “Deleted” in the top right corner of the related element

Id	Test-11
Name	Customer is notified for the completion of an activated booking
Initial page	Mobile app, booking list page
Preconditions	- The Customer is logged in - The Customer has already activated a booking which is not completed yet
Steps	1) The Customer scrolls the booking list until they find the activated booking
Expected result	When the shown left minutes approaches 0, a loader spinner appears next to the booking details. The Customer receives a push notification which informs them that the booking is completed. The completed booking shows its new status “Completed” in the top right corner of the related element. In the web app, the related tuple of charging point - socket of the correspondent charging station is now shown as available.

6 INSTALLATION INSTRUCTIONS

The MVP can be accessed with minimal installation requirements because our staging environment is publicly available.

On the other hand it is also possible to deploy and test it locally.

6.1 Accessing the publicly available staging environment

To test the MVP using the staging environment the only requirements are:

- Smartphone Android or iOS
- Web browser

Procedure to access the mobile app:

1) Download Expo Go from the Play Store or from the App store
2) Open the app Expo Go on the smartphone
3) Tap “Scan QR code”
4) Scan the QR code available at QR Code
5) Wait until the app loads

Procedure to access the web app:

1) Visit the link https://dashboard.papaia.smg-team.net/
2) Login credentials:
   mail: juliet@my-cpo.com
   password: password

6.2 Deploying the whole system locally

To deploy the MVP locally the requirements are:

- Smartphone Android or iOS
- Web browser
- Windows 11 (with WSL2 enabled), Linux or Mac machine with at least 8 GB of RAM

Procedure to deploy the cluster locally:

1) Install Docker Desktop following the instructions available at the following links (pick the one associated to your operating system)
   Install on Mac | Docker Documentation
   Install on Windows | Docker Documentation
   Install on Linux | Docker Documentation
2) Open Docker Desktop
3) Go to Settings > Kubernetes > Enable Kubernetes
   This step will deploy a Kubernetes cluster composed only of the host machine
4) Install kubectl, following the instructions available at
   Install and Set Up kubectl on macOS | Kubernetes
   Install and Set Up kubectl on Windows | Kubernetes
   Install and Set Up kubectl on Linux | Kubernetes
5) Install helm, following the instructions available at
   Helm | Installing Helm

6) Make sure to have both kubectl and helm in the PATH variable by typing in a fresh terminal
   “kubectl -- help”
   “helm --help”
   In case the output of the terminal mentions that the commands are not found, make sure to include them in the PATH variable
7) Download the folder /DeliveryFolder/ITD/LocalDeployment of the repository into a folder of the computer and open a terminal session inside it
8) Type the following commands:
   “kubectl create namespace papaia”
   “helm repo add bitnami-repo https://charts.bitnami.com/bitnami”
   “helm install kafka bitnami-repo/kafka -n papaia --create-namespace -f kafka/kafka-helm.yaml”
   “kubectl apply -f microservices-manifests”
   “kubectl apply -f webapp”
9) Run “kubectl get pod -n papaia”
10) Repeat the previous step until all the rows have the value “Running” for the column “Status”. This step might take more than 10 minutes depending on the machine

Procedure to run the mobile app locally:

1) Install nodejs and npm
   https://docs.npmjs.com/downloading-and-installing-node-js-and-npm
2) Make sure to have npm in the PATH variable by typing in a fresh terminal
   “npm -v”
   In case the output of the terminal mentions that the commands are not found, make sure to include npm in the PATH variable
3) Download the whole repository in a folder of the computer
4) Edit the file found in IT/FE/emsp-mobile-app/api/ApiConfig.ts at line 4 writing the *private* ip address of the machine instead of the string ‘https://api.papaia.smg-team.net’, ‘http’ instead of ‘https’ and appending at the end ‘:8080’ .
   The final result should be in this form:
   OpenAPI.BASE = "http://192.168.43.215:8080";
   Instead of 192.168.43.215 insert the *private* ip address of the machine
5) Open a terminal in the folder that contains the repository and change directory to IT/FE/emsp-mobile-app
6) Create an account at Expo
7) Type in the terminal
   “npm install”
   “npm run merge-openapi”
   “npm run generate”
8) Type in the terminal “npx expo login” and follow the instructions presented in the terminal
9) Type in the terminal
   “npx expo start --tunnel -c”
10) A qr code will appear in the terminal
11) Download Expo Go from the Play Store or from the App store on the smartphone

12) Open on the smartphone the app Expo Go
13) Tap “Scan QR code”
14) Scan the QR code that appeared in the terminal
15) Wait until the app loads

Procedure to access the web app:

3) Visit the link http://localhost:80
4) Login credentials:
   mail: juliet@my-cpo.com
   password: password

7 EFFORT SPENT

Section	Total effort spent
Back-end development	80 h
Front-end development	80 h
Deployment procedures	20 h
Documentation	20 h

Team member's contributions:

Brugnano Matilde	100	hours
Buttiglieri Giorgio Natale	100	hours

8 REFERENCES

1. "Assignment RDD AY 2022-2023_v3"
2. "Assignment IT AY 2022-2023"
3. RASD
4. RDD