AY 2022/2023

Politecnico di Milano

DD: Design Document

Marcello De Salvo Riccardo Grossoni
Francesco Dubini

Professor
Elisabetta Di Nitto

Version 0.1
December 23, 2022

Contents

1 Introduction

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 focussed to programmers, but stakeholders are also invited to read as it can be useful to understand the characteristics of the development.

1.2 Scope

The scope of this design document sets down the definition of the behavior of the system, for general and critical cases, and in the design of the system architecture by analyzing the logical designation of the components and their interactions. As stated before, this document will also delve into the implementation, the testing plan and a possible mock-up for the user interface.

1.3 Definitions, acronyms, abbreviations

Definitions

- **def1**: text.
- **def2**: text.

Acronyms

- **RASD**: Requirement Analysis and Specification Document
- **DD**: Design Document
- **ITD**: Implementation Document
- **API**: Application Programming Interface

- **DBMS**: Database Management System
- **DMZ**: Demilitarized Zone
- **OCPP**: Open Charge Point Protocol
- **UML**: Unified Modeling Language
- **GPS**: Global Positioning System
- **IT**: Information Technology
- **GUI**: Graphic User Interface
- **UI**: User Interface
- **HTTPS**:HyperText Transfer Protocol Security
- **HTML**: HyperText Markup Language
- **CSS**: Cascade Style Sheet
- **JS**: JavaScript

1.4 Revision history

- Version 0.1: setup version
  - text

1.5 Reference documents

- Specification document: "Assignment RDD AY 2022-2023"
- Requirements Analysis Specification Document (RASD)
- UML documentation: https://www.uml-diagrams.org/
- Slides of the lectures

1.6 Document structure

- **Section 1** gives a brief description of the design document, it describes the purpose and the scope of it including all the definitions, acronyms and abbreviations used.
- **Section 2** delves deeply into the system architecture by providing a detailed description of the components, the interfaces and all the technical choices made for the development of the application. It also includes detailed sequence, component and ArchiMate diagrams that describe the system in depth.
- **Section 3** contains a complete description of the user interface (UI), it includes all the client-side mockups with some graphs useful to understand the correct execution flow.
- **Section 4** maps the goals and the requirements described in the RASD to the actual functionalities presented in this DD.
- **Section 5** presents a description of the implementation, testing and integration phases of the system components that are going to be carried out during the technical development of the application.

2 Architectural Design

2.1 Overview

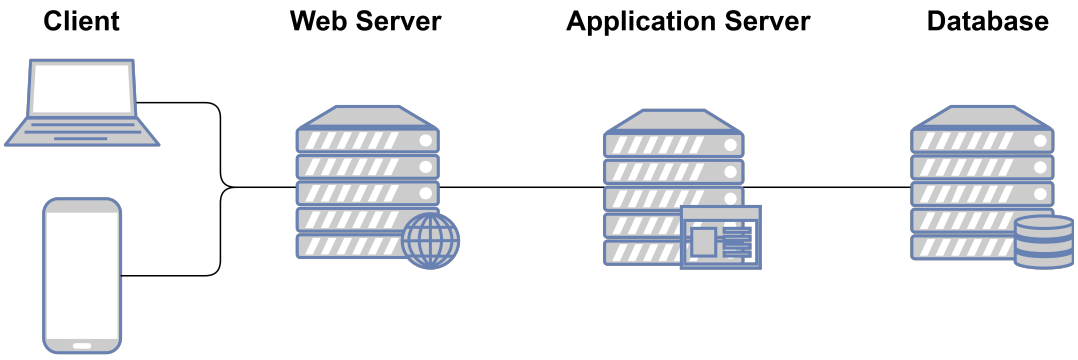


Figure 1: Three Tier Architecture

The system is an application distributed across multiple devices and follows the common client-server paradigm. The architecture is organized in three logical layers:

- **Presentation Layer (P)**: It's responsible for rendering the application's user interface to the client through the Web Server. It's expressed by a GUI designed to enable the user to interact with the application efficiently.
- **Application Layer (A)**: It's responsible for the business logic of the application. It receives requests from the Presentation Layer, processes them, and then sends back the results to be displayed.
- **Data Layer(D)**: It's in charge of the access to data sources, it provides data through the database and direct it to the other layers. It's also necessary in order to grant a high level of abstraction from the database for an easy to use model.

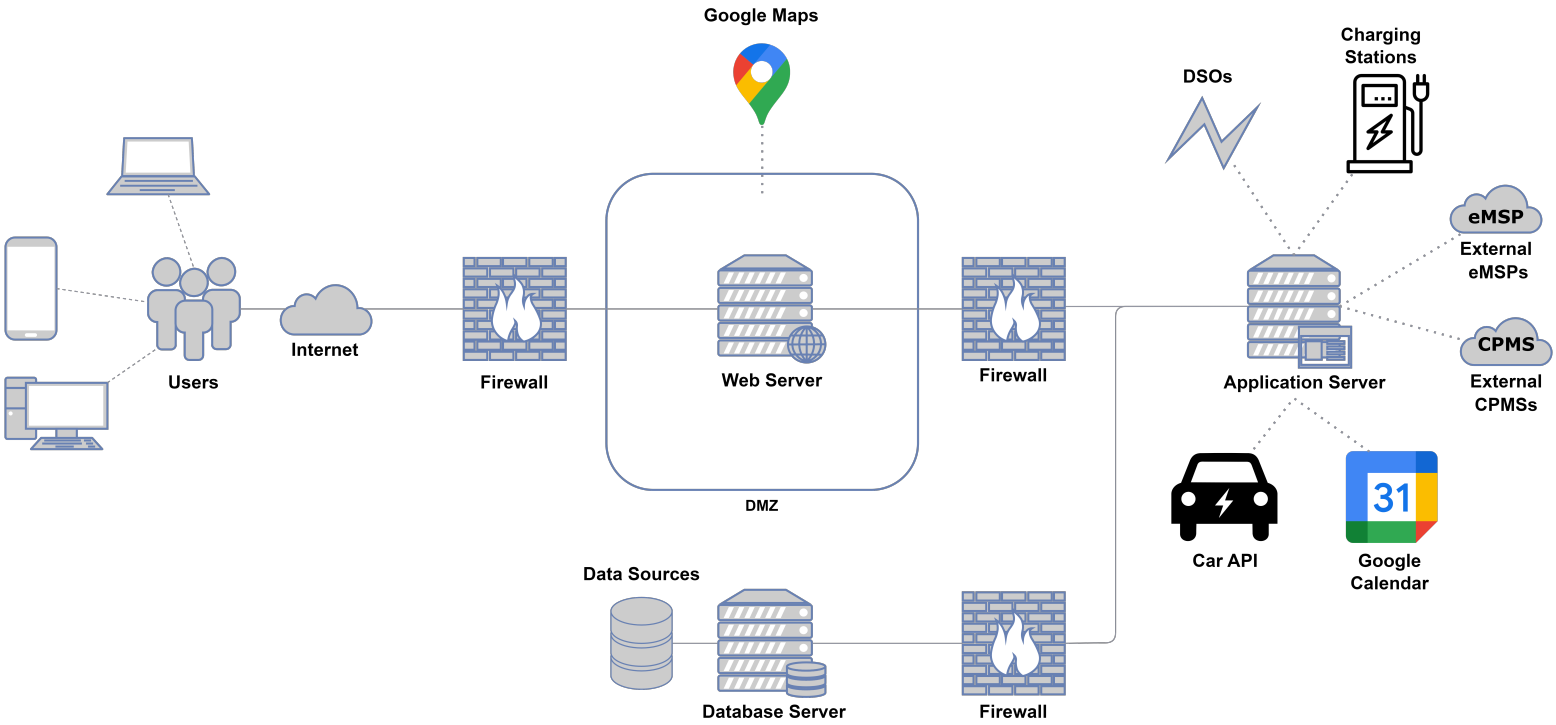


Figure 2: High Level Architecture

The system's architecture, as shown in **??**, is composed by a DMZ, that includes the Web and the Mail Server in order to prevent a direct access to the internal system and improve the overall security, and firewalls that divide each newtwork segment. To reduce computation client side a thin client is used, which allows all the heavy operations to be performed at server side. The Web Server handles the HTTP requests by the users and directs them to the Application Server, while also managing the GoogleMaps API. The Application Server communicates with the Database Server and elaborates data following its business logic. The Application server also handles external APIs and services, such as GoogleCalendarAPI, CarAPI, and the exetrnal CPMSs and eMSPs systems. The Apllication Server's CPMS subsystem will also communicate with its Charging Stations and their relative sockets through the OCPP API, and with the external DSOs through their proprietary API. The Email Server will manage all the e-mail notification

by communicating with the Application Server, The database server is responsible for storing and managing the data that is used by the application. It receives requests for data from the application logic tier, retrieves the requested data from the database, and returns the data to the application logic tier.

2.2 Component view

General Component View

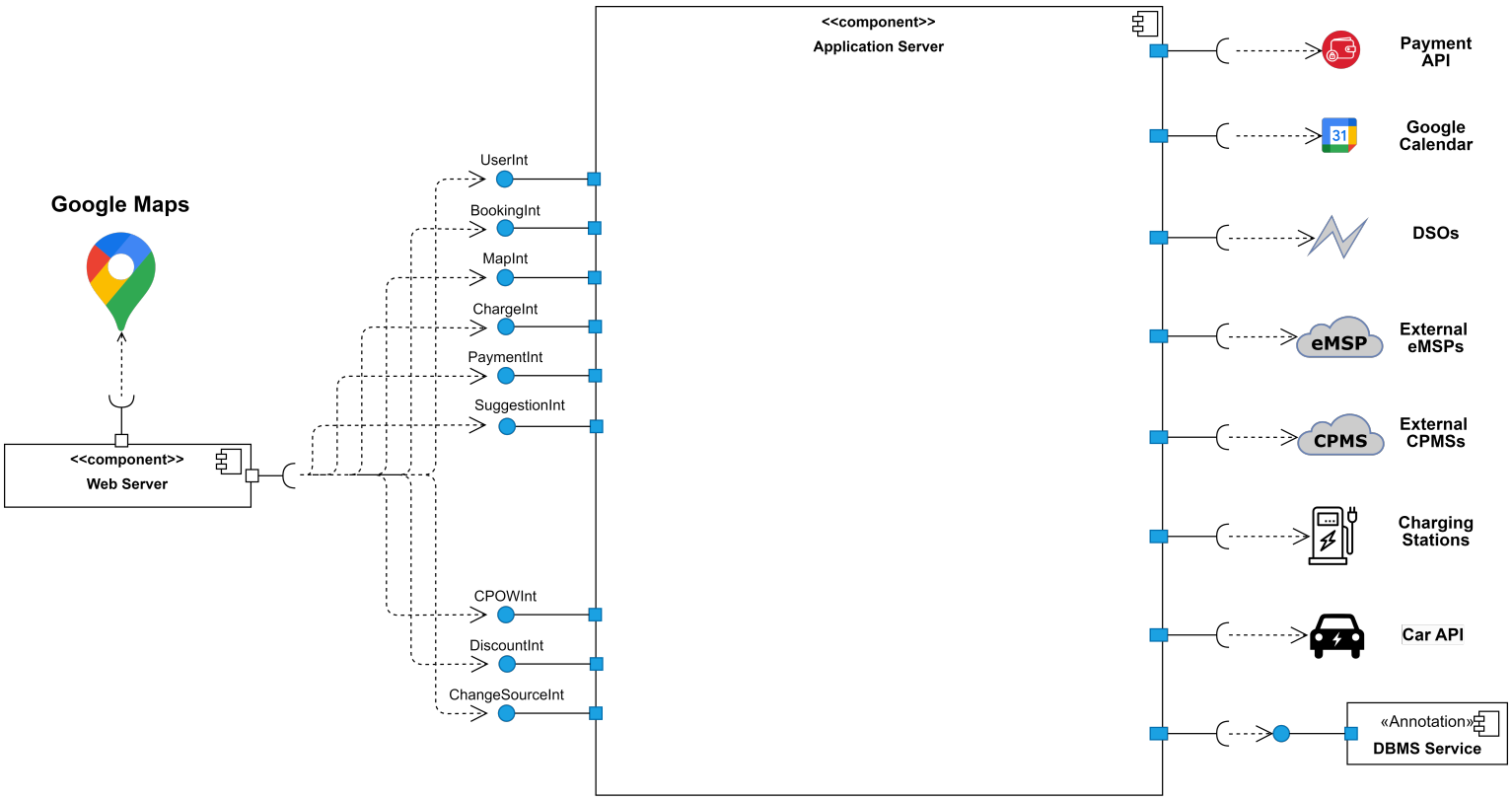


Figure 3: General Component Diagram

This image gives a high level representation of the components of the system. *description*

Application Server Component View

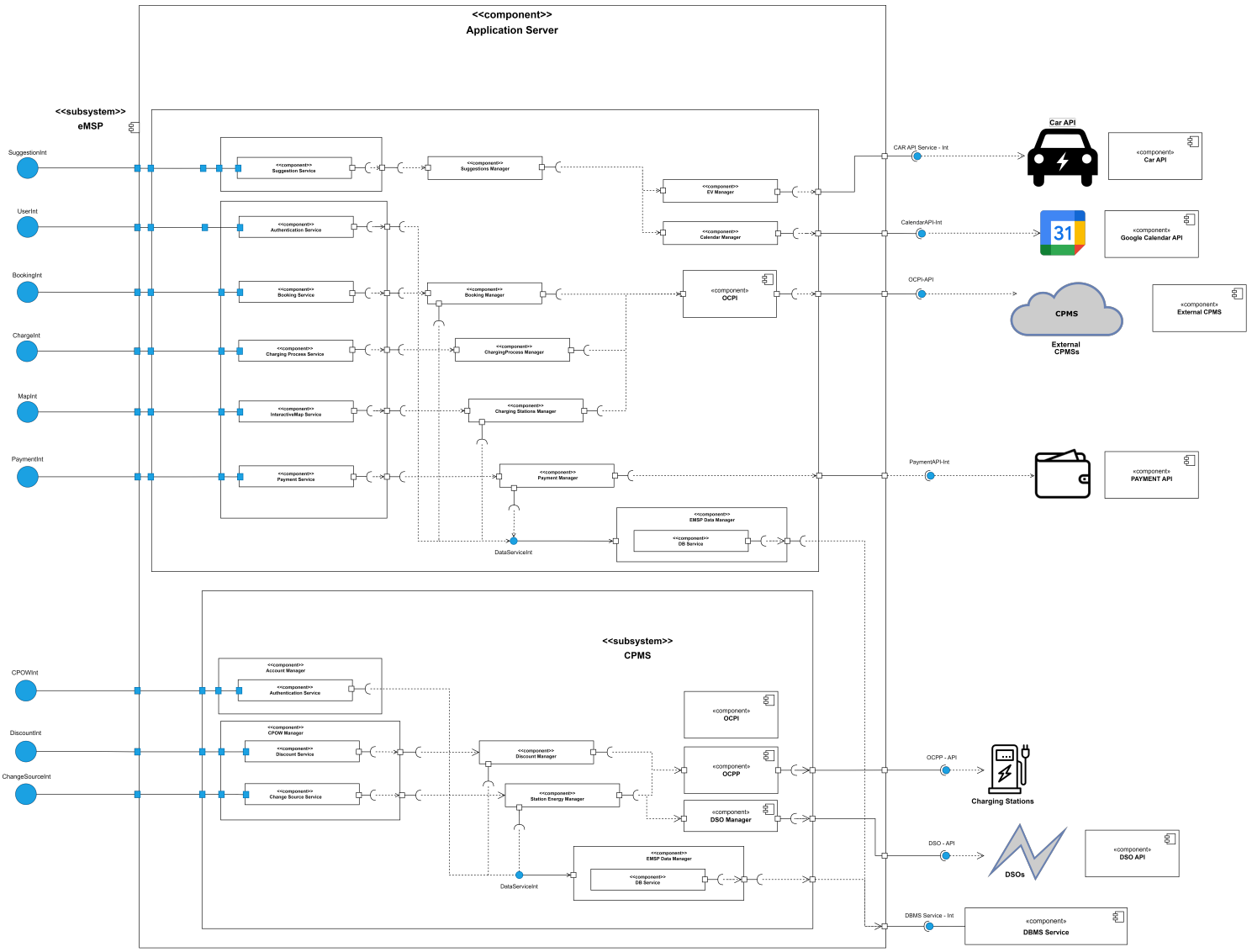


Figure 4: Application Server Component Diagram

The following component diagram gives a detailed view of the Application Server. It shows the internal structure and the interaction between the

components. External elements in the diagram are represented in a simplified way.

- **first component**: description of compontent.

Web Server Component View

Regarding the Web Server the main components are:

- **first component**: description of component.

2.3 Deployment view

Figure 5: Deployment Diagram

The deployment diagram in *Figure (5)* shows the most important components necessary for the correct behaviour of the system. The devices shown in the diagram are:

- **first device/component**: description of device (e.g. pc or Smartphone, firewall, load balancer, web server, app server, database server...).

2.4 Runtime view

Here the runtime views of some relevant use cases of the system are represented through sequence diagrams. In the diagrams, the part regarding the user is omitted because it has been deemed as redundant to the understanding of the interaction. In later diagrams, some parts, like the login phase or returning to home page, were omitted for the the same motivations.

Sign Up

Figure 6: Sign Up

description of sign up as seen of the app by the user, with actions and final result

Log In

Figure 7: Log In

The Log In phase simply consists in the user action of inserting their email (unique key in the database) and password in the login fields, then the system checks if the pair corresponds to a user entry in the database. In case of success, the user can log in the system and use its functionalities available for that particular type of account chosen during the Sign Up phase.

in app action 3

Figure 8: *in app action 3*

description of the "app" action 3.

2.5 Component interfaces

Figure 9: Component Interfaces Diagram

This diagram above describes in detail the interfaces and the corresponding methods offered by each component, it also shows the interaction between them. *other text*

2.6 Selected architectural styles and patterns

- **type of architecture**
  description and reasoning
- **Type of client**
  description and reasoning
- **Scalability**
  how can the architecture manage help scalability
- **MVC?**
  general description

- Model: the central component of the pattern. It is the application's dynamic data structure, independent of the user interface. It directly manages the data, logic and rules of the application.
- View: The view defines how the app's data should be displayed.
- Controller: it contains logic that updates the model and/or view in response to input from the users of the app.

2.7 Other design decisions

2.7.1 Servers availability and response time

text

2.7.2 exaple of design decision

description of example

$$possible\ equation\ to\ be\ included \tag{1}$$

development of said equation

$$description\ of\ development\ pt1\ description\ of\ development\ pt2 \tag{2}$$

other development and various considerations
conclusion of design decision

2.7.3 design decision 2

- text

2.7.4 design decision 3

text

3 User Interface Design

Figure 10: Sign Up and Log In

Figure 11: carOwner interactions

Figure 12: CPOW interactions

4 Requirements Traceability

Requirements	**Components**
R1) text. R2) text. R3) text.	• text. • text. • text.
R4) text.	• text.

R5) text.
R6) text.
R7) text.
	• text.
	  – text.
	  – text.
	  – text.
	• text.
	• text.
	• text.
	• text.

R8) text.	• text.

R9) text.	• text. – text. • text.
R10) text.	• text. – text. • text. • text. • text.
R11) text.	• text. – text. • text. • text.
R12) text.	• text. – text. • text.

R13) text. R14) text. R15) text. R16) text. R17) text.	• text. – text.
R18) text.	• text. • text. – text.
R19) text.	• text. • text. • text.
R20) text.	• text. • text.
R21) text.	• text.

Here, we present a summary of the table above for a more immediate visualization. *component: name associated in the next table*

component 1:	c1
component 2:	c2

Table 2: Components' legend

	c1	c2
R1	x	x
R2	x	x
R3	x	x
R4	x	
R5		
R6		
R7		
R8		
R9		
R10		
R11		
R12		
R13		
R14		
R15		
R16		
R17		
R18		
R19		
R20		
R21		

Table 3: Component and requirement mapping

5 Implementation, Integration and Test Plan

5.1 Implementation Plan

Multiple components will be implemented at the same time, in order to parallelize the development when possible. The general plan is to follow a bottom-up approach, so that core and basic functionalities with very few dependencies can be tested as soon as their incapsulating component is done. By doing so, the application will be built up with solid and tested foundations that will ease the further testing of bigger and complex components. In any case, unit testing will be performed on each component on the go, in order to

find flaws out in advance. This will positevely impact the necessary actions to fix the faults since they will be done in an earlier stage.

The implementation's order of the component will be as follow:

1. list, of, components
2. list, of, components, 2
3. list, of, components, 3

Each group is composed by independent modules so they can be easily developed in parallel. Furthermore, it is expected that external services (e.g. GoogleMapsAPI, *others* and DBMS Service) work properly since they're not a responsibility of the *name* app.

explanation of the implementation plan component order

5.2 Integration Strategy

Considering both the overall system's architecture and the implementation plan, the chosen integration strategy is the bottom-up approach. System integration begins with the integration of the lowest level modules and uses test drivers to drive and pass appropriate data to the lower-level modules. As and when the code for the other module gets ready, these drivers are replaced with the actual module.

This approach allows to start the integration and testing without necessarily waiting for the completion of the development and the unit testing of each system's component. Being the low-level modules and their combined functions often invoked by other modules, it is more useful to test them first so that meaningful effective integration of other modules can be done. Moreover, starting at the bottom of the hierarchy means that the critical modules are built and tested first and therefore any errors in these modules are identified early in the process.

Each integration in the same level (defined by the groups of the previous section) is independent and there is no specific order in which to complete them. In this way, the integration process and its testing are more flexible.

5.2.1 Integration and Testing

In this section it is defined the order of the integration between components. Test drivers will be used to simulate higher components not yet implemented.

order of the components shown in the next figures, with references

Figure 13: Integration of first component

Figure 14: Integration of second component

(a) component 3a

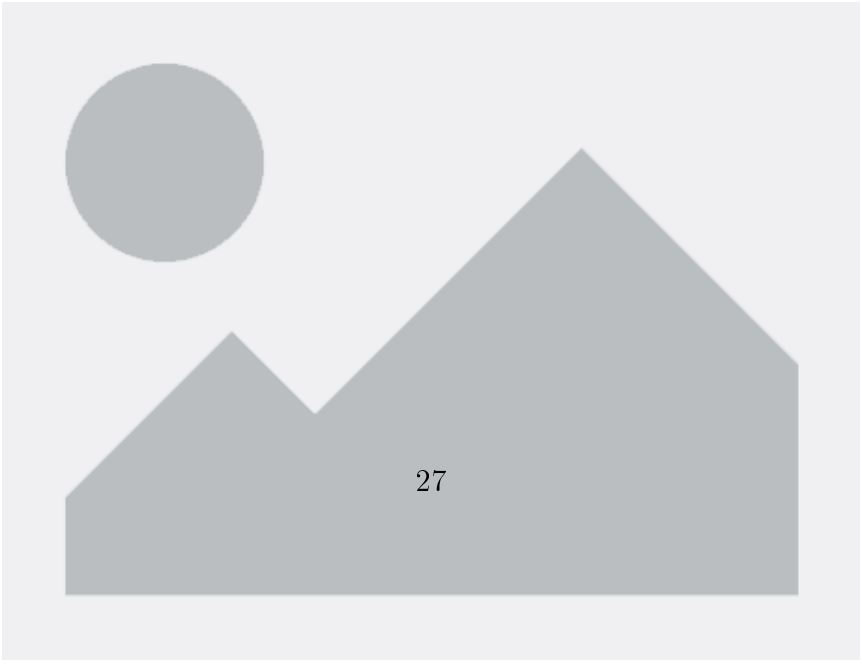
(b) component 3b

Figure 15: Integration of 3a and 3b components

order of components for the middle tier, with references

(a) component middle1-a

(b) component middle1-b



(c) component middle1-c

Figure 17: Integration of middle2 component

order of last group components, with references to the images

Figure 18: Integration of carOwnerManager components

Figure 19: Integration CPOWmanager components

Figure 20: Integration of otherManager components

Figure 21: Integration of AccountManager components

5.3 System testing

text for sys testing

6 Effort Spent

Student	Time for S.1	S.2	S.3	S.4	S.5
stud1	0h	0h	0h	0h	0h
stud2	0h	0h	0h	0h	0h
stud3	0h	0h	0h	0h	0h

7 References

References

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

[2] description: `urlhere`