POILTECNICO

MILANO 1863

e-MALL

e-Mobility for All

DD

Design Document

Sara Limooee, 10886949
Parham Ebadi, 10870289

Fall 2022

Table of Contents

1. Introduction

1.1. Purpose

In recent years, due to world warming and the increase in the number of pollutions in the air produced by fuel vehicles because of using petrol, gasoline, etc., there is a huge need for models of cars which must be less disastrous for the environment. Hybrid electric vehicles (HEVs) and electric vehicles (EVs) use less or even no fuels so they have much less effect on the environment. However, these kinds of cars need to be charged whenever their battery is low. The idea is to develop an electric Mobility Service Provider(eMSP) so that users who are EV drivers can easily make a decision among various charging points and book a place for charging their electric vehicles based on some factors e.g., distance, price, etc.

The goal of this document is to provide more technical information about the eMSP application and its interaction with different components of the eMSP system and the CPMS system of charging stations. In fact, developers of the application use this document as a guide to developing the application. In general, the main different features of this document are:

- The high-level architecture
- Main components of the system
- Interfaces provided by the components
- Design patterns adopted
- Implementation, integration, and testing

1.2. Scope

In order to help EV-drivers to find nearby charging stations for their electric car and know about any special offer that either the eMSP or the CPMS (means any offer or suggestions that the CPOs give to the customers), The application has the following parts:

1. EV-driver login to the eMSP system and inserts data about his/her car. According to this information, he can search for charging stations by various filters e.g., distance, price, type of sockets, etc. He can also pay for the obtained service through the interface provided by the eMSP.
2. CPOs log in to the CMPS system to control the general internal and external status of their charging stations, the current price of energy obtained from each DSO that the charging station is working with and to control some other functionalities that can be also controlled by humans manually.
3. eMSP system interacts with the CPMS system to start the charging process, shows the charging process to the end user (EV-driver), giving information about the available charging stations and free sockets of any type of socket (slow, fast, rapid) and their prices and generally any information that must be changed between eMSP and CPMS.

EV-drivers might receive some special offers either from eMSP system for using the application or from CPMSs. Moreover, some suggestions can be given to EV-drivers about when to charge their car based on the previous charging and the distance that the car has passed (this information can be obtained by an API between the CPMS and the navigation system of the car in order to estimate how much distance a car can go at most). Since this project has been done by a group of two students, the last point mentioned above, the suggestions given by CPMS to the EV-drivers have not been considered in the project.

1.3. Definitions, Acronyms and Abbreviations

1.3.1. Definitions

Definitions	Descriptions
External Status	number of charging sockets available, their type, cost, and estimated amount of time until the first occupied socket is freed
Internal Status	amount of energy available in its batteries, number of vehicles being charged, amount of power absorbed, and remaining time of the charge of each vehicle
Notification	A message shown to the user by the system when he/she must be notified about something (ex: when a new offer is available, or when the charging process starts or finishes)
API	Stands for Application Programming Interface. APIs are mechanisms that enable two software components to communicate with each other using a set of definitions and protocols.

1.3.2. Acronyms

Acronyms	Descriptions
eMPS	e-Mobility Service Provider
DD	Design Document
CPMS	Charge Point Management System
DSO	Distribution System Operators
API	Application Programming Interface
TLS	Transport Layer Security
HTTP	Hypertext Transfer Protocol
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
SQL	Structured Query Language

1.3.3. Abbreviations

Abbreviations	Descriptions
G	Goal
R	Requirement
C	Component

1.4. Revision History

Version	Date	Modifications
1.0	04/01/2023	First version
2.0	08/01/2023	Final version Edited: • Component view • Deployment View • Effort spent

1.5. Reference Documents

- Specification Document: "Assignment RDD AY 2022-2023_v2.pdf"
- Course slides

1.6. Document Structure

- Section 1
  In this section, a brief description of the design document is given and the purpose and the scope are introduced. This section also includes definitions, acronyms and abbreviations.

- Section 2
  This part is the fundamental part of the design document which includes the component diagram and interfaces, deployment view, runtime view, and architectural patterns chosen with the other design decisions.

- Section 3
  This section includes the user interface mockups with more details and is more complete than the mockups in RASD document.

- Section 4
  This section contains the traceability matrix that shows which components satisfy certain requirements.

- Section 5
  This section includes the implementation plan, integration plan, and testing plan, and also shows how these plans are executed.

- Section 6

The amount of time spent for each section and each member of the group is included in this section.

2. Architectural Design

2.1. Overview: High-level Components and their Interactions

In this project, as discussed in the RASD document, there are two individual systems in the eMALL, eMSP and CPMS systems. Each of these systems follows the three-tier pattern with a presentation layer, a business logic layer, and a data layer(tier)[1].

- Presentation tier

The presentation tier is the user interface and communication layer of the application, where the end-user interacts with the application. Its main purpose is to display information to and collect information from the user. This top-level tier can run on a web browser, as desktop application, or a graphical user interface (GUI), for example, Web presentation tiers are usually developed using HTML, CSS, and JavaScript. Desktop applications can be written in a variety of languages depending on the platform.

- Application tier

The application tier, also known as the logic tier or middle tier, is the heart of the application. In this tier, information collected in the presentation tier is processed - sometimes against other information in the data tier - using business logic, a specific set of business rules. The application tier can also add, delete or modify data in the data tier.
The application tier is typically developed using Python, Java, Perl, PHP or Ruby, and communicates with the data tier using API calls.

- Data tier

The data tier, sometimes called database tier, data access tier or back-end, is where the information processed by the application is stored and managed. This can be a relational database management system such as PostgreSQL, MySQL, MariaDB, Oracle, DB2, Informix or Microsoft SQL Server, or in a NoSQL Database server such as Cassandra, CouchDB or MongoDB.

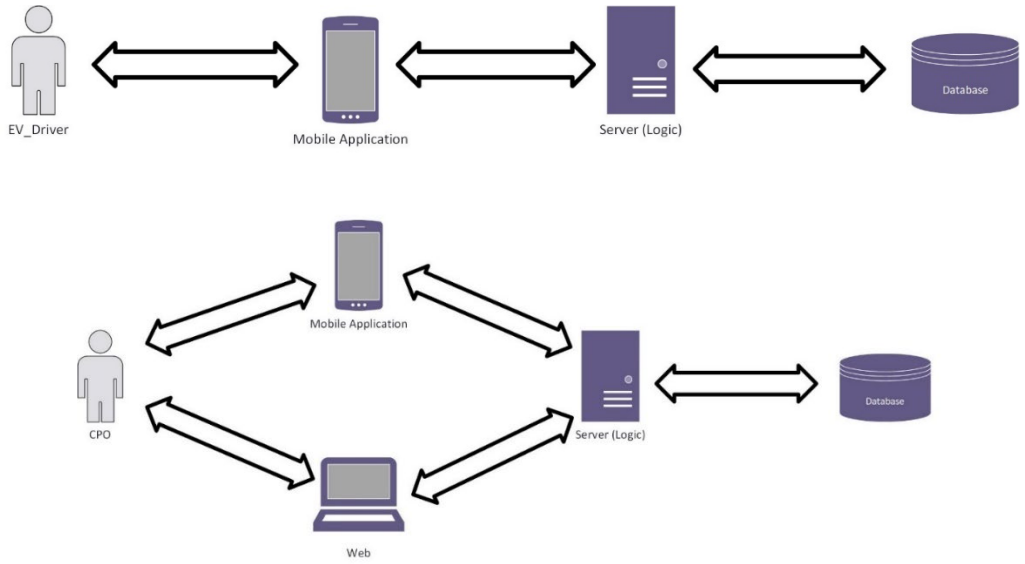

Figure-1 (Three tire architecture)

[1] Layer and tier words can be used interchangeably.

2.2. Component View

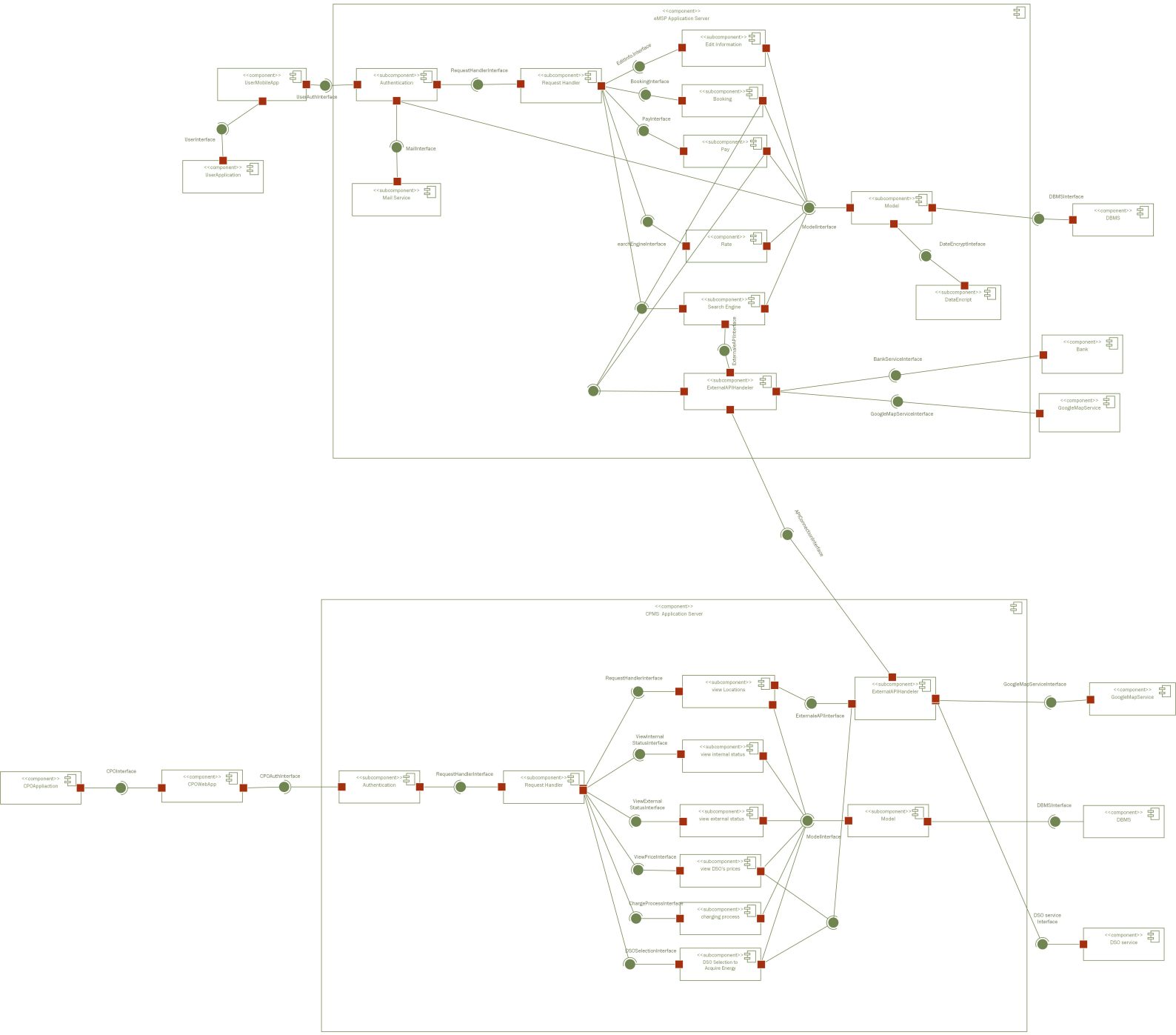

Figure-2 (Component view)

eMSP Application components:

- User Mobile App
  Subcomponent in charge of interacting with the user and sending their requests to the eMSP application server.

- Authentication
  For verifying EV-drivers and allowing them to log in.

- Mail Service
  This component is in charge of sending a verification email to clients (EV-drivers) for verifying their accounts.

- Request Handler
  The request of clients will be redirected to the related component to be handled and responded.

- Booking
  Subcomponent which is in charge of checking booking requests and notifying CPMSs about the new booking.

- Pay
  This subcomponent handles payments for the service that the EV-drivers receive. They can either pay with their credit card or from the internal wallet of their account.

- Search Engine
  Subcomponent which is in charge of searching for charging stations based on different factors, distance, rate, socket type and etc. This component interacts with Google Map Service Component to get the details about the current location.

- External API Handler
  Subcomponent is responsible for interacting with other components out of eMSP application server.

- Model
  Subcomponent with an object-relation mapping (ORM) for retrieving data from the database and responsible for communicating with the data tier (DBMS).

- Data Encrypt
  Subcomponent for encrypting and decrypting data when working with DBMS since the data is encrypted every time.

- DBMS
  This component is an external component which is responsible for storing data.

- Google Map Service
  Component for gathering details of each location and charging station.

- Bank Service

Component for doing payments and adding money to internal wallet and in interacting with bank system.

CPMS application components:

- CPO Mobile App
  Subcomponent in charge of interacting with CPO and sending their requests to the CPMS application server.

- Authentication
  For verifying CPOs and allowing them to log in.

- View Internal/External Status, View Stations' Locations
  Subcomponents are responsible for viewing the internal/external status of stations and their locations.

- Charging Process
  Subcomponent which is in charge of starting the charging process and deciding the amount of power to be used while charging each vehicle and finishing the charging process when the vehicle is fully charged.

- DSO Selection to Acquire Energy from
  Subcomponent which is in charge of selecting the best DSO to acquire energy from.

- DSO Service
  The external component is in charge of replying to messages of CPMS.

- External API Handler
  Subcomponent is responsible for interacting with other components out of the CPMS application server.

- Model
  Subcomponent with an object-relation mapping (ORM) for retrieving data from the database and responsible for communicating with the data tier (DBMS).

- Data Encrypt
  Subcomponent for encrypting and decrypting data when working with DBMS since the data is encrypted every time.

- DBMS
  This component is an external component that is responsible for storing data.

- Google Map Service
  Component for gathering details of each location and charging station.

2.3. Deployment View

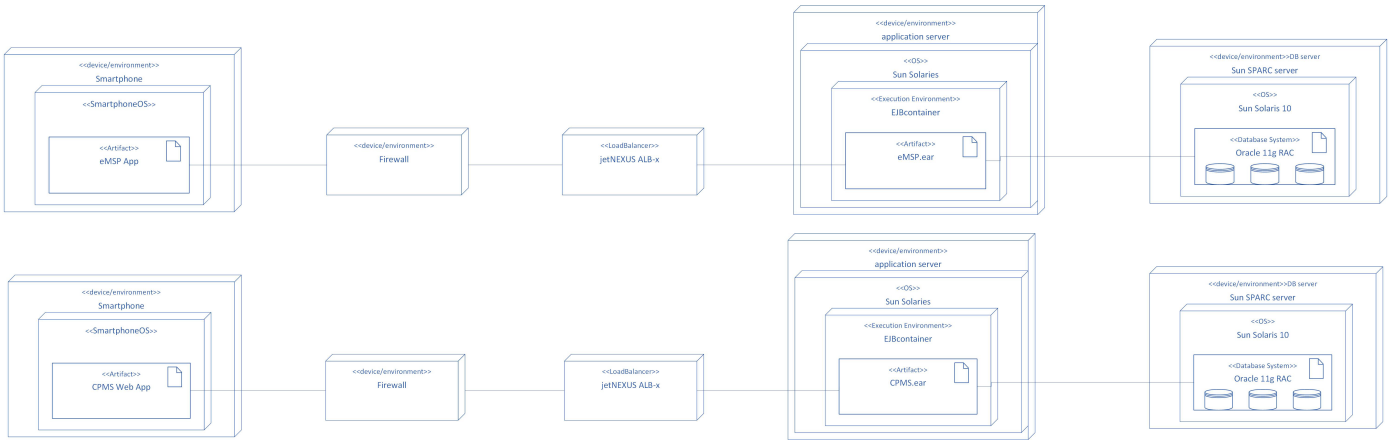

Figure-3 (Deployment view)

• Smartphone: The user (EV-driver or CPO) can use their smartphones to access the systems (EV-driver accesses the eMSP, CPO accesses the CPMS). The two systems have presented many functions that can be done by users and CPOs. For instance, the EV-driver can use the eMSP to add his/her car's specifications, book an appointment, search for stations, add a wallet, pay, etc. On the other hand, CPOs can use the CPMS in different ways, such as to view the internal and external status of the stations and their locations of the stations.

• Computer: The computer device is only specified for CPOs and not for the users. CPOs can use web applications by computer or other devices that contain web applications. The web application also includes all features that mobile application has.

• Firewall: In order to provide safe access to the system and protect data in both systems, there is a strong firewall to protect the server and database as well as prevent external attacks.

• Load balancer: To increase the speed, capacity, and reliability of the applications while trying to avoid the overload of a single server, we used a load balancer that distributes the workloads across multiple servers.

• Web Server: The responsibility of the web server is to receive queries from clients from web applications and smartphones and transfer them to the application server for processing by using java RMI.

- **Application Server:** The logic of the applications is located in this part. The server handles all the requests and provides the appropriate answers for all the offers.

- **Database Management System (DBMS):** For performing the job of DBMS, the SQL database is selected. SQL can manage several data transactions simultaneously where large volumes of data are written concurrently. As an SQL query is written and run, it is processed by the 'query language processor' having a parser and query optimizer. The SQL server then compiles the processed query in three stages:
  1. Parsing: This refers to a process that cross-checks the syntax of the query.

  2. Binding: This step involves verifying query semantics before executing it.

  3. Optimization: The final step generates the query execution plan. The objective here is to identify an efficient query execution plan that runs in minimal time. This implies that the shorter the response time for the SQL query, the better the results. Several combinations of plans are generated to have a practical end execution plan.

2.4. Runtime View

In the following sequence diagrams, we discuss the detailed interactions between the user and each component. The main interactions between the user and the system have been already discussed in RASD. In order to keep the diagrams readable, only the main flow of the events is visualized and exceptions are not considered.

2.4.1. EV-Driver Register

EV-driver enters personal information (phone number, email, first name and last name, password) through the Client application component (EMSP). Then the information is sent to the Authentication component. Now, it is time to check if another EV-driver with the same phone number or email address has already signed up for the application or not. This is checked in the Model component. So, the Authentication component sends the information to the Model component. The model sends a query to DBMS. The answer of DBMS is encrypted for security purposes. So, there is a DataEncrypt component that decrypts the reply from DBMS. If the retrieved data is null, it means there is no user with entered phone number and email address. Then the final step is to send an email through MailService component to complete the registration. When the user clicks on the confirmation link, he/she will be redirected to a web page. Finally, a query will be sent to the database to update this user flag to True.

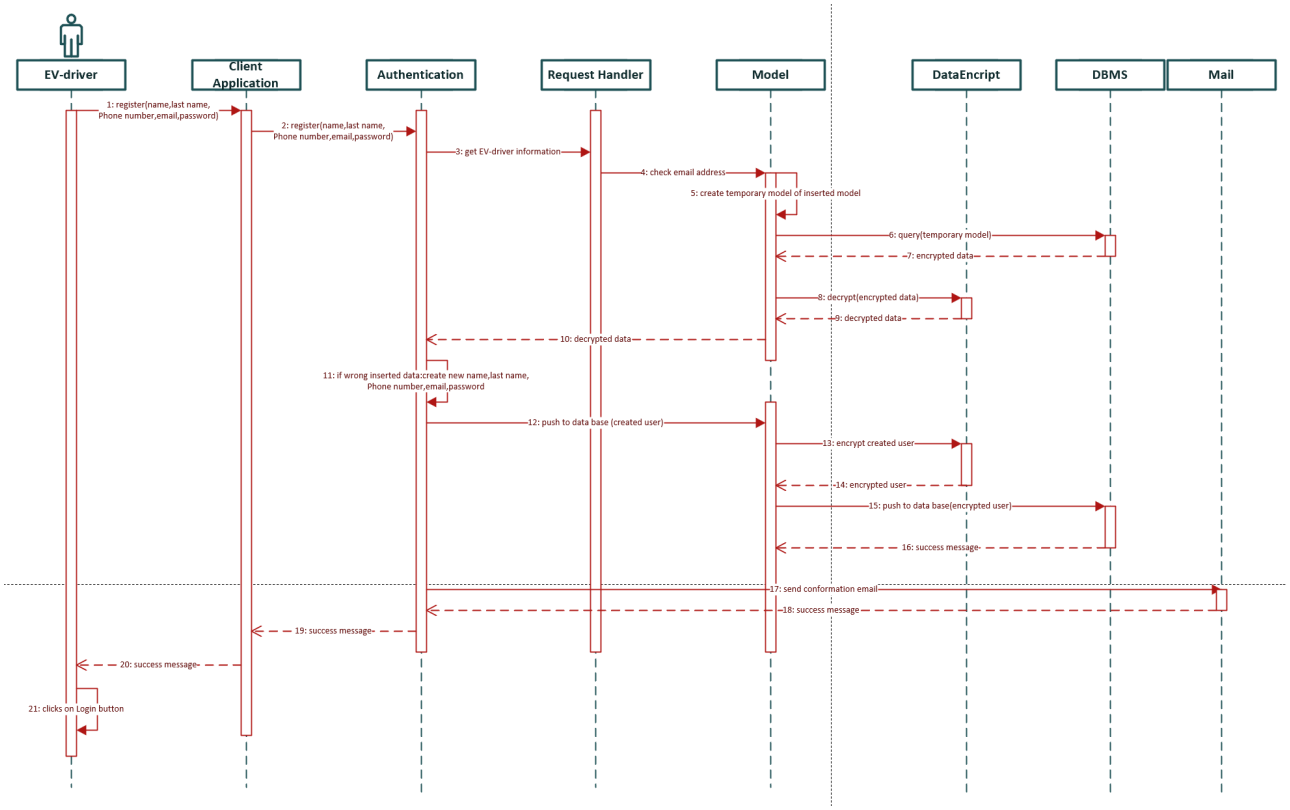

Figure-4 (Runtime view: EV-driver registration)

2.4.2. Login EV-driver

EV-driver enters email and password through eMSP component. Then, his/her information is sent to the Authentication component. Now, it should be checked if the user has registered before or not. This is checked in the Model component. So, the Authentication component sends the information to the Model component. The model sends a query to DBMS to check if the email has registered in the app before or not. The answer of DBMS is encrypted for security purposes. So, there is a DataEncrypt component that decrypts the reply from DBMS. If the retrieved data is null, it means there is no user with the entered email. Otherwise, the user exists. Now, the Authentication component should check if the entered password is the same as decrypted password retrieved from DBMS or not. If the check result is positive, the retrieved user is returned, otherwise a null value is returned.

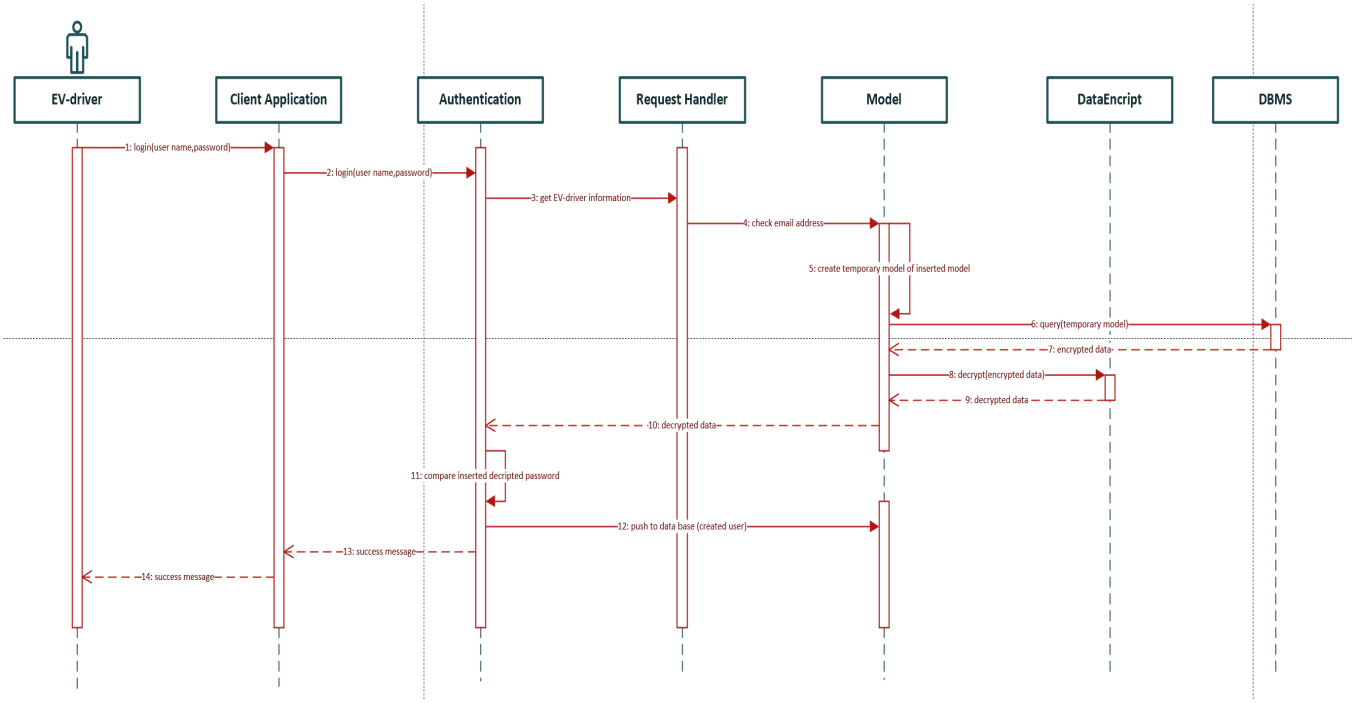

Figure-5 (Runtime view: EV-driver login)

2.4.3. Add vehicle's specifications

EV-driver enters the vehicle's specifications in the client application (EMSP) component. The request is then sent to the request handler component to check to which component should the request be sent for being handled. Then the request and vehicle specifications are sent to Modify Information component. Then this component sends the information to the Model. The model sends a query to the database to update the vehicle specifications of this user. Finally, DBMS sends an encrypted result which should be first decrypted through DataEncrypt component. If the retrieved data from DataEncrypt component is not null, it means that new vehicle specifications have been added to DBMS successfully.

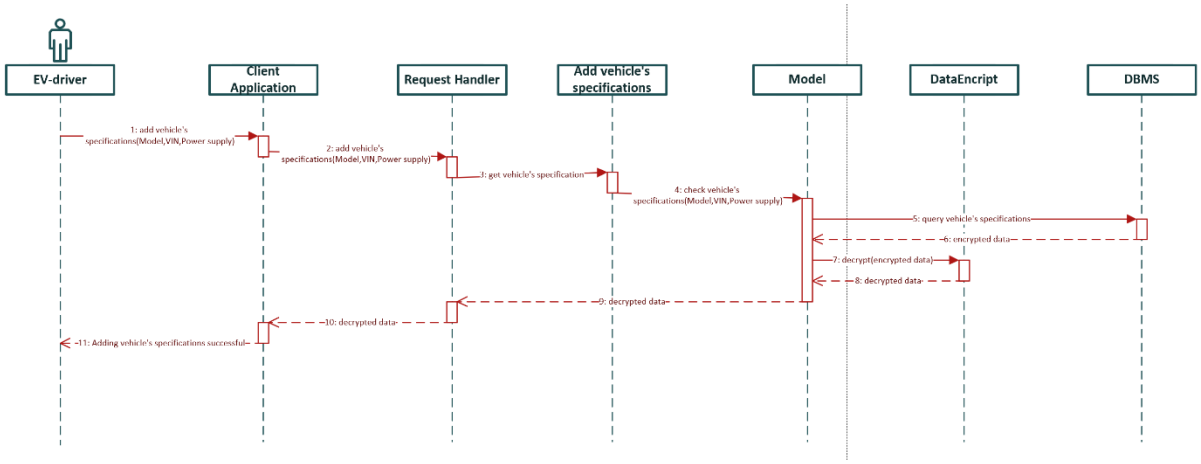

Figure-6 (Runtime view: EV-driver add vehicles specifications)

2.4.4. Edit personal information

This diagram is too similar to the Add vehicle specification since both tasks are in relation to EV-driver's information and can be done in the same component named Modify Information. EV-driver new personal information in the client application (EMSP) component. When EV-driver clicks on the Save button, the request is then sent to the request handler component to check to which component should the request be sent for being handled. Then the request and new user data are sent to Modify Information component. Then this component sends the information to the Model. The Model sends a query to the database to update the user's personal information. Finally, DBMS sends an encrypted result which should be first decrypted through the DataEncrypt component. If the retrieved data from the DataEncrypt component is not null, it means that the new personal information of the named user has been added to DBMS successfully.

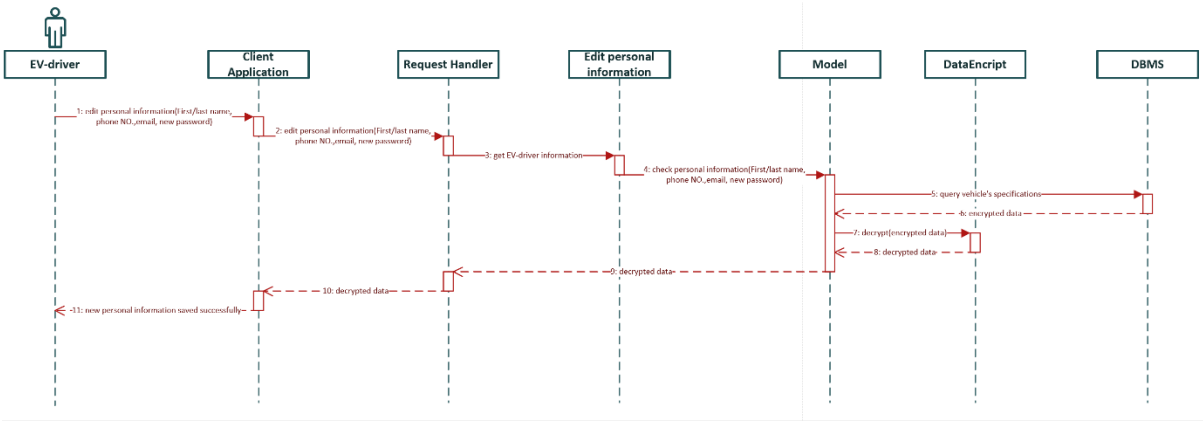

Figure-7 (Runtime view: EV-driver edit personal information)

2.4.5. Adding money to wallet

EV-driver clicks to add money to the internal wallet in the app. So, the request will be handled by the Add Money to Wallet component. This component sends a request to the external API Handler to connect to the Bank system and do the payment in order to increase the internal credit.

If the transaction is successful, it will be inserted in the database as an increased credit action and the success message will be sent back to the user interface.

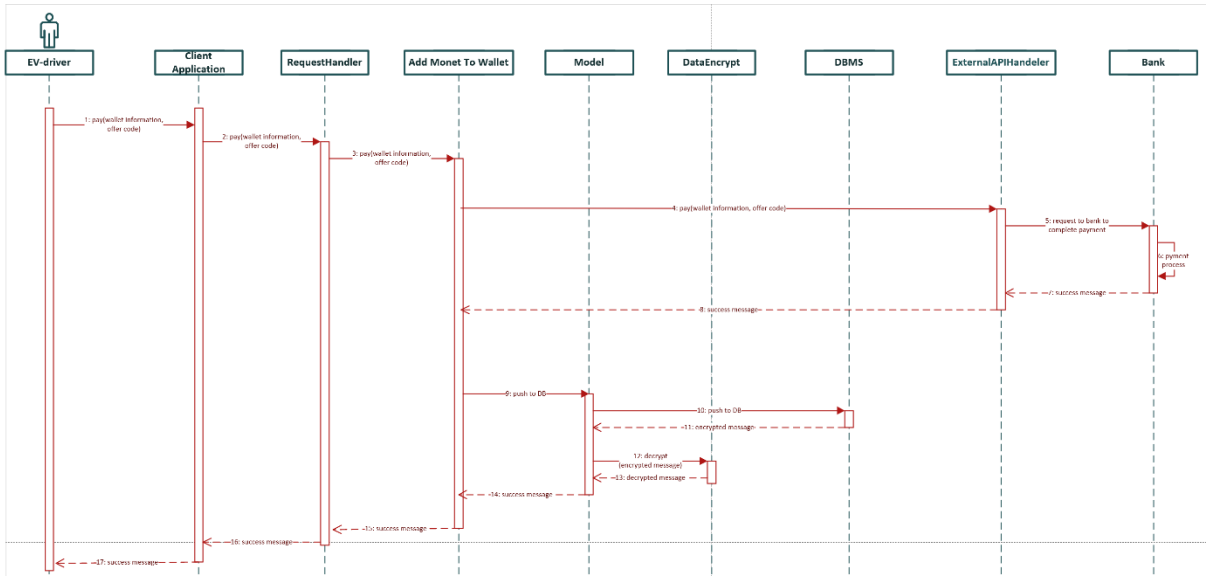

Figure-8 (Runtime view: EV-driver add money to wallet)

2.4.6. Search

EV-driver clicks view charging stations on the client application component. On the first page, the nearby charging stations are shown to the user. So, the client application sends the request to “Request Handler” component. Request Handler sends the request to of searching based on the distance to the Search Engine component. Now, Search Engine must first get the current location of EV-driver. So, it sends a request to the Google Map Service component to get the current location. Then, it is time to check the DBMS for nearby charging stations based on location details received from Google Map Service component. So, the search details along with the search request are sent to the Model component. It sends a query to DBMS. DBMS sends the result encrypted. So, we use the DataEncrypt to decrypt the result of DBMS. Finally, the retrieved decrypted data is the list of nearby charging stations that will be shown to the user.

Now, it is time to search for charging stations based on other factors e.g. rate, price, type of socket and etc. Therefore, the above steps must be repeated here again but the search details can be other factors. So, the only thing that changes here is the arguments of the request sent to the Model component for searching in the DBMS.

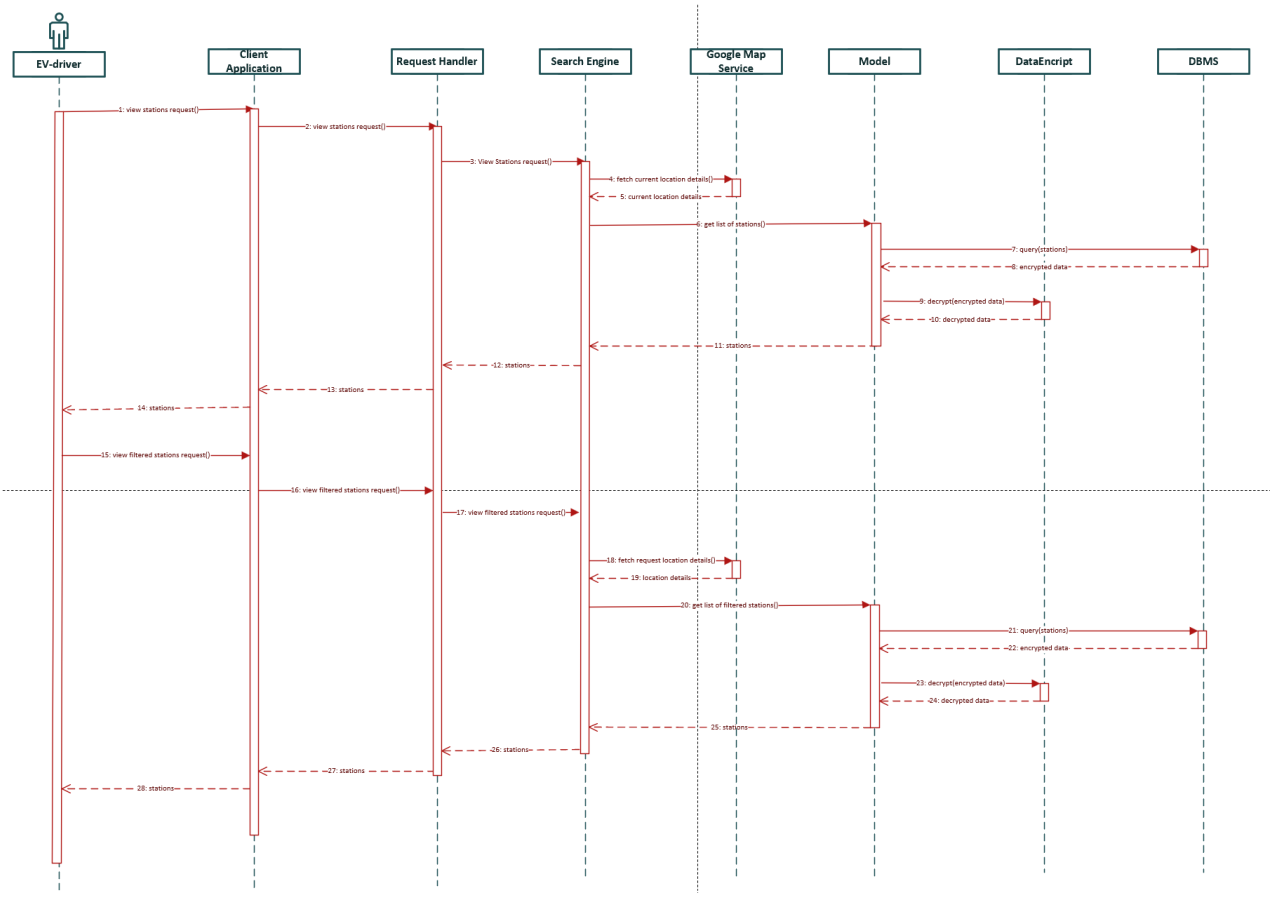

Figure-9 (Runtime view: EV-driver search for charging stations)

2.4.7. Book

EV-driver clicks on the book button after choosing the station. Then he/she must choose the vehicle to be charged at the station. So the list of all vehicles should be fetched from DBMS through the Model component. Then another request is sent for booking to book a time slot for a specific vehicle. If the booking is successful, which means that the time is empty and the socket type is available, a confirmation message is sent back to the EV-driver, while a notification is sent to the CPMS through the external API Handler component so that CPMS can notify other eMSPs about the fact that this specific socket in that station is booked for a specific period of time.

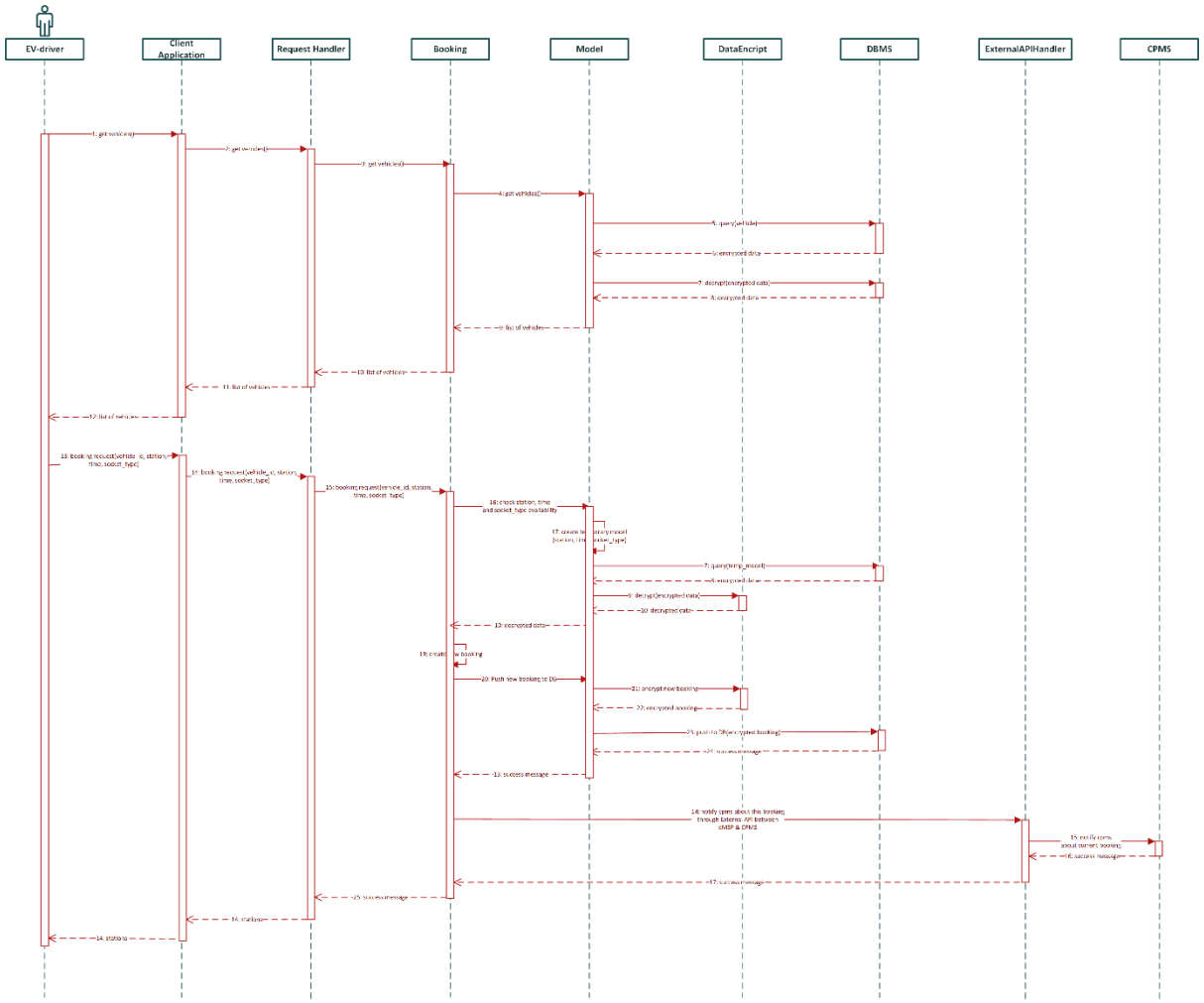

Figure-10 (Runtime view: EV-driver book)

2.4.8. Pay

EV-driver clicks to pay after the charging process is finished. He/she can choose to pay with a credit card or pay from the internal wallet credit. In case the payment method is by credit card, a request must be sent to the bank to do the payment with payment details (Card number, CVC, etc.). Finally, if the payment is successful, a success message is sent back to the user interface.
Otherwise, if the payment method is using the internal wallet in the app, it just updates the amount of credit in the wallet in DBMS through the Model component which is for interacting with data tire.

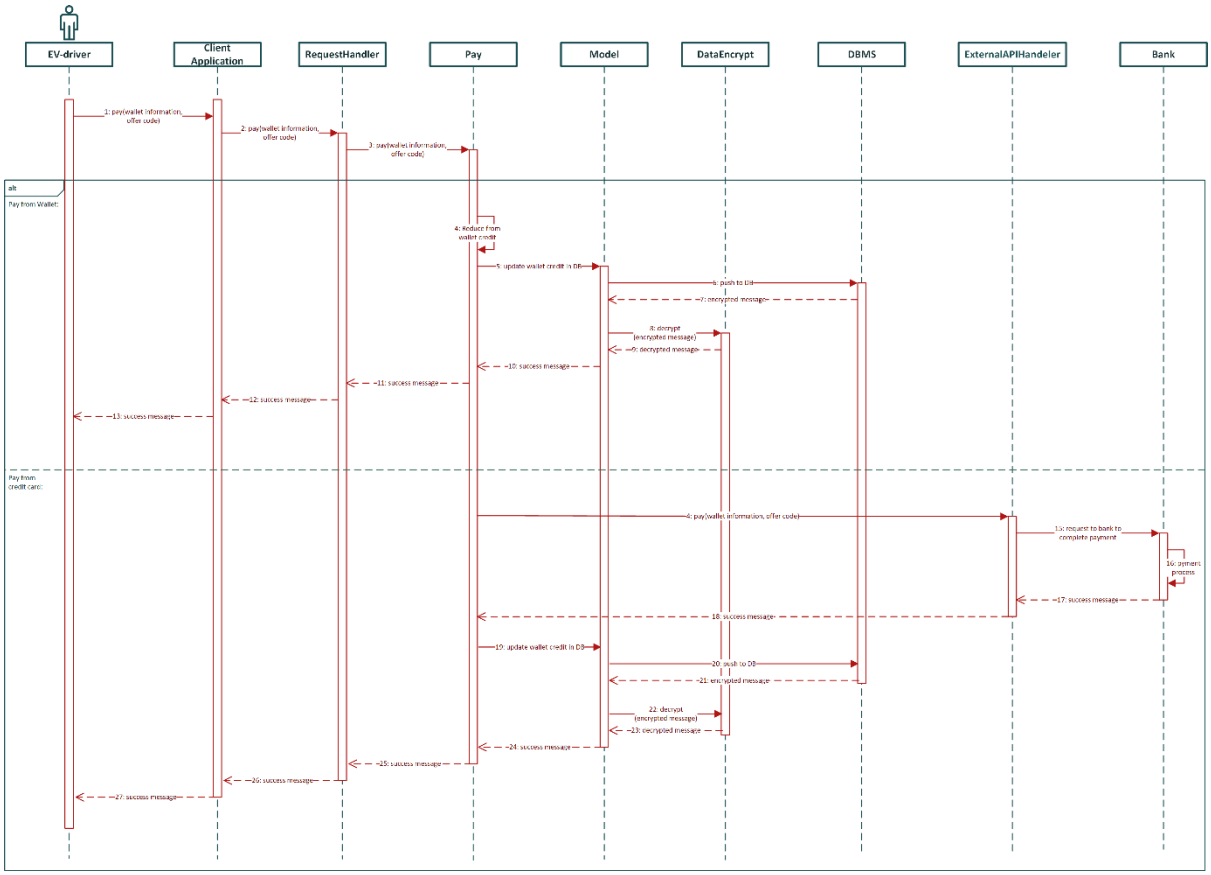

Figure-11 (Runtime view: EV-driver pay)

2.4.9. Rate

EV-driver clicks to rate for the charging station which the user obtained service after payment through the client application component. The request which includes the score and the reason and any other requires argument is then sent to the Request Handler component. Then the request is sent to the related component which is Rate. The score must be inserted in the DBMS. So, the data should be sent to Model component. The Model send a query to DBMS. The reply from DBMS is encrypted and should be decrypted using DataEncrypt component. If the result is not null, it means that the score has been inserted in the database successfully and a message is then shown back to the EV-driver.

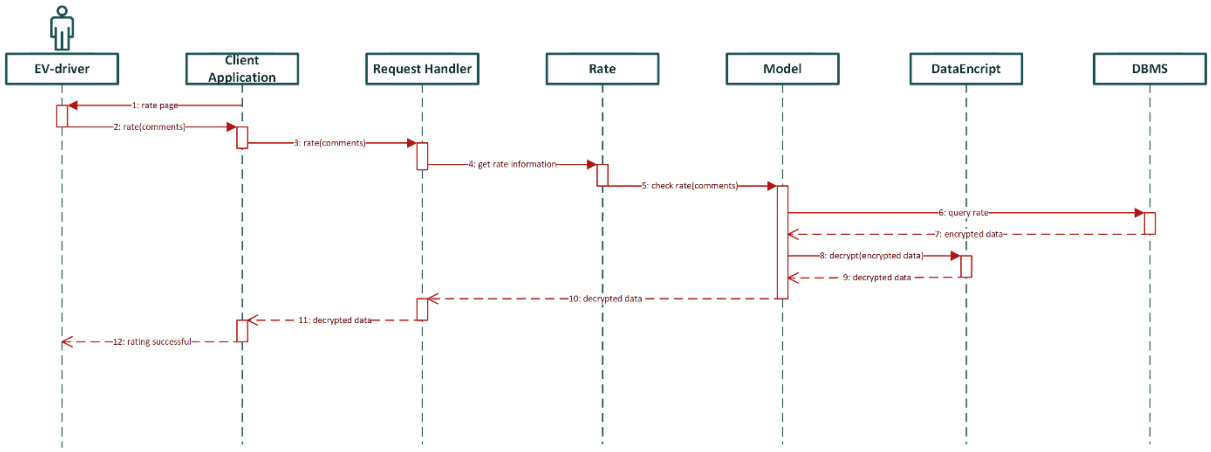

Figure-12 (Runtime view: EV-driver rate)

2.4.10. CPO login

CPO enters email and password through CPMS application component. Then, his/her information is sent to the Authentication component. Now, it should be checked if the information entered is correct or not. First, the email must be checked to see if it exists in the DBMS or not. Then the password must be checked. This is checked in the Model component. So, the Authentication component sends the information to the Model component. The model sends a query to DBMS to check if the email exists in the app before or not. The answer of DBMS is encrypted for security purposes. So, there is a DataEncrypt component which decrypts the reply from DBMS. If the retrieved data is null, it means there is no user with the entered email. Otherwise, the user exists. Now, the Authentication component should check if the entered password is the same as decrypted password retrieved from DBMS or not. If the check result is positive, the retrieved user is returned, otherwise a null value is returned.

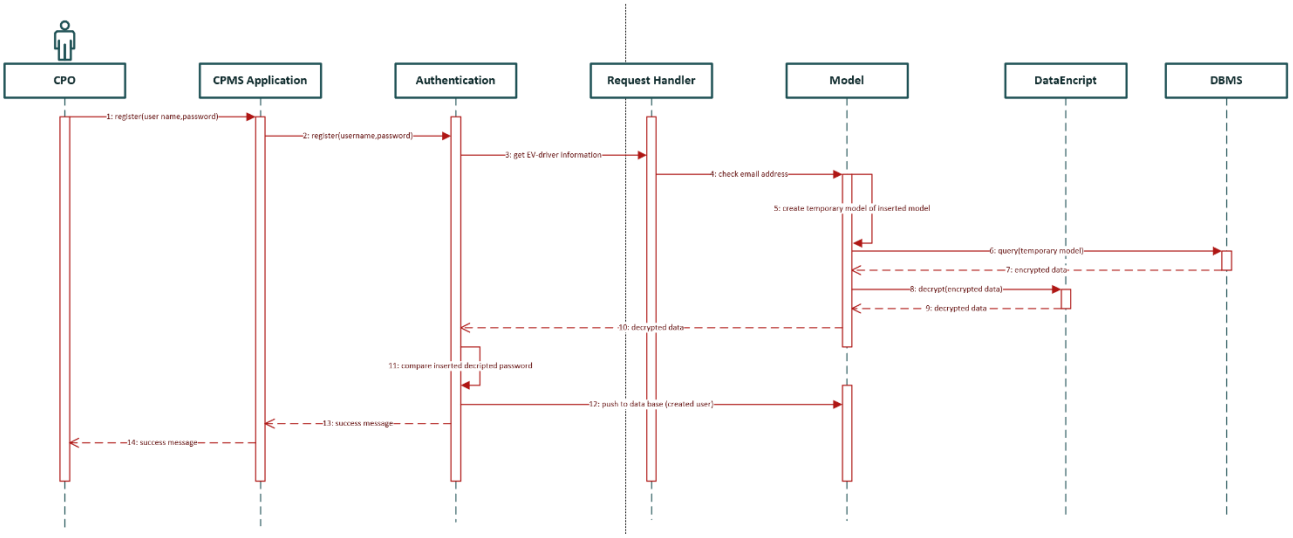

Figure-13 (Runtime view: CPO login)

2.4.11. CPO view charging stations' locations

EV-driver clicks to view the locations and details of charging stations through CPMS Application component. The request is then sent to Request Handler component to be sent to the related components for being handled. The related components send a request to database since all of this information must be got from DBMS. However, a request is sent to Model first. Then Model component sends a query to DBMS to get the related information. The data received from DBMS is encrypted and must be decrypted using DataEncrypt component. The data retrieved from DataEncrypt component is the information that must be shown to the CPO through CPMS application component. Therefore, the data is sent back again to all components until CPMS application component. These explanations are the same for the following 2 diagram, viewing internal and external status.

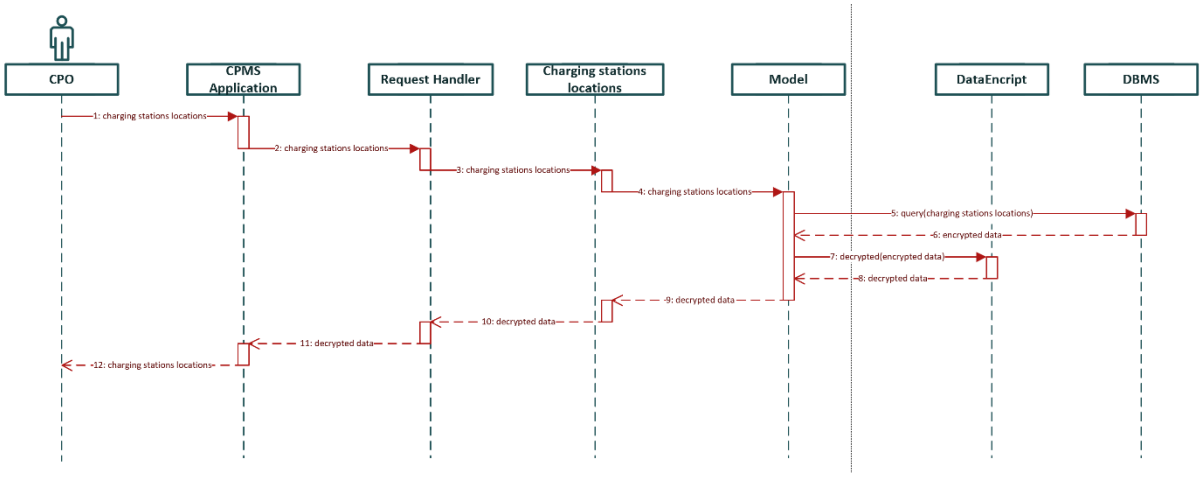

Figure-14 (Runtime view: CPO view stations' locations)

2.4.12. CPO views internal status

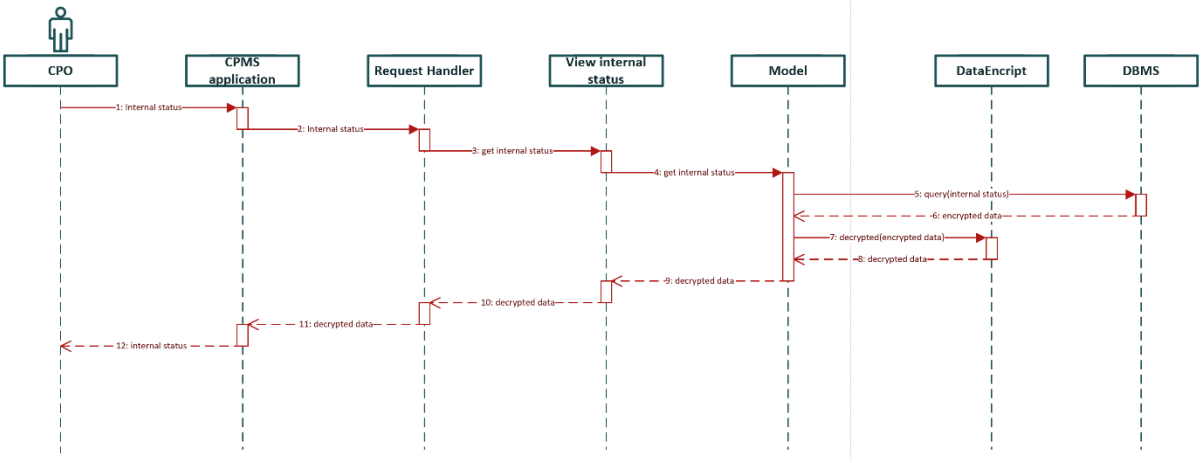

Figure-15 (Runtime view: CPO view internal status)

2.4.13. CPO view external status

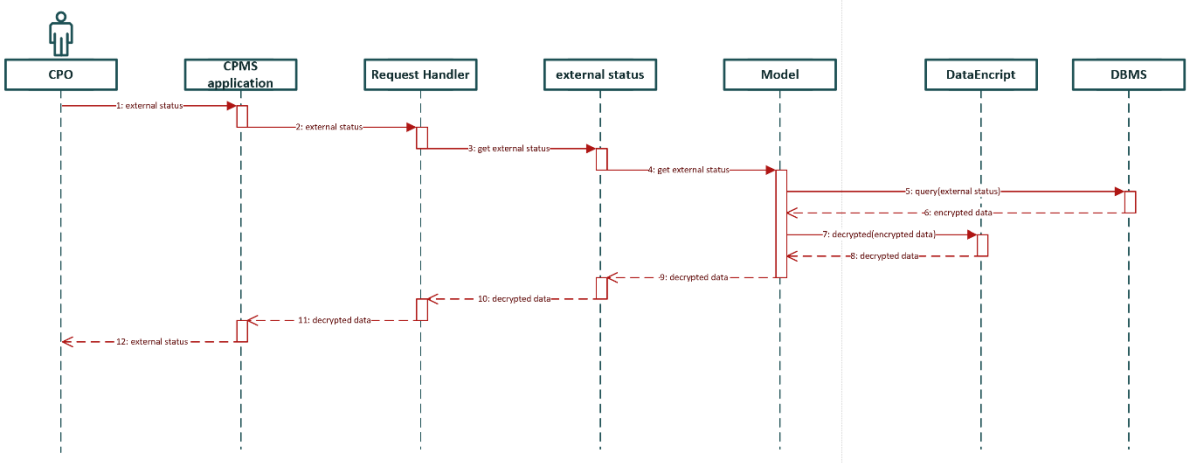

Figure-16 (Runtime view: CPO view external status)

2.5. Component Interface

In the following section, there is a diagram represented in which the interfaces that are offered by the different components, are shown. These interfaces are based on the most important processes which are mentioned in the runtime view section.

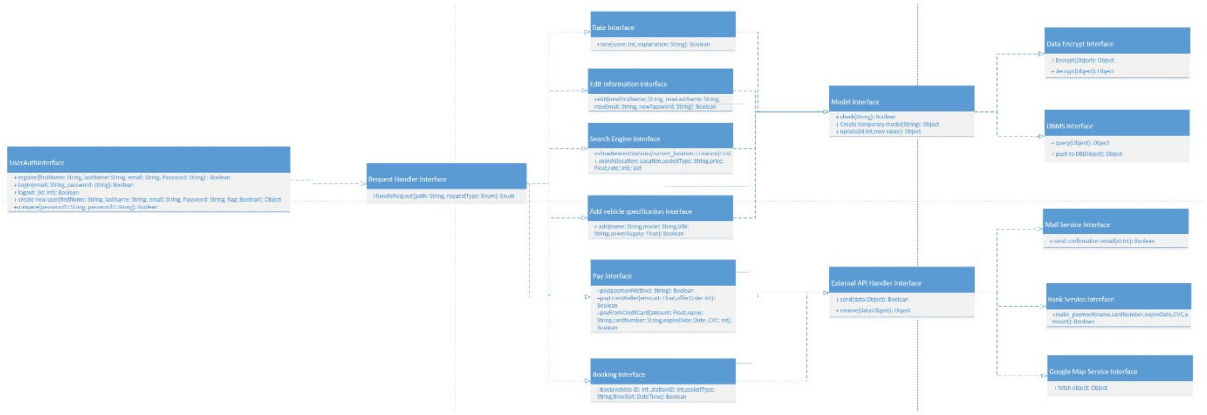

Figure-17 (eMSP Component interface)

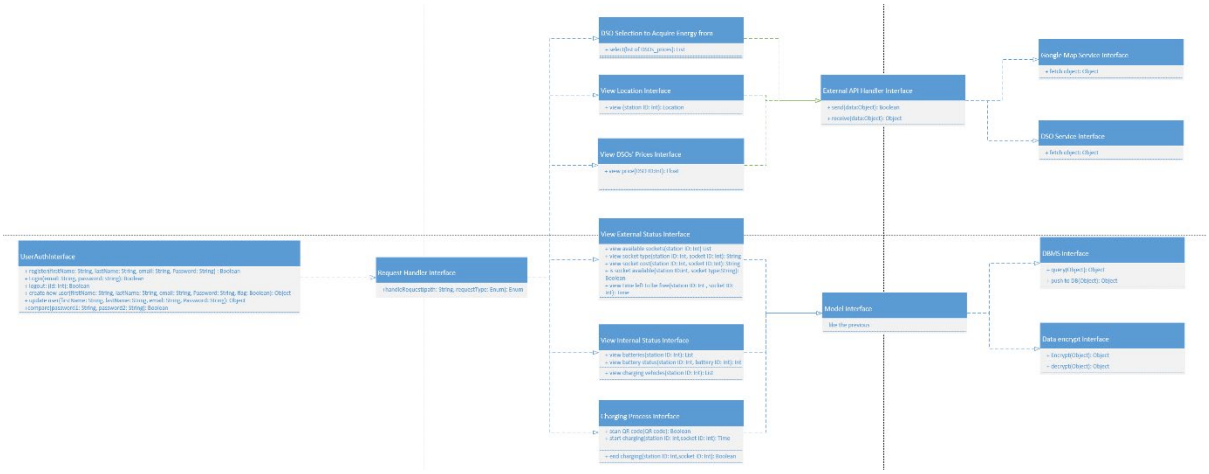

Figure-18 (CPMSComponent interface)

2.6. Selected Architectural Styles and Patterns

In this project, as we have two applications (eMSP and CPMS), we used a three-tier client-server architecture: a presentation tier, an application tier, and a data tier. Each of them was implemented both in eMSP and CPMS systems. By dividing the architecture into these three parts, we can expect increases in flexibility, reusability and scalability.

According to the fact that all data which is being transferred between two systems are sensitive and should be protected, TLS is used to guarantee the security and reliability of the connections. Furthermore, in order to exchange messages on the world wide web, we used HTTP. In this way it is possible to reduce the coupling between the client and server on both sides.

On the other hand, using the cache in the client tier seems necessary because allows the system to speed up the whole system by omitting some interactions between client and server.

For transmitting data in a web application, JSON is selected to be used for its simplicity. It is less verbose and this fact makes it more readable and allows much faster parsing. The JSON file is transmitted over the network via XMLHTTPRequest.

The REST is used for using the GoogleMaps service. GoogleMaps is considered an external component in our system, so this protocol is suitable for that. It also allows customizing the requested content.

In the CPMS system, for receiving data from DSOs or CPOs or sending information to eMSP, SOAP protocol is used to exchange the XML messages. SOAP perfectly fits into the world of internet applications and promises to improve internet inter-operability for application services. In essence, SOAP packages method calls into XML strings and delivers them into component instances through HTTP.

Model-View-Controller (MVC)

The Model-View-Controller (MVC) is an architectural pattern that separates an application into three main logical components: the model, the view, and the controller. Each of these components is built to handle specific development aspects of an application. 1. Model: The Model component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data. For example, a customer object will retrieve the customer information from the database, manipulate it and update its data back to the database or use it to render data. 2. View: The View component is used for all the UI logic of the application. For example, the customer view will include all the UI components such as text boxes, dropdowns, etc. that the final user interacts with. 3. Controller: Controllers act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output. For example, the Customer controller will handle all the interactions and inputs from the Customer View and update the database using the Customer Model. The same controller will be used to view the Customer data.

2.7. Other Design Decisions

As mentioned in the previous sections, two databases have been represented for both eMSP and CPMS systems. This role is performed by SQL database. As with other relational MySQL stores data in tables made up of rows and columns. eMSP and CPMS can define, manipulate, control, and query data using structured Query language, more commonly known as SQL. A flexible and powerful program, SQL is the most popular open-source database in the world. Furthermore, it is easy to extend and does not require any physical organization. In RDBMS, tables are fundamental database objects logically designed to collect data in rows and columns format. While rows reflect entities, columns define the attributes of each entity. For instance, in a customer data table, each row reflects a record for a specific customer, and each table column contains corresponding customer information, like the customer's name and address. The following are key elements of the SQL database table:

- Columns: Each column holds specific attribute information, and column properties define the data type (for example, numeric or textual data) and the range it can accept. Each table has a primary key to uniquely identify an entity. A specific column, for example, Customer ID in a customer data table, can be the primary key.
- Rows: Database users can add data to each row and execute SQL queries to retrieve data. For the primary key, each row holds a unique value which also helps overcome data duplication challenges.

Here are some benefits of using SQL database:

- **Commonality**
  One of the main benefits of using SQL is the commonality of the language. It's useful in several IT systems, and you can use it with multiple other languages. The commonality of the language can benefit beginners in the profession since it's likely that they'll use SQL throughout their careers. The commonality of SQL also contributes to ease of application, which can benefit the production and efficiency of a business. A new programmer may easily apply SQL to whatever IT systems their company uses.
- **Simplicity**
  Another benefit of using SQL is the simplicity of the language. SQL commands are common English phrases, which can help programmers better understand what they're asking the language to do. Additionally, the simplicity of the language can help new

professionals in the field learn more quickly. Even professionals who have little to no coding background could learn the basics of SQL as a result of its simple syntax.

- **Integration**
  SQL is also beneficial because it can integrate easily with other programming languages. It works best with integrating with Python and R. When using the integration feature, you can more easily manipulate the data and manage the database since you're using the same coding language throughout the system. Data analysts, engineers, or web development professionals may use this feature the most.

- **Speed**
  SQL has the ability to function at a high operating speed. This high speed can increase the amount of data retrieval a professional completes. It can provide a quick and efficient way for users to retrieve, manipulate or store data.

3. User Interface Design

3.1. EV-Driver

3.1.1. Register/Verifying email

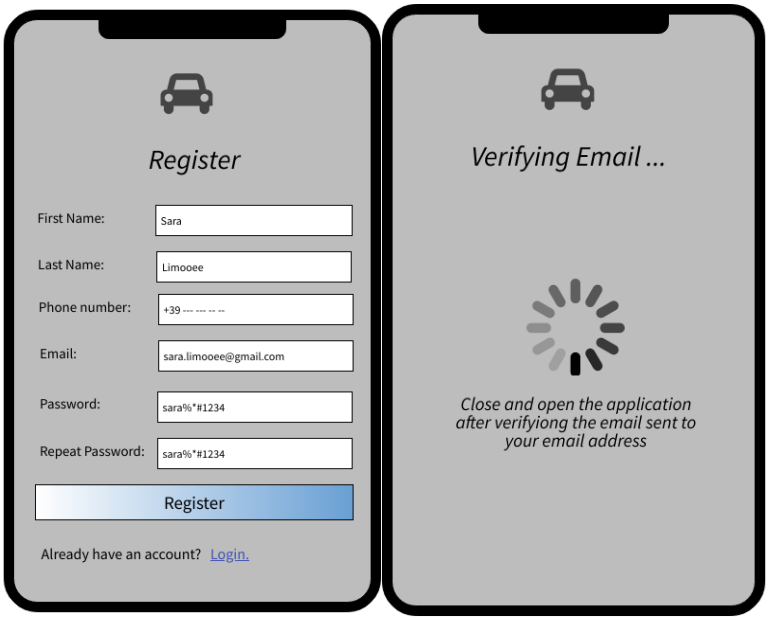

Figure-19 (UI for Login and signing in/Verifying email address)

3.1.2. Login

Figure-20 (UI for Login)

3.1.3. eMSP main page

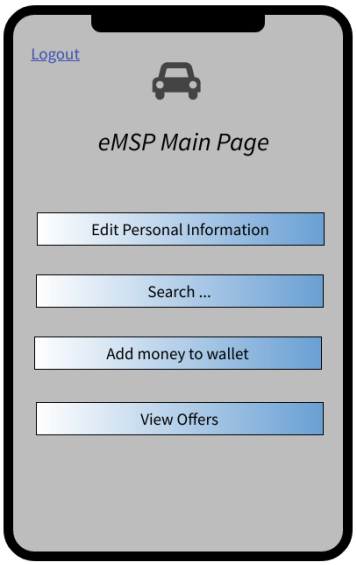

Figure-21 (UI for eMSP main page)

3.1.4. Add vehicle specification

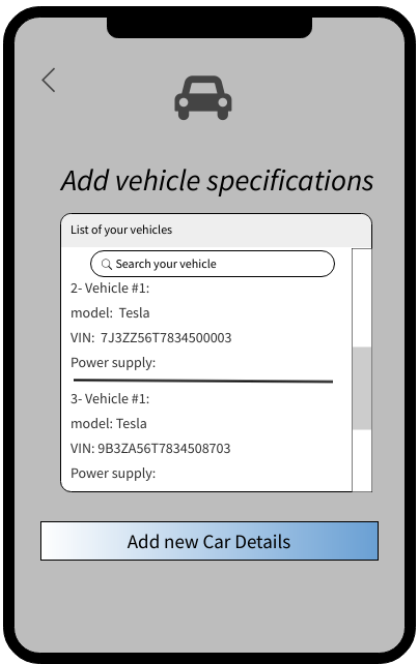

Figure-22 (UI for adding vehicle specifications)

3.1.5. Edit personal information

Figure-23 (UI for editing personal information)

3.1.6. Add money to the Wallet

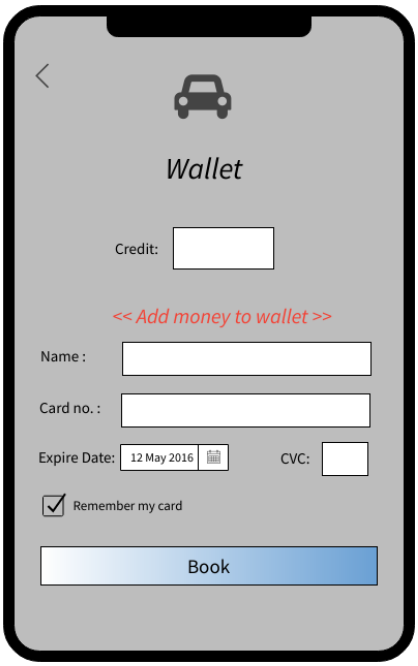

Figure-24 (UI for adding money to wallet)

3.1.7. Search for charging stations/Filter charging stations

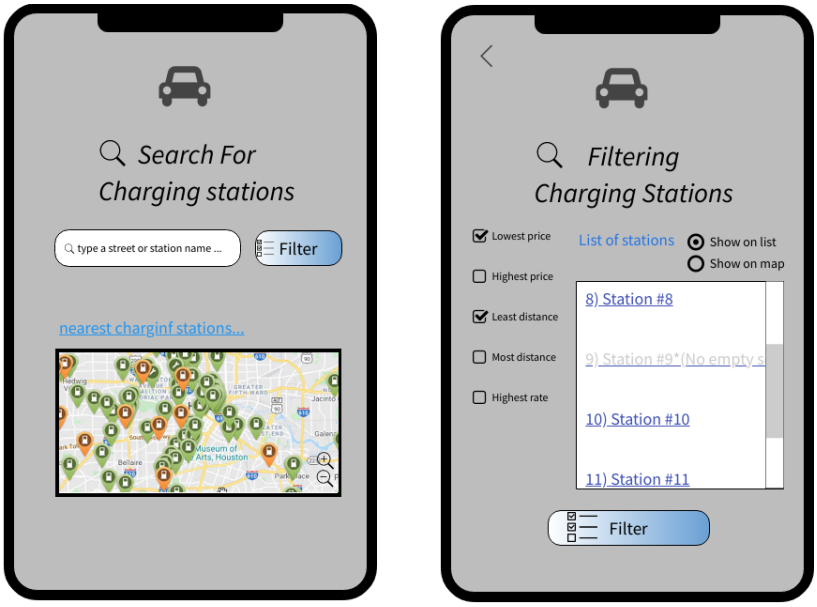

Figure-25 (UI for Searching for charging stations/Filtering charging stations)

3.1.8. Charging station details

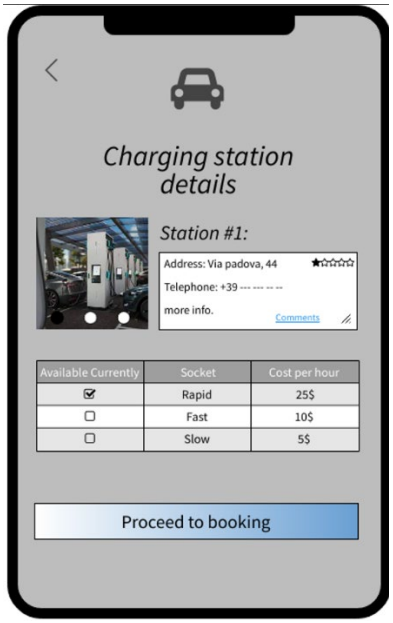

Figure-26 (UI for charging stations' details)

3.1.9. Book/Book receipt

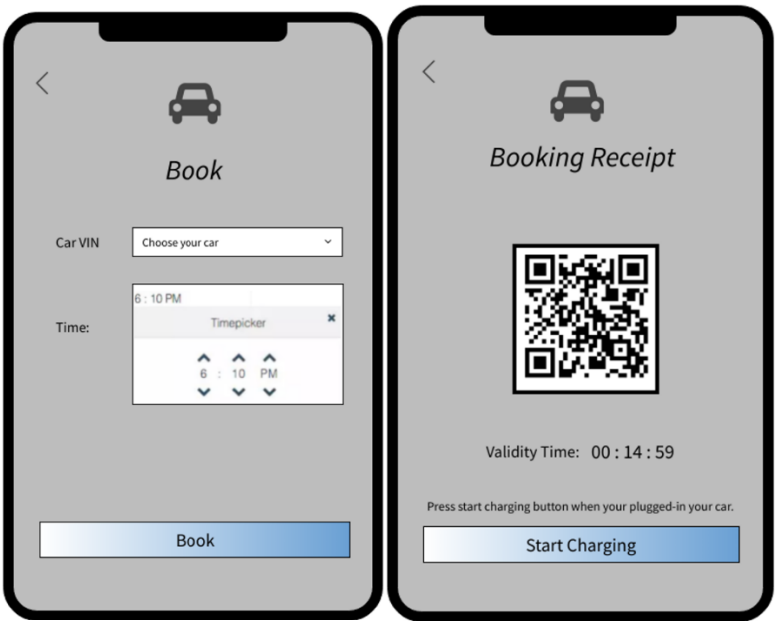

Figure-27 (UI for booking/booking receipt)

3.1.10. Charging process/Charging process finished

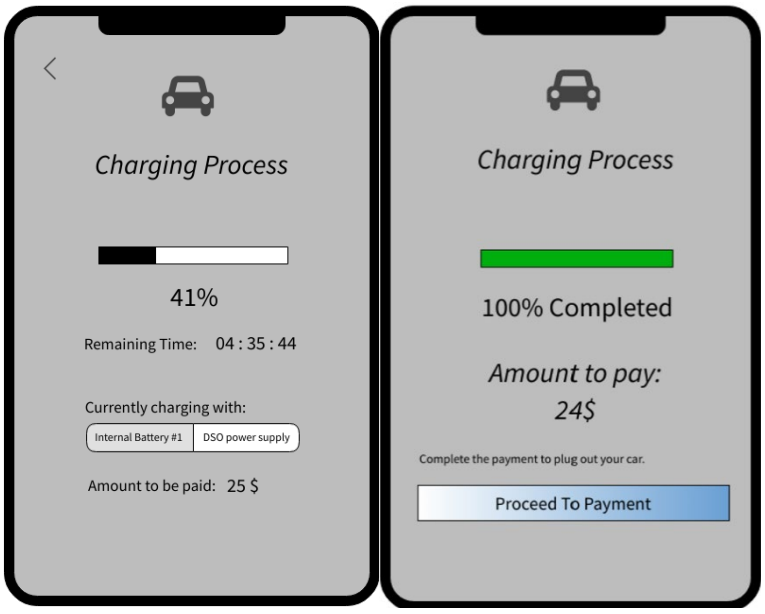

Figure-28 (UI for Charging process/Charging process finished)

3.1.11. Pay/Rate

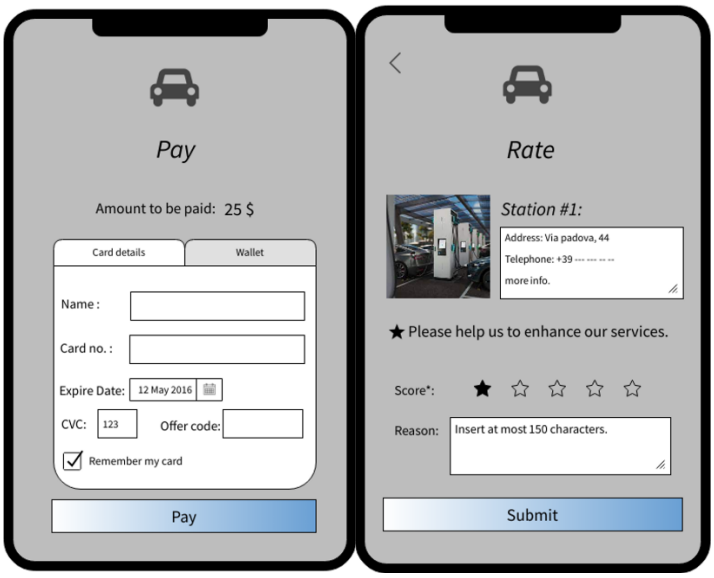

Figure-29 (UI for paying the bill/Giving a rate to received serviced)

3.2. CPO

3.2.1. Login

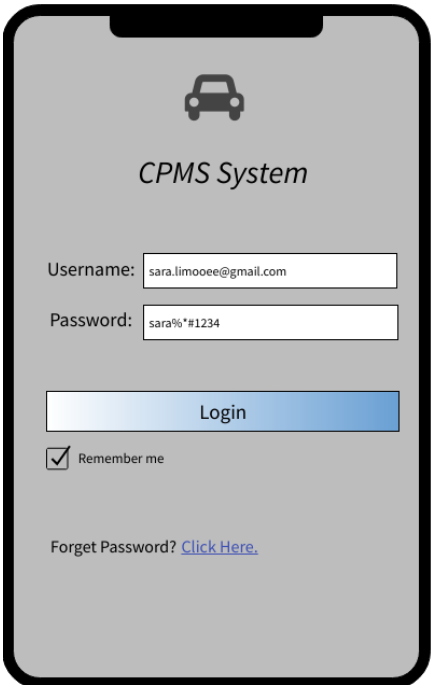

Figure-30 (UI for CPO login)

3.2.2. Charging stations status/Charging station details

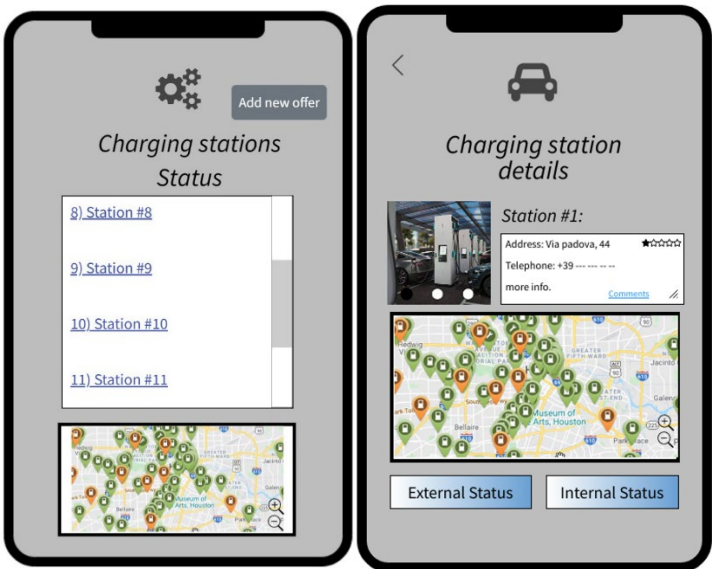

Figure-31 (UI for Charging stations status/Charging station details)

3.2.3. Internal status/charging vehicles

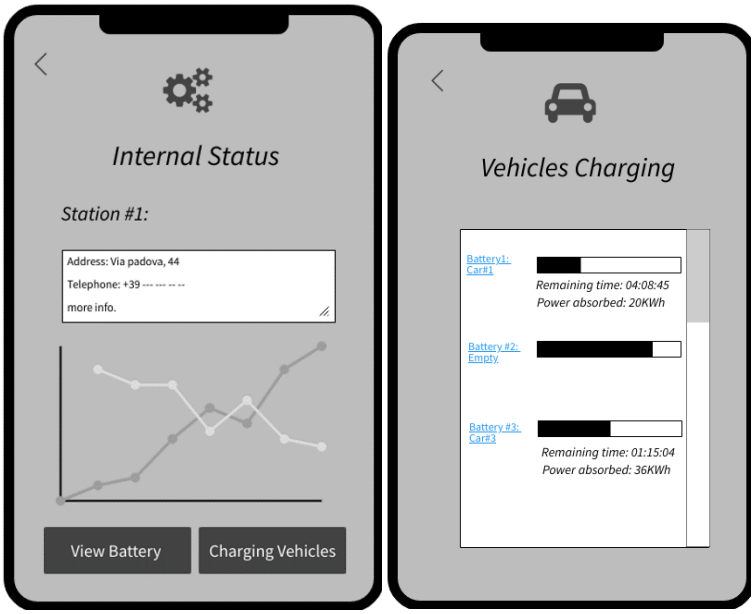

Figure-32 (UI for Internal status/charging vehicles)

3.2.4. External status/Batteries status

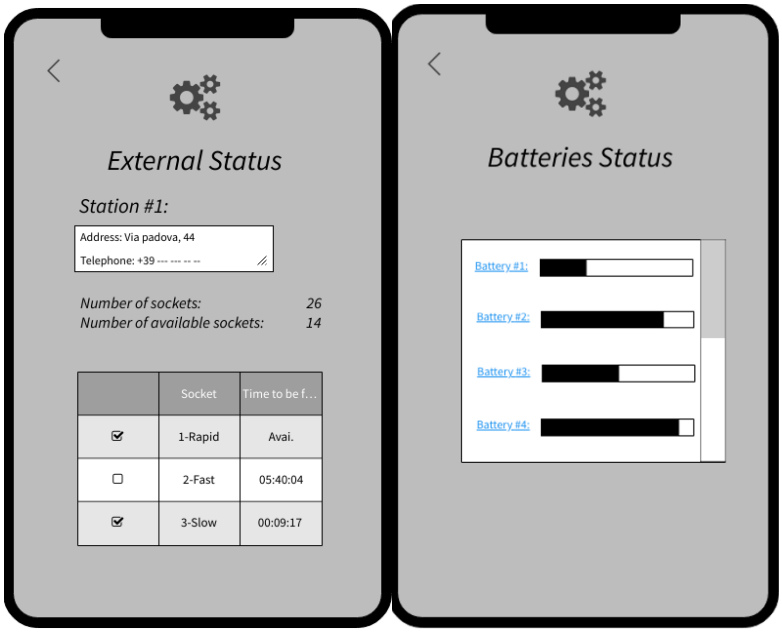

Figure-33 (UI for External status/Batteries status)

4. Requirement Traceability

In this section, a table is provided in which the components that are required in order to full fill each of the requirements specified in RASD are explained. Furthermore, in order to save space, the abbreviations of the components have been written in the list below.

- EV-driver:
    - EVA: EV-driver application
    - AU: Authentication
    - MA: Mail
    - RH: Router Handler
    - GMS: Google map service
    - SE: Search Engine
    - BO: Booking
    - PA: Pay
    - EI: Edit information
    - EAPIH: External API handler
    - MO: Model
    - DB: DBMS
    - RA: Rate
    - BA: Bank
    - DE: Data Encrypt

- CPO:
    - CA: CPO application
    - AU: Authentication
    - RH: Request Handler
    - VL: View locations
    - VIS: View internal status
    - VES: View external status
    - VDP: View DSO's prices
    - CP: Charging process
    - SDE: Select DSO to acquire energy
    - EAPIH: External API handler
    - MO: Model
    - GMS: Google map service
    - DB: DBMS
    - DS: DSO service

- EV-driver

R	UA	AU	MA	RH	GMS	SE	BO	PA	EI	EAPIH	MO	DB	RA	BA	DE
R1	*	*		*					*		*	*			*
R2	*		*	*							*	*			*
R3	*	*		*							*				
R4		*	*	*						*	*				
R5	*	*	*	*							*	*			*
R6	*	*		*				*	*	*	*	*		*	*
R7	*	*	*	*				*		*	*	*		*	*
R8	*	*		*				*	*		*	*			*
R9	*			*	*	*				*	*	*			*
R10	*			*	*	*				*	*	*			*
R11	*	*		*		*	*				*	*	*		*
R12	*			*		*				*	*	*			*
R13	*			*		*				*	*	*			*
R14	*	*		*		*					*	*	*		*
R15	*			*	*	*	*			*	*	*			*
R16	*			*			*				*				*
R17	*			*	*		*			*	*	*			*
R20	*			*							*				*
R23	*	*		*						*	*	*			*
R25	*	*		*							*	*			*
R26				*							*				*
R28	*	*		*							*	*		*	*
R29	*	*	*	*				*		*	*			*	*
R31	*	*	*	*				*		*	*			*	*
R32	*	*	*	*				*		*	*			*	*
R33	*	*	*	*							*	*	*		*
R34	*	*	*	*							*	*	*		*
R35	*	*		*							*	*	*		*
R36	*	*		*							*	*		*	*

- CPO

R	CA	AU	RH	VL	VIS	VES	VDP	CP	SDE	EAPIH	MO	GMS	DB	DS
R18			*	*						*	*		*	
R19	*		*	*	*			*			*	*		*
R21	*		*	*	*	*		*		*	*	*	*	
R22	*		*							*	*			*
R24			*	*	*			*			*	*		*
R27	*		*			*	*			*	*		*	*
R30			*							*	*		*	
R37	*		*							*	*		*	
R38	*	*	*								*		*	
R39	*	*	*								*		*	
R40	*	*	*	*		*	*		*		*	*	*	*
R41	*	*	*	*	*						*	*	*	
R42	*	*	*	*	*						*	*	*	
R43	*	*	*				*			*	*		*	*

5. Implementation, Integration and Test Plan

5.1. Overview

In this section, the plans for the implementation and testing of the application are introduced and all preliminary considerations needed to implement and test both EMSP and CPMS applications are presented.

5.2. Implementation Plan

In order to implement both systems with regard to the component diagram introduced in section 2.2., we use a bottom-up approach so that each component can be implemented and tested separately.

The components and subcomponents described in the component view section can be divided into these systems:

- Client application (EMSP)
- CPMS application

Other subcomponents are different parts of these two systems that interact with each other.

Since the implementation of some components depends on some other components, the implementation must be done from the lower components to the top ones. For example, each component and subcomponent, rely on DBMS. So, it should be implemented first. Or, the Request Handler component must be implemented before other components since it determines which component should handle each request of users of the system (EV-driver or CPO).

On the other hand, Google Maps and external API Handler are two other components that play a role like a database and are the main components. So, they should be implemented in this step.

After that, we can proceed to the implementation of the subcomponents in each Client Application component and CPMS application component. Here, there are some main subcomponents such as the Model and the DataEncrypt subcomponents. Most components use them as a connection to access the database.

Then, The Mail Service and Authentication subcomponents turn. The Mail subcomponent should be implemented before the other one because the functionalities of Authentication rely on the Mail Service subcomponent and we want to keep using the bottom-up approach while testing.

5.3. Integration & Test Plan

The sequence in which we test the system will follow the implementation order described in the previous section. The right-to-left order, as seen from the component diagram, corresponds to a bottom-up testing strategy which should work well considering the quite simple hierarchical structure of the system.

5.3.1. eMSP

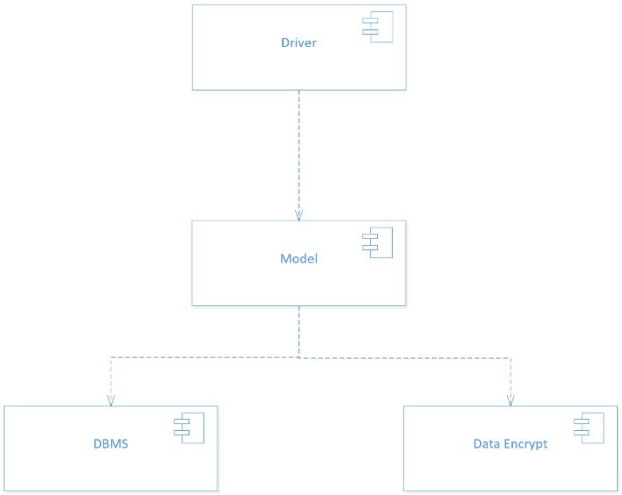

Figure-34 (Integration: Driver, Model, DBMS, Data Encrypt)

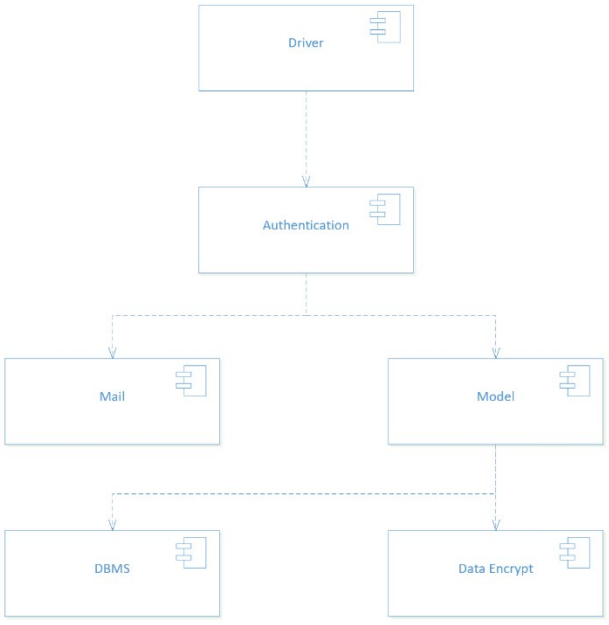

Figure-35 (Integration: Driver, Authentication, Mail, Model, DBMS, Data Encrypt)

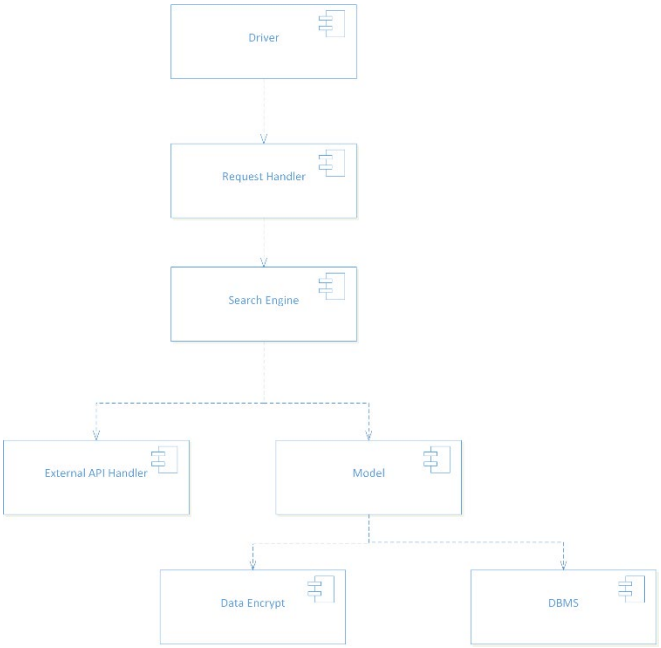

Figure-36 (Integration: Driver, Request Handler, Search Engine, External API Handler, Model, DBMS, Data Encrypt)

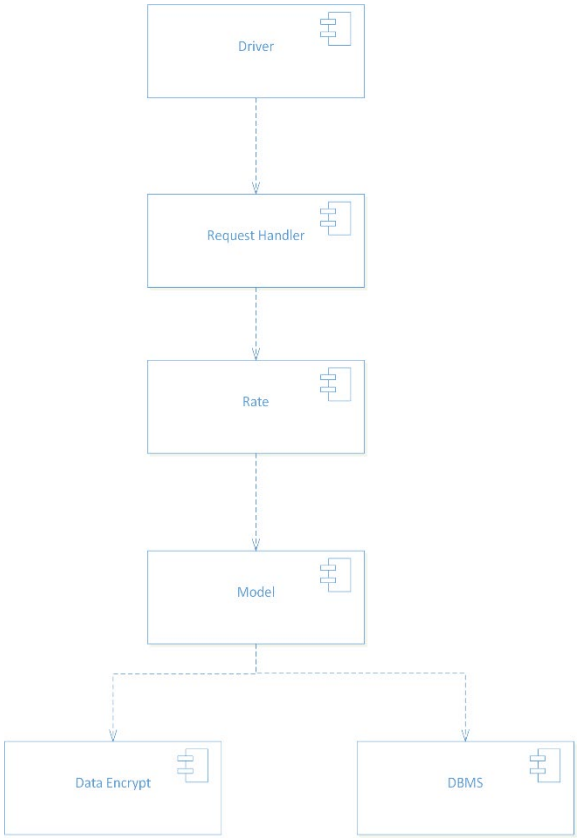

Figure-37 (Integration: Driver, Request Handler, Rate, Model, DBMS, Data Encrypt)

5.3.2.CPMS

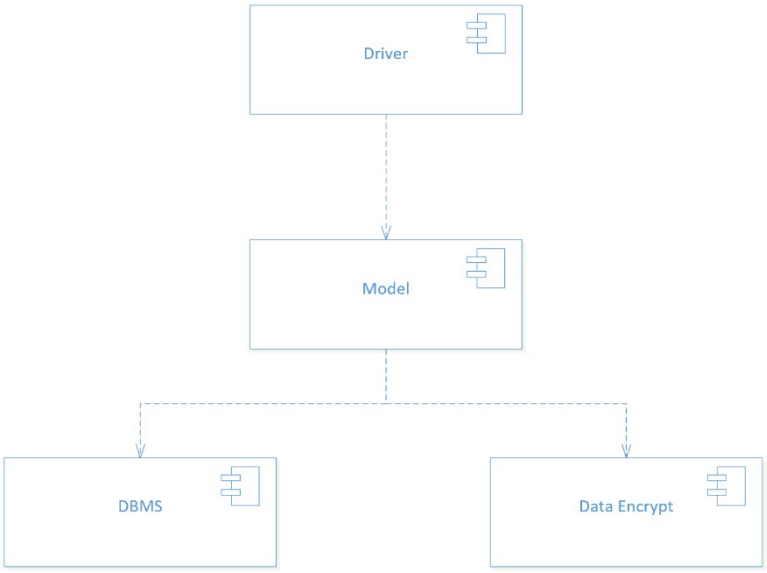

Figure-38 (Integration: Driver, Model, DBMS, Data Encrypt)

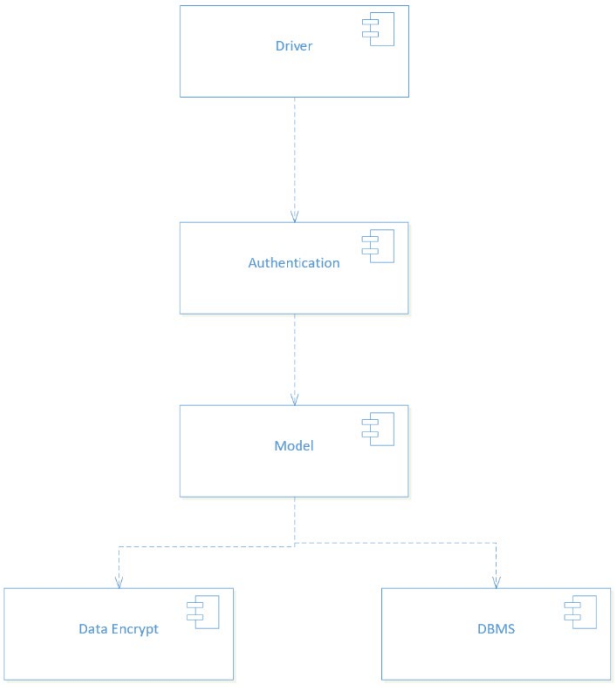

Figure-39 (Integration: Driver, Authentication, Model, DBMS, Data Encrypt)

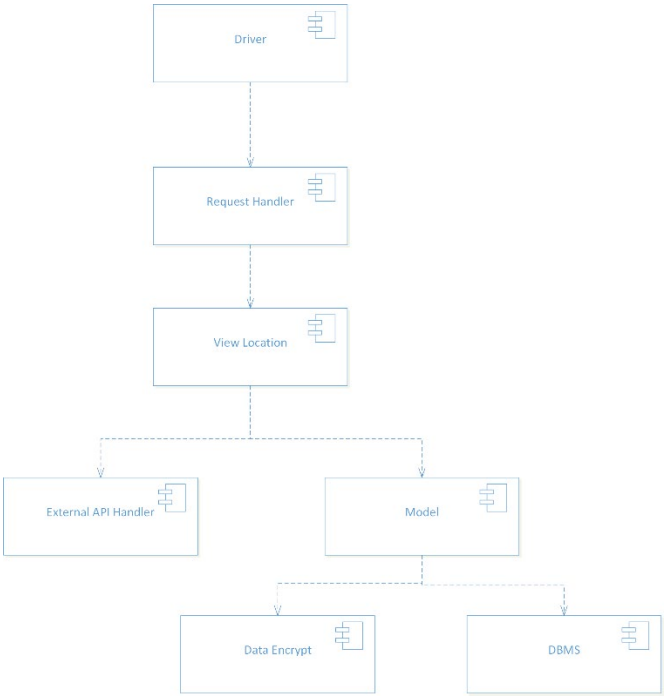

Figure-40 (Integration: Driver, Request Handler, View Location, External API Handler, Model, DBMS, Data Encrypt)

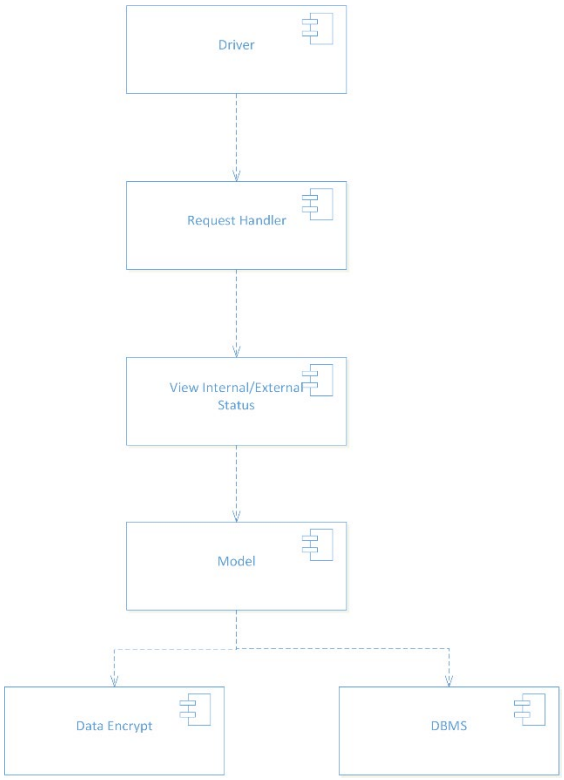

Figure-41 (Integration: Driver, Request Handler, View Internal/External Status, Model, DBMS, Data Encrypt)

5.3.3. CPMS/eMSP

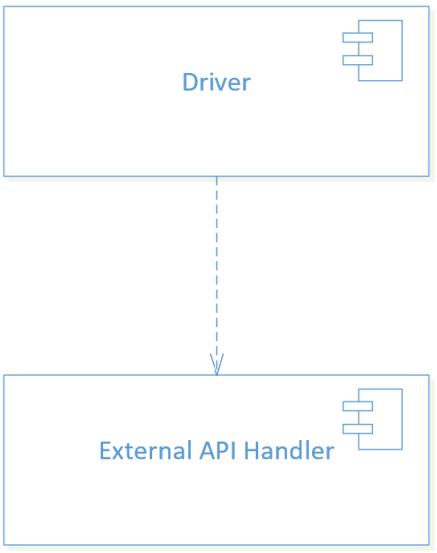

Figure-42 (Integration: Driver, External API Handler)

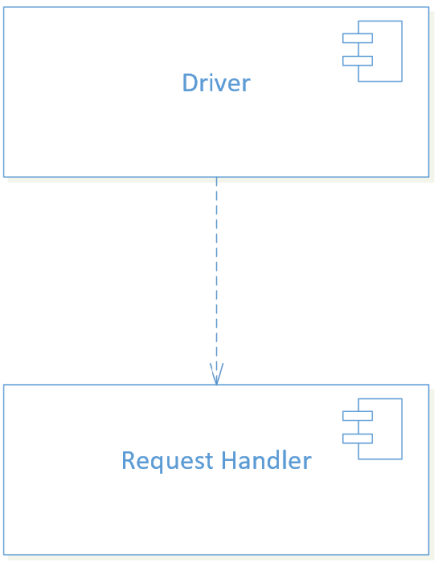

Figure-43 (Integration: Driver, Request Handler)

6. Effort Spent

- Student 1:

Topic	Hours
Introduction	2h
Architectural design	20h
User Interface Design	4h
Requirements Traceability	2.5h
Implementation, Integration and Test Plan	4h
Reasoning	5h
Total	37.5h

- Student 2:

Topic	Hours
Introduction	2h
Architectural design	20h
User Interface Design	3.5h
Requirements Traceability	3.5h
Implementation, Integration and Test Plan	3.5h
Reasoning	5h
Total	37.5h

7. References

- Specification Document: “Assignment RDD AY 2022-2023_v2.pdf”
- Course slides