AY 2022/2023

Politecnico di Milano

ITD: Implementation Document

Marcello De Salvo Riccardo Grossoni
Francesco Dubini

Professor
Elisabetta Di Nitto

Version 1.0
February 1, 2023

Contents

1 Introduction

The code can be found in the official project repository on GitHub at the link: https://github.com/MarcelloDeSalvo/DeSalvoDubiniGrossoni.git.

1.1 Purpose

The objective of this document is the realization of a full technical description of the system presented in the RASD document. Here we will analyze both hardware and software architectures, focussing on the interaction between components that constitute the system. Additionally, we will also delve into the implementation, testing and integration process. This document will use technical language as it's aimed for programmers, but stakeholders are also invited to read as it can be useful to understand the characteristics of the development.

1.2 Definitions, acronyms, abbreviations

Acronyms

- **RASD**: Requirement Analysis and Specification Document
- **DD**: Design Document
- **ITD**: Implementation Document
- **API**: Application Programming Interface
- **DBMS**: Database Management System
- **OCPP**: Open Charge Point Protocol
- **OCPI**: Open Charge Point Interface
- **CPOW**: Charge Point Operator Worker
- **UML**: Unified Modeling Language
- **GPS**: Global Positioning System
- **UI**: User Interface

- **HTTPS**:HyperText Transfer Protocol Security
- **CSRF**: Cross Site Request Forgery
- **HTML**: HyperText Markup Language
- **CSS**: Cascade Style Sheet
- **JS**: JavaScript
- **MVVM**: Model View View-Model
- **MVC**: Model View Controller
- **REST**: Representational State Transfer
- **JSON**: JavaScript Object Notation
- **JWT**: JSON Web Token
- **URL**: Uniform Resource Locator
- **TS**: TypeScript
- **DRF**: Django REST Framework
- **ACID**: Atomicity-Consistency-Isolation-Durability

1.3 Revision history

- Version 1.0: first release

1.4 References

- Django Framework: `https://www.djangoproject.com/`
- REST Framework: `https://www.django-rest-framework.org/`
- Nuxt 3 Framework: `https://nuxt.com`
- Vue.js: `https://vuejs.org/`
- PostgreSQL: `https://www.postgresql.org/docs/14/index.html`

- Vercel: `https://vercel.com/docs`
- Digital Ocean: `https://www.digitalocean.com`
- Tailwindcss: `https://tailwindcss.com/docs`

2 Development

2.1 Implemented Functionalities

We implemented the following functionalities for emsp users and cpo's operators:

User

- Visualization of nearby stations and their status
- Make a booking
- Delete a booking
- Visualization of all bookings
- Start charging session

CPOW

- Access informations on the stations
- Change the power Source of the stations (battery and DSO)
- Add and remove discounts

2.2 Functionalities not implemented

For the eMSP we didn't implement the payment system as it wasn't required and a mail service that notifies the user when the charging process is finished (but it gets notified while inside the page). For the CPMS a lot of functions required a real communication with a DSO and the ChargingStation, so it was not possible to truly implement them. We inserted data relative to the DSOs and the batteries in the database based on our discretion.

2.3 Implemented requirements

Here it follows the list of requirements taken from the RASD. The implemented ones are checked with a checkmark.

Shared requirements

R1. The system must allow registered and logged-in users to use the app. ✓

R2. The system must respect the GDPR and the user's privacy. ✓

R3. The system must allow registered users to log-in using their e-mail. ✓

R4. Both the CMPS and the eMSP subsystem must abide to the OCPI 2.2.1 protocol. ∼

EMSP requirements

R5. The eMSP subsystem must show the user the nearby charging stations through an interactive map. ∼

R6. The eMSP subsystem must be allowed to use the user's GPS location in order to view the nearby charging stations, if given permission. ×

R7. The eMSP subsystem must notify the user when the charge has finished. ✓

R8. The eMSP subsystem must show the user the discounted prices. ✓

R9. The eMSP subsystem must show the user the prices of the charging stations. ✓

R10. The eMSP subsystem must access the User calendar schedule in order to suggest the best charging timeframes, if given permission. ×

R11. The eMSP subsystem must suggest the user on which charging stations to go based on the EV battery and his daily schedule. ×

R12. The eMSP subsystem must communicate with the CPMSs in order to exchange all needed information about the charging stations. ✓

R13. The eMSP subsystem must allow the user to book a charging spot for a future date. ✓

R14. The eMSP subsystem must allow the user to cancel a future reservation for a charging spot. ✓

R15. The eMSP subsystem must allow the user to pay for the charge through an external payment service. ×

R16. The eMSP subsystem must allow the user to start or stop the charge via the application. ✓

R17. The eMSP subsystem must notify the user of the upcoming booked sessions. ×

CPMS requirements

R18. The CPMS subsystem must communicate with the DSO according to a standard protocol. ×

R19. The CPMS subsystem must retrieve the DSO current energy price. ✓

R20. The CPMS subsystem must automatically decide from which DSO to acquire energy. ∼

R21. The CPMS subsystem must retrieve the charging station's battery status. ✓

R22. The CPMS subsystem must dynamically change the energy source depending on the internal station's battery status and the available DSOs' prices. ∼

R23. The CPMS subsystem must communicate the location of the charging stations to all connected eMSPs. ✓

R24. The CPMS subsystem must retrieve the internal status of the sockets through the OCPP standard protocol. ∼

R25. The CPMS subsystem must retrieve and communicate to all connected eMSPs the external status of the sockets through the OCPP and the OCPI standard protocols. ✓

R26. The CPMS subsystem must allow to start or stop a charging session through the OCPP standard protocol. ✓

R27. The CPMS subsystem must retrieve the battery status of the EV through the DIN/ISO specifications. ×

R28. The CPMS subsystem must allow the CPOW to add or change the current special offer. ✓

R29. The CPMS subsystem must allow the CPOW to change the energy provider of a charging station. ✓

R30. The CPMS subsystem must unlock the reserved charging spot if the user doesn't show up. ✓

2.4 Design Choices

By using TailwindCSS, we efficiently created a UI that is both scalable and optimized for mobile devices. However, during the development and testing process, the focus was primarily on the desktop version of the web application.

2.5 Adopted Development Frameworks

Framework selection was based on factors such as ease of use, availability of support, and alignment with commonly used design patterns in current real-world applications. For the front-end we opted for Nuxt 3 and Vue, which utilizes the Model-View-ViewModel (MVVM) design pattern (an evolution of the traditional MVC). For the back-end we chose Django, which enforces a standard Model-View-Controller (MVC) pattern for the API.

MVC

In the MVC pattern, the View handles user requests and returns a response, the Model manages the data access and manipulation logic, and the Controller acts as an intermediary between the Model and View to process user requests and manage the flow of data.

MVVM

As previously said, the MVVM is and extension of MVC mainly used in web development. MVVM separates the different components of the development process into three categories, model, view and ViewModel. The ViewModel serves a similar role as the Controller in MVC, but with some key differences: the ViewModel is responsible for exposing data from the Model in a form

that can be easily consumed by the View thanks to a binder that automates their communication. With respect to the MVC, the MVVM pattern makes it easier to manage and test large applications, but it lacks standardization and increase the complexity of the architecture.

2.6 Programming languages

The programming languages used in the project are Python and JavaScript. For both the backends (CPMS and EMSP) Python was chosen since it is a very versatile language that is easy to learn and use. For the front-end, TypeScript was chosen since it is a superset of JavaScript that adds static typing and object-oriented programming to the language.

2.6.1 Django Framework

We used Django as the backend framework for the CPMS and EMSP subsystems. Django is a widely-used high-level Python web framework that provides a clean and pragmatic design. It has several advantages that make it a great choice for web development projects:

- **Rapid Development:** Django has a lot of built-in functionality, including a powerful admin interface, which makes it possible to develop web applications very quickly.
- **Scalability:** Django is designed to handle high traffic, which makes it a great choice for large-scale projects. It provides a solid foundation for scaling up your application as your needs change.
- **Security:** Django provides robust security features, including protection against cross-site scripting (XSS), cross-site request forgery (CSRF), and SQL injection attacks.
- **Ease of use:** Django has a straightforward syntax and follows the Model-View-Controller (MVC) architectural pattern, making it easy for developers to understand and work with.
- **Large Community:** Django has a large and active community of developers who contribute to the development of the framework, provide support and share best practices.

- **Third-party Packages:** Django has a large number of third-party packages, which makes it easy to add additional functionality to your application, such as authentication, payment processing, and more.

Overall, Django is a robust and reliable framework that provides a lot of built-in functionality, security, scalability, and ease of use, making it a good choice for web development projects.

Django Middlewares

To manage the communication between the two backends and frontend, we mainly use the following Django middlewares:

- *SecurityMiddleware*
- *SessionMiddleware*
- *CsrfViewMiddleware*
- *AuthenticationMiddleware*
- *MessageMiddleware*
- *CorsMiddleware*

2.6.2 Django REST Framework

We decided to pair REST framework to our Django backends since REST allows even more security features. Django Rest Framework (DRF) is a powerful and flexible toolkit for building Web APIs. It is built on top of the Django web framework, which makes it easy to integrate with existing Django applications. DRF provides a number of features that make it a good choice for building RESTful APIs, including:

- **Serialization:** DRF provides a simple and flexible way to serialize and deserialize data, which helps to convert complex data structures into a format that can be easily transmitted over the web.
- **Authentication and Permissions:** DRF provides a range of authentication and permission classes, which can be easily configured to secure your APIs. This allows you to control who has access to your

data and what actions they can perform. We chose *JWTauthentication* and *IsAuthenticated* as the authentication and permission classes.

- **URL Routing:** DRF provides a simple and powerful URL routing mechanism, which makes it easy to map URLs to views and handle different HTTP methods (such as GET, POST, PUT, and DELETE).
- **Extensibility:** DRF is highly extensible, which means that you can easily add custom functionality to meet the specific needs of your project. There are a wide range of third-party packages available that can be used to add additional features to your APIs.

2.6.3 Nuxt 3 Vue

Nuxt.js is a JavaScript framework built on top of the popular Vue.js framework. Nuxt.js is often used to build modern and powerful web applications, including Single Page Applications (SPAs) and Server-Side Rendered (SSR) applications. Some of the benefits of using Nuxt.js include:

- **Easy setup and deployment**: Nuxt.js makes it easy to set up and deploy your applications, with a simple configuration file and easy-to-use command-line tools.
- **Automatic code splitting**: Nuxt.js automatically splits your code into smaller chunks, making it easier for users to load your application quickly.
- **Built-in support for Vue.js**: Nuxt.js is built on top of Vue.js, so you can take advantage of the powerful features and tools that Vue.js provides, such as reactive data bindings and a component-based architecture.
- **Modular and reusable code**: Nuxt.js allows you to easily write modular and reusable code, making it easier to maintain and scale your application.
- **SSR capabilities**: Nuxt.js provides built-in support for server-side rendering, making it easy to create fast and scalable applications that are optimized for search engines.

- **Integration with other tools and technologies**: Nuxt.js can be easily integrated with other tools and technologies, such as APIs and databases, making it a versatile and flexible choice for web development.

2.7 API Integration

Ne abbiamo messe 0 lmao. Qua potremmo discutere il perchè era poco fattibile farle

2.7.1 Maps API

2.7.2 Car APi

Scuffed

2.8 DataBase

For our database we opted for PostgreSQL, a well known object-relational database that provides a reliable, robust and ACID-compliant system and that guarantees high performance and support with Django.

2.9 Vercel

We chose to host our front-end with Vercel, a platform as a service (PaaS) that allows to build, run, and host applications entirely in the cloud for free.

- `de-salvo-dubini-grossoni.vercel.app`

2.10 Digital Ocean

In order to host both the CPMS and the eMSP backends in a simple way, we decided to use Digital Ocean, another platform as a service (PaaS) that allows to host and run django applications and postgres databases on the cloud. We created a domain for each backend system that can be used to make HTTP request and access django rest framework API views:

- `https://sea-lion-app-4fmmp.ondigitalocean.app`
- `https://sea-lion-app-4fmmp.ondigitalocean.app/cpms/`

3 Source Code

3.1 Backend Structure

Spiegazione ella struttura del progetto. Direi di spiegare qua la cosa del doppio backend e magari di stare attenti alle varie env.
TBH la parte sotto si può lasciare cosi com'è, non è che sia una cosa che cambia molto. Il problema forse è il filtro anti plagio.

- **ITD**: da vedere
- **dream_backend**: da vedere
- **__init.py__**: it tells the Python interpreter that the directory is a Python package
- **settings.py**: main setting file for the Django project, used to configure all the applications and middleware, it also handles the database settings
- **urls.py**: URL declarations for the Django project, it contains all the endpoints that the website should have
- **wsgi.py**: entry-point for WSGI-compatible web servers to serve your project, it describes the way in which servers interact with the applications
- **asgi.py**: entry-point for ASGI-compatible web servers to serve your project, ASGI works similar to WSGI but comes with some additional functionality
- **migrations**: Django's way of propagating changes to the models into the database schema, when changes occur this folder is populated with the records of them
- **admin.py**: used for registering the Django models into the Django administration, it allows to display them in the Django admin panel
- **apps.py**: common configuration file for all Django apps, used to configure the attributes of the app

- **models.py**: it defines the structure of the database, it allows the user to create database tables for the app with proper relationships using Python classes. It tells about the actual design, relationships between the data sets and their attribute constraints
- **tests.py**: used to test the overall functionality of the app through unit tests
- **views.py**: provide an interface through which a user interacts with a Django website, it contains the business logic of the app
- **manage.py**: command-line utility for executing Django commands; these includes debugging, deploying and running

3.1.1 Apps

Lista delle varie app e funzionalità (opterei per metterla in una subsubsection)

- **app**: implements common functionalities that cover the entire application, it has one endpoint:
  - **farms_list**: handled by *FarmView* view, it manages GET requests by returning a list of farms

3.2 Frontend Structure

Spiegare la struttura della repo del frontend, simile a quella di vercel

4 Testing

AHAHAAHAHAHA

4.1 Unit Testing

Segue una eterna lista di test che non abbiamo fatto

4.2 System Testing

Testing as a whole done throughout

4.3 Post-deployment Testing

Testing post deployment

5 Installation

As a web application we chose to deploy it on Heroku, so a fully running version of the software is available at:
`https://de-salvo-dubini-grossoni.vercel.app`.

Spiegare nel caso di seguire il readme

5.1 Requirements

dire che server python di una certa versione, abbastanza copiabile dal Loro

5.2 Installation

Riscrivere a roba già detta nel readme

5.2.1 EMSP backend

5.2.2 CPMS backend

5.2.3 Frontend

6 Effort Spent

Student	Time for implementation
Marcello De Salvo	20000h
Francesco Dubini	0 h (licenziato)
Riccardo Grossoni	3 KW/h

7 References

References

[1] MDN Web Docs Glossary: Definitions of Web-related terms -> MVC `https://developer.mozilla.org/en-US/docs/Glossary/MVC`