EPHEC

Development Of Car Rental Management System

Rapport de stage

Daniel Olivier

31 may 2021

Table of contents

List of Tables

List of Figures

Listings

Acknowledgements

First and the foremost, I am grateful to my supervisor, Louis VAN DORMAEL, for insightful conversations during the development of this project, and for helpful comments on what should be corrected.

Finally, I would like to thank the various EPHEC professors who helped me to obtain the skills required to accomplish this task.

Daniel Olivier

Louvain-La-Neuve, 10 May 2021.

Abstract

This Car Rental System project is designed to aid the stakeholder manage his cars that are up for rent through an online reservation system. The latter has many features such as users can search for available cars; view profile; book the car for a given time period and so much more...

By using this application, the administrator can manage all registered customers, reservations and available car details. It has a user-friendly interface which helps a customer search for rented cars and book any of them for a specified period. Upon a successful reservation a customer receives a confirmation email detailing everything the latter needs to know about their reservation.

The main focus of this project is to help any customers easily find a car that's available in the system as well as help the stakeholder manage rented cars and customers in the system efficiently.

To propose the best possible available car to customers a Technique for Order Preference by Similarity to an Ideal Solution (TOPSIS), was used. More on this later.

1 Chapter I : Introduction

1.1 Overview

> "*If your business is not on the internet, then your business will be out of business*"
>
> — Bill Gates, Founder of Microsoft.

The car rental management system presented in this report is a web-based application for the stakeholder's company that's accessible to the public through the web and furthermore it was developed to keep and maintain records about reservations.

The main feature of this application is to keep tracks of vehicles, customers and booking. The latter provides useful information to the administrator such as making reports of booked vehicles and acts as a vehicle management system by monitoring the use and price of these vehicles. From the customers point of view, this application helps them find that car to move from one place to another for business or school purposes, tour, and sightseeing occasions. Thus, making it extremely useful.

1.2 Problem Statement

Personally speaking, I have found myself in a situation trying to get a car over the phone but all lines seemed busy, or maybe I was in a noisy place, where it's difficult to speak, or vice versa, somewhere I had to be quiet. A tricky task at times. Modern life is busy and dynamic. No wonder people lay their hands on anything that can make things quicker and easier. Car rental software development significantly simplifies the entire process of booking a car.

The stakeholder has some personal vehicles at disposal for people looking to rent a car for a given period. He wants to go digital and to accept online reservations and manage his fleet with ease. In today's digital environment, customers would rather book cars online instead of calling the rental company to make reservations. If you are running a car rental business, developing a car rental software becomes a must.

I got in touch with him through a mutual acquaintance to implement a well-designed car rental system to help not only manage and maintain his entire fleet online but as well as his potential customers book available cars on the to-be developed website. This car rental application should allow him to run his business smoothly and effectively.

1.3 Stakeholder(s)

Several types of stakeholders can be identified when it comes to any software. The most obvious are those that requested a system to be developed. In my case it's my Dutch born client that basically hired me (*figuratively speaking*) to develop this car rental management web based application for his upcoming business.

1.4 Aim and Objectives

Below are the objectives of this project :

- Develop a user-friendly & secure system that protects client information as well as confidential information of the company
- A customer self-service platform to view vehicle availability in real-time,
- 24/7 online system to allow customers to book cars,
- Remove the paper-based processes,
- Detailed analytics and statistics, i.e., the software should deliver an up-to-date analytics on customers' reservations,
- Avoiding risks of overbooking and reduce the factor of human error,
- The reservations' management system to track vehicles due for maintenance, booked ones, or currently on the road,
- Provide customers the ability to apply different filters while searching for available cars,
- Provide customers the ability to cancel their reservation,
- Make data-driven decisions based on detailed statistics,
- Feedback system for clients to give reviews and rate the service,
- Provide an estimation of the influx of bookings to prepare for future demands.

1.5 Scope of Proposal

The scope of an application is there to define its boundaries. In other words, what is in scope and what is out of scope.

- The platform should 24/7 available to customers
- The system does not support online payment at the moment
- This system is for use by only one company
- Rented cars should be used at the moment in one country Belgium
- No mobile application will be developed for this car rental web application at the moment.

2 Chapter II : Literature Review

2.1 Overview

This chapter contains a literature review for the application that was developed. The review describes the existing systems that are similar to the car rental management system.

Furthermore, literature review helps to provide an overview on how the **TOPSIS**[1] technique was implemented as a method in multiple criteria decision-making to prioritize the best car possible to customers.

2.2 Multi-Criteria Decision Making Method

As the name implies, Multi-Criteria Decision-Making also known as **MCDM** is about methods for making decisions when multiple criteria need to be considered together, in order to rank or choose/prioritize between the alternatives being evaluated.

2.2.1 Technique for Order of Preference by Similarity to Ideal Solution - TOPSIS

> The **TOPSIS** is a multi-criteria decision analysis method developed by Hwang and Yoon (1981) with further developments by Yoon (1987) and Hwang, Lai and Liu (1993) (Surendra, 2016). It's based on the concept that the chosen alternative should have the shortest geometric distance from the positive ideal solution, and the longest geometric distance from the negative ideal solution.
>
> — Wikipedia

After some research about how to provide the best ideal car possible to client, TOPSIS is the best algorithm for this application because it will make the latter more efficient.

The TOPSIS algorithm is to contrive the best ideal solution (note as $s+$), and the worst ideal solution (note as $s-$) to the problem of multiple criteria while the $s+$ is known as the optimal solution from the criteria, but the $s-$ is the worst solution from the criteria. The rule is to rank and compare each alternative of the result with $s+$ and $s-$.

The TOPSIS algorithm was carried out as follows:

[1] TOPSIS

Step 1 : Construct the decision matrix and determine the weight of criteria.

Let's say we have three cars available for rent with four criteria as shown in Table 1

Table 1: Table of Criteria.

Attribute/Criteria	Price	Numbers of Passengers	Number of Bags	Class
Volkswagen Polo	110€	3	2	Basic
Toyota Yards	230€	5	4	Standard
Opel Zafira	300€	9	7	Premium

Table 1 shows that each car have their criteria. If class linguistic terms are converted using the 5 point scale, the following table appears:

Table 2: Table of Criteria with 5 point scale.

Attribute/Criteria	Price	Numbers of Passengers	Number of Bags	Class
Volkswagen Polo	110€	3	2	2
Toyota Yards	230€	5	4	3
Opel Zafira	300€	9	7	5

Step 2 : Calculate the normalized decision matrix.

This is the formula for vector normalization :

$$n_{ij} = \frac{x_{ij}}{\sqrt{\sum_{i=1}^{m} x_{ij}^2}} \tag{1}$$

Let's start by calculating the denominator for the price column :

$$\begin{aligned} \sqrt{\sum_{i=1}^{m} x_{ij}^2} &= 110^2 + 230^2 + 300^2 = 155000 \\ &= \sqrt{155000} = 393.70 \end{aligned} \tag{2}$$

Table 3: Table of Criteria with the performance value

Attribute/Criteria	Price	Numbers of Passengers	Number of Bags	Class
Volkswagen Polo	110€	3	2	2
Toyota Yards	230€	5	4	3
Opel Zafira	300€	9	7	5
Performance Value	393.70	10.72	8.30	6.16

Next, the data for each criterion was normalised. Each criterion was divided with their own criteria performance values as shown in Table 3 :

Table 4: Table of Criteria with the normalised values

Attribute/Criteria	Price	Numbers of Passengers	Number of Bags	Class
Volkswagen Polo	$\frac{110}{393.70}$ = 0.27	0.27	0.24	$\frac{2}{6.16}$ = 0.32
Toyota Yards	0.58	0.46	0.48	0.48
Opel Zafira	0.76	0.83	0.84	0.81

The value in each sell is known as the normalized performance value.

Step 3 : Calculate the weighted normalized decision matrix

The weighted normalized value v_{ij} v is calculated in the following way:

$v_{ij} = w_j \, n_{ij}$ for i = 1, …, m; j = 1, …, n where w_j is the weight of j-th criterion, $\sum_{j=1}^{n} w_j = 1$

This means that each criterion should have its own weight so that all of them will sum up to 1.

Let weight price be = 0.4, number of passengers = 0.2, number of bags = 0.1 and class = 0.3. The normalised value will be multiplied by corresponding normalised weight. as shown in Table 5 :

Table 5: Table of Criteria with the weighted normalised values

Attribute/Criteria	Price	Numbers of Passengers	Number of Bags	Class
Volkswagen Polo	0.27 * 0.4 = 1.10	0.05	0.02	0.32 * 0.3 = 0.09
Toyota Yards	0.23	0.09	0.04	1.44
Opel Zafira	0.30	0.16	0.08	0.24

Step 4 : Determine the worst alternative and the best alternative

Table 6: Table of Criteria with positive ideal and negative ideal solutions.

Attribute/Criteria	Price	Numbers of Passengers	Number of Bags	Class
Volkswagen Polo	1.10	0.05	0.02	0.09
Toyota Yards	0.23	0.09	0.04	1.44
Opel Zafira	0.30	0.16	0.08	0.24
V^+	0.23	0.16	0.08	0.09
V^-	1.10	0.05	0.02	1.44

From table 6, V^+ represents the best ideal solution while V^- is the worst ideal solution. V^+ is taken from the highest value while V^- taken from the lowest value. For the price column a lower value is desired hence V^+ indicates the lowest value same goes for the class column as these two are linked.

Step 5 : Find the Euclidean distance between the best ideal solution(V+), and the worst(V-).

- *The Euclidean formula from the ideal best value :*

$$s_i^+ = \sqrt{\sum_{i=1}^{m} (v_{ij} - v_j^+)^2} \tag{3}$$

$$\sqrt{\sum_{i=1}^{m}(v_{ij}-v_j^+)^2} = (1.10-0.23)^2+(0.05-0.16)^2+(0.02-0.08)^2+(0.09-0.09)^2 = 0.765$$
$$= \sqrt{0.7656} = 0.87 \tag{4}$$

- *The Euclidean formula from the ideal worst value :*

$$s_i^- = \sqrt{\sum_{i=1}^{m}(v_{ij}-v_j^-)^2} \tag{5}$$

$$\sqrt{\sum_{i=1}^{m}(v_{ij}-v_j^+)^2} = (1.10-1.10)^2+(0.05-0.05)^2+(0.02-0.02)^2+(0.09-1.44)^2 = 1.82$$
$$= \sqrt{1.82} = 1.35 \tag{6}$$

Table 7: Table of Criteria with the euclidean distance

Attribute/Criteria	Price	Numbers of Passengers	Number of Bags	Class	S^+	S^-
Volkswagen Polo	1.10	0.05	0.02	0.09	0.87	1.35
Toyota Yards	0.23	0.09	0.04	1.44	1.35	0.86
Opel Zafira	0.30	0.16	0.08	0.24	0.30	1.45
V^+	0.23	0.16	0.08	0.09		
V^-	1.10	0.05	0.02	1.44		

Step 6 : Calculate the relative closeness to the positive ideal solution

The formula to calculate the performance score is as follows :

$$p_i = \frac{s_i^-}{(s_i^- + s_i^-)} \tag{7}$$

Table 8: Table of Criteria with the performance score

Attribute/Criteria	Price	Numbers of Passengers	Number of Bags	Class	p_i
Volkswagen Polo	1.10	0.05	0.02	0.09	0.6
Toyota Yards	0.23	0.09	0.04	1.44	0.3
Opel Zafira	0.30	0.16	0.08	0.24	0.8

Step 7 : Rank the performance score in descending order or select the alternative closest to 1

Table 9: Table of Criteria with ranked performance score

Attribute/Criteria	Price	Numbers of Passengers	Number of Bags	Class	Rank
Volkswagen Polo	1.10	0.05	0.02	0.09	2
Toyota Yards	0.23	0.09	0.04	1.44	3
Opel Zafira	0.30	0.16	0.08	0.24	1

After all these steps its safe to say that Opel Zafira is the most ideal solution for given criteria. The application will display cars based on their ranks meaning Opel Zafira will be at the top, followed by Volkswagen Polo and lastly Toyota Yards. This is how I was able to provide to the client the best car possible given some criteria.

2.3 Related Work

A. Avis

Avis[2] is Belgium a company based in Brussels.

This platform is used to ensure the customers have access car hire services. There are several characteristics for this application. User can find car and pick-up point based on a customer selected location. Customer can easily choose a car, book it after a successful online payement. The customer can select which location to drop off the car upon return date.

[2] Avis

B. Europcar

Europcar[3] a global leader in car rental business.

This application is dedicated to making car hire online as easy as possible and providing online services worldwide.

This application provides many services such as negotiated rate that are numeric-only discount codes for companies that have a partnership with Europcar.

[3] Europcar

3 Chapter III : Software Requirement Specifications - SRS

3.1 Overview

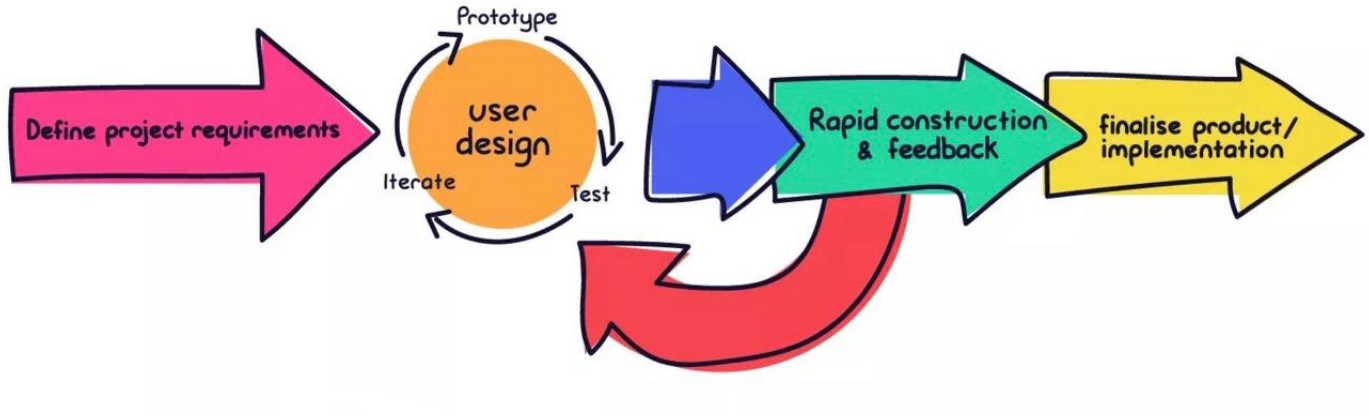
This chapter contains the analysis and design of this application, the chosen type of the methodology, function & non-functional requirements and software tools that were used during the development phase. In addition, this chapter contains UML diagrams, the database design, the application architecture to better understand how the latter is designed.

3.2 Proposed Methodology

Methodology in software development is the process of dividing the latter into distinct phases to improve design, product quality etc… Each of methodology is chosen based on the nature of the application to be implemented and on what the stakeholders of the system required. For this particular application **Prototyping model** was chosen.

3.2.1 Prototyping Model

Figure 1: Prototyping Model



— From codebots.com

Prototyping model was chosen firstly because it allowed me to implement many prototypes which were presented to the stakeholder in order to get some feed-back based on how the final application should look and work from his point of view. Secondly because I knew that stakeholder was going to be heavily involved in the creation of this application which by the way helped him not only get as soon as possible a glimpse of how the application works and looks but as well as point out unwanted features as they came.

3.2.2 Approach to Prototyping Methodology

Prototyping methodology has many software development life cycle (**SDLC**) phases, at the first stage, all non-functional and functional application requirements are gathered from the stakeholder through multiple conversations, then the second stage was about presenting the designed prototype in order to identify its flaws from his perspective. Refining the prototype according to the gathered feedback was done in the third stage.

3.3 Requirements Analysis

Requirements are the list of functions and features that an application must possess. After several conversations with the stakeholder, Over time all requirements that were needed to be implemented meet his needs were gathered.

My analysis went through many phases such as making a difference between functional and non-functional requirements, setting up a data dictionary for the database metadata, entity–relationship model to better understand the interrelations between the system entities finally a relational data model that literally represents the database's tables.

3.3.1 Function Requirements

Function requirements describes *What The Application Should Do*.

A full list detailed of function requirements can be found here.

Table 10: Function Requirements

Req. No.	Description
R-1	A customer should be able to register with email account
R-2	A customer should be able to view the details of any particular car
R-3	The application should display available cars to the customer
R-4	A customer should be able to cancel a reservation
R-5	A customer should be able to book a car through the application

3.3.2 Non-functional Requirements

Non-functional requirements describes *How The Application Should Behave*.

A full detailed list of non-functional requirements can be found here.

Table 11: Non-functional Requirements

Req. No.	Description
R-1	The application's interface should be user-friendly & easy to use
R-2	The application should be 24/7 available to customers
R-3	Customer's data should be protected from attacks
R-4	The application should maintain data integrity through backups
R-5	The website's load time should not be more than 10 seconds

3.3.3 Data Dictionary

In a Database Management System (**DBMS**), a data dictionary contains database metadata, in other words characteristics of the stored data and relationships between entities.

The full data dictionary can be found here.

Table 12: Data Dictionary

Attribute	Description	Type	Constraints
name	The client's name	VARCHAR	Required
bookingId	An id of a reservation	UUID	Required & Unique
cancelledDate	A reservation's cancelled date	DATE	Not required
brand	The car's model brand	VARCHAR	Required & Unique
costPerDay	The cost of a rented car per day	INTEGER	Required

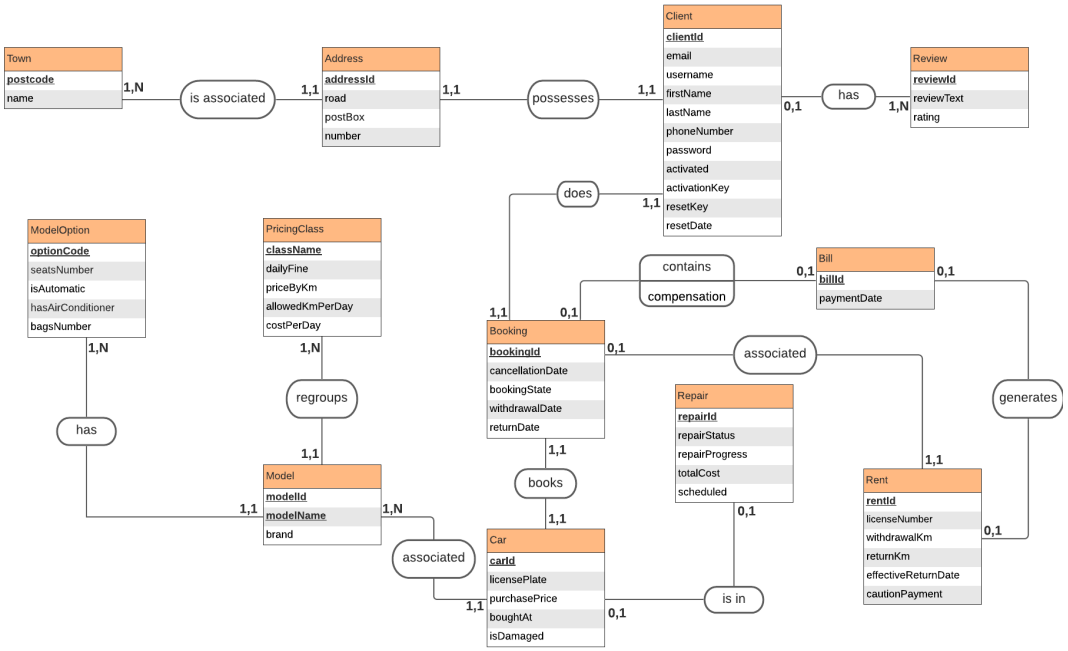
3.3.4 Entity–relationship Model

W

In software engineering, an **Entity–relationship** model is commonly formed to represent things a business needs to remember in order to perform business processes. Consequently, the ER model becomes an abstract data model, that defines a data or information structure which can be implemented in a database, typically a relational database.

— Wikipedia

Figure 2: Entity–relationship Diagram



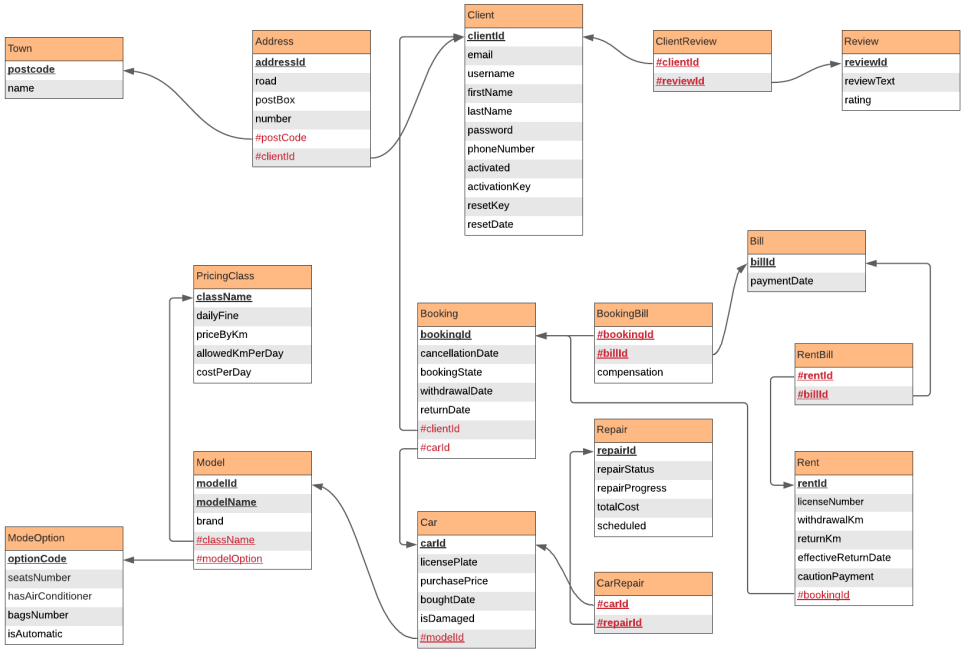
3.3.5 Relational Data Model

W

The purpose of the **relational model** is to provide a declarative method for specifying data and queries: users directly state what information the database contains and what information they want from it, and let the database management system software take care of describing data structures for storing the data and retrieval procedures for answering queries.

— Wikipedia

Figure 3: Relational Diagram



3.3.6 Business Rules

> A **business rule** defines or constrains some aspect of business and always resolves to either true or false. Business rules are intended to assert business structure or to control or influence the behavior of the business.
>
> — Wikipedia

A full detailed list of business rules can be found here.

Table 13: Business Rules

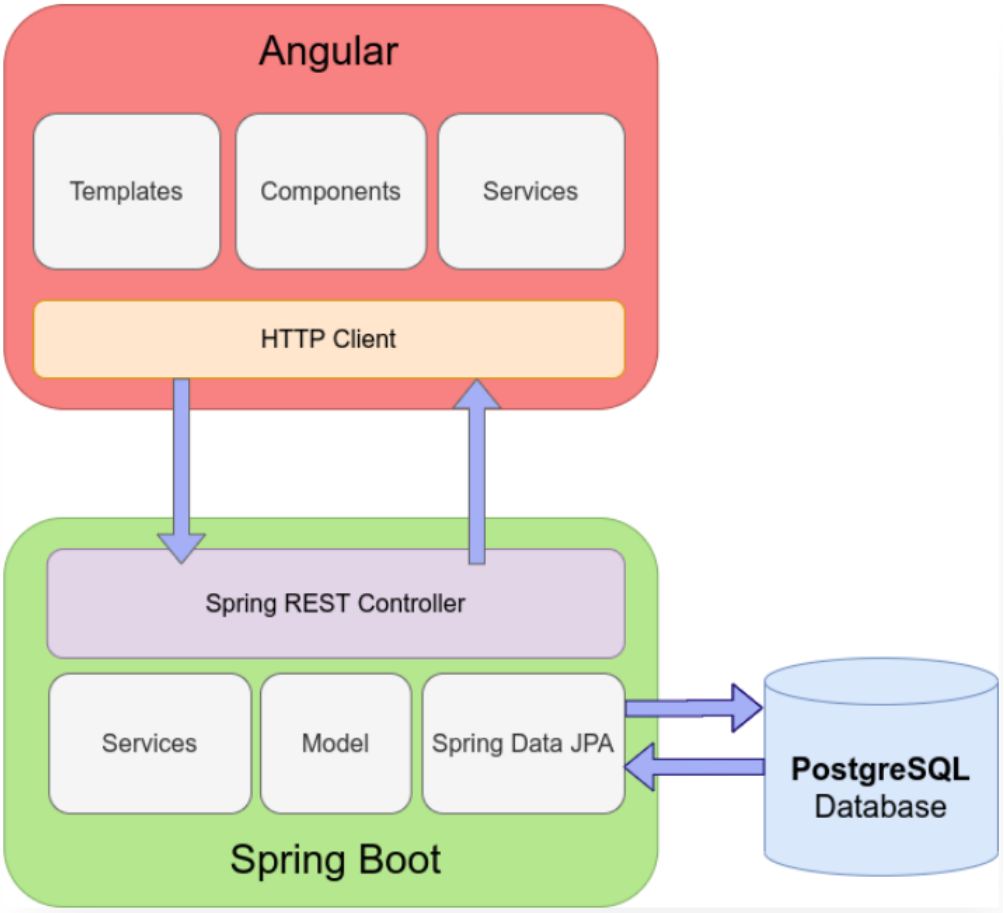
BR. No.	Description
BR-1	Only registered customers can book cars
BR-2	Every car in the system must have model
BR-3	A client must have a driving licence to rent a car
BR-4	A customer can only book one car at once
BR-5	Every rent must be linked to its reservation

4 Chapter IV : Software Design

4.1 System Architecture

The application was developed using the following architecture :

Figure 4: System Architecture



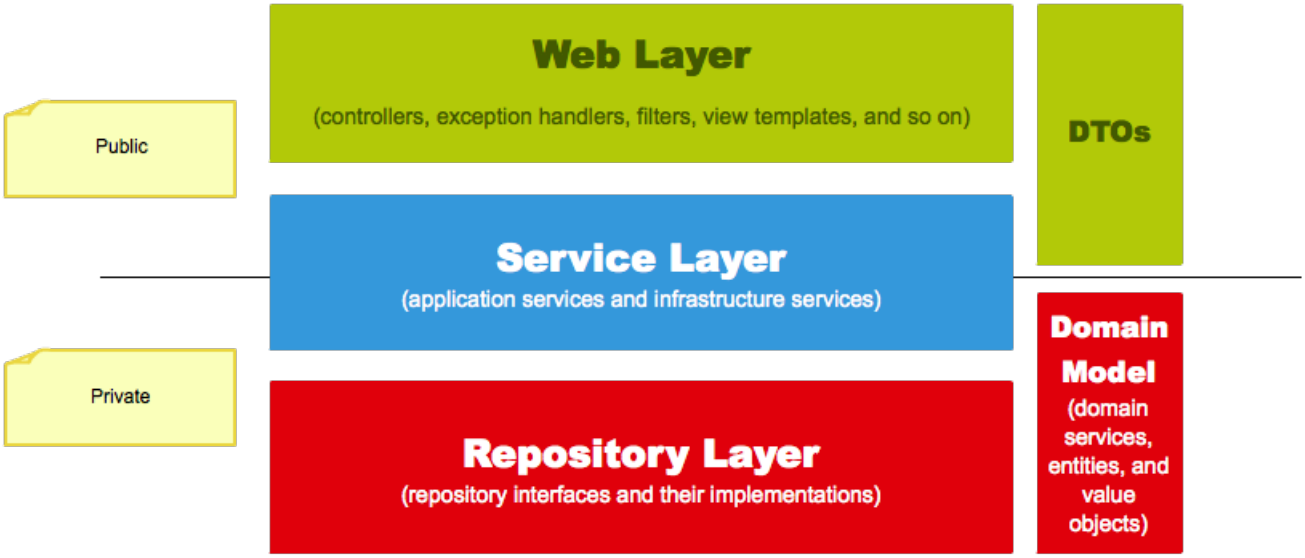
— From frontbackend.com

The system is made of three components :

- The backend Spring Boot application as a REST API to be consumed by the Angular application.
- The frontend Angular application as the client side of this whole system.
- The PostgreSQL database used by the API.

4.2 Backend Architecture

Figure 5: Backend REST API Architecture



— From petrikainulainen.net

4.2.1 Technology stack

- Language : JAVA 15
- Web Framework: Spring Boot
- Build Tool : Maven

  Database :

  - DBMS : PostgreSQL
  - ORM : Hibernate
  - Migrations : Flyway

4.2.2 Why Java ?

- Java Ergonomics

The craftsmanship of JetBrains makes Java really easy to use. Most java features are autocompleted, jump to java doc is really fast, method and class refactoring is done efficiently. However, I gravitated towards Java because I wanted a good and efficient developer experience with third-party libraries. When consuming third-party libraries in Java, you always know exactly what types are needed for a method but most importantly, an incorrect usage of the latter will result into a compilation error.

- Nominal Typing

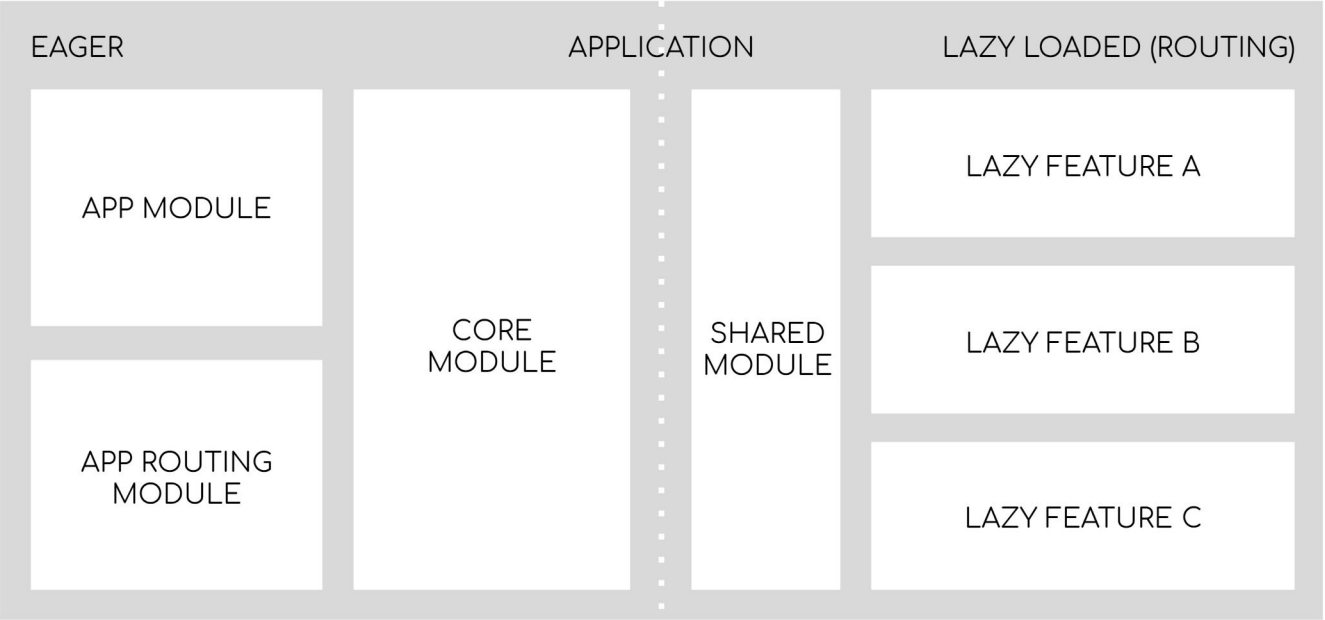
On of the reasons a dynamic typed language wasn't chosen is because I wanted my application to fail at compile time rather than at runtime upon a change in a third-party method. It's completely a waste of time when you have to refer back to the implementation of a method to figure out which type(s) to pass it.

The typed version of Javascript called TypeScript somehow solves this problem, but still lack the ability to validate passed types at runtime without extra code. You have to implement yourself typing/interfaces for you to have some type safety in your application.

4.3 Frontend Architecture

The Angular application was developed using the following architecture:

Figure 6: Frontend Architecture



— From tomastrajan.medium.com

4.3.1 Technology stack

- Language : TypeScript
- Framework: Angular 10
- Build Tool : NPM
- CSS Library: PrimeNG

5 Chapter V : Implementation

5.1 API

The API of this application is simply a Spring Boot Java application running on an embedded Apache Tomcat Server[4]. To test different endpoints of this API, **Postman**[5] was used a testing API tool.

- **API Documentation**

There are very many api documentation tools out there but since the api was developed in java, a built-in tool in IntelliJ IDEA was used to generate the ***javadoc*** for the API. The latter can be found here.

5.1.1 MapStruct - Mapping Library

Since an API is designed to be consumed by other applications, data integrity and security become crucial when designing an API. Most of the time, an external API or the end-user doesn't need to access the entirety of the data from a database model, but only some specific fields. In a such scenario Data Transfer Objects (**DTOs**) come in handy.

> In the field of programming a data transfer object (**DTO**) is an object that carries data between processes. The motivation for its use is that communication between processes is usually done resorting to remote interfaces (e.g., web services), where each call is an expensive operation.
>
> Because the majority of the cost of each call is related to the round-trip time between the client and the server, one way of reducing the number of calls is to use an object (the DTO) that aggregates the data that would have been transferred by the several calls, but that is served by one call only.
>
> — Wikipedia

Since DTOs are a just reflection of objects stored in the database - mappers between DTO classes & model classes play a major role in the conversion process. For this application **MapStruct** was used a mapping library to map a DTO from its entity and contrariwise. It tremendously reduces the amount of boilerplate code which would have had to be written by hand but since it uses annotation-processing to generate mapper class implementations at compile time, all I had to write were interfaces.

[4] Tomcat
[5] Postman

- **MapStruct Example**

The first thing that was implemented for mappers was the Entity Mapper interface for a generic dto to entity.

Listing 1: EntityMapper

```
/**
 * @param <D> - DTO type parameter.
 * @param <E> - Entity type parameter.
*/

public interface EntityMapper<D, E> {

    E toEntity(D dto);

    D toDto(E entity);

    Collection<E> toEntity(Collection<D> dtoList);

    Collection<D> toDto(Collection<E> entityList);

    @Named("partialUpdate")
    @BeanMapping(nullValuePropertyMappingStrategy =
       NullValuePropertyMappingStrategy.IGNORE)
    void partialUpdate(@MappingTarget E entity, D dto);
}
```

This interface is extends by all interface mappers in the application. Listing 4, shows how I was able to map a dto to it's model class.

Let's say we have to map a `Booking` class to it's `BookingDTO` class.

Listing 2: Booking

```
@Entity
public @Data class Booking extends AbstractAuditingEntity {

    @Id
    private UUID id;
    private Instant cancellationDate;

    @Enumerated(EnumType.STRING)
    private BOOKINGSTATE bookingState;

    private Instant withdrawalDate;
    private Instant returnDate;
}
```

Listing 3: BookingDTO

```
public @Data class BookingDTO {

    private String bookingId;

    private Instant cancellationDate;

    @JsonProperty(access = JsonProperty.Access.READ_ONLY)
    private BOOKINGSTATE bookingState;

    private Instant withdrawalDate;

    private Instant returnDate;

    @JsonProperty(access = JsonProperty.Access.READ_ONLY)
    private Instant createdAt;

    @JsonProperty(access = JsonProperty.Access.READ_ONLY)
    private CarDTO carDTO;

}
```

Now, to make a mapper between these two classes, a `BookingMapper` interface was created. By annotating it with `@Mapper`, MapStruct concludes that this is a mapper between the two classes :

Listing 4: BookingMapper

```
@Mapper(componentModel = "spring", uses = {CarMapper.class})
interface BookingMapper extends EntityMapper<BookingDTO, Booking> {

    BookingDTO toDto(Booking booking);

    @Mapping(target = "id", ignore = true),
    Booking toEntity(BookingDTO bookingDTO);

    void partialUpdate(@MappingTarget User user, UserInfoDTO
       userInfoDTO);

}
```

At compile time, the MapStruct annotation processor plugin will pick up the `BookingMapper` interface and generate an implementation for it. The `BookingMapperImpl` class implements all `BookingMapper` interface methods which one of them (`toDto`) maps the `Booking` class fields to the `BookingDTO` class fields.

5.1.2 Mail Service

To send mails to customers upon a successful booking confirmation, **SendGrid API** was used as an Email Delivery Service.

> **SendGrid** (also known as Twilio SendGrid) is a Denver, Colorado-based customer communication platform for transactional and marketing email. It provides a cloud-based service that assists businesses with email delivery. The service manages various types of email including shipping notifications, friend requests, sign-up confirmations, and email newsletters.
>
> — Wikipedia

Listing 5: MailService

```java
@Profile("prod")
@Service
public class MailService {

    private final String sendGridAPIKey;

    public MailService(SendGridConfiguration sendGridConfiguration) {
        this.sendGridAPIKey = sendGridConfiguration.getApiKey();
    }

    public void sendEmailConfirmation(String emailTo) {

        var from = new Email("he201718@students.ephec.be");
        var subject = "Confirm Your Email !";
        var to = new Email(emailTo);
        var content = new Content("text/html", emailTemplate());
        var mail = new Mail(from, subject, to, content);
        var request = new Request();

        try {
            request.setMethod(Method.POST);
            request.setEndpoint("mail/send");
            request.setBody(mail.build());
            SendGrid sg = new SendGrid(sendGridAPIKey);
            Response response = sg.api(request); // 202 if ok
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

5.1.3 Twilio SMS API

To send SMS to customers to remind them for instance when to pick up their rented car or to confirm their phone number, **Twilio API** was used as an SMS Delivery Service.

> **Twilio** is an American cloud communications platform as a service (CPaaS) company based in San Francisco, California. It allows software developers to programmatically make and receive phone calls, send and receive text messages, and perform other communication functions using its web service APIs.
>
> — Wikipedia

5.1.4 Error Handling

Handling exceptions is a crucial part when developing a robust application. Spring Boot offers more than one way of doing it. Since it's version 3.2, there is the `@ControllerAdvice` annotation to unify exception handling across the whole application.

Why is it called "Controller Advice" ?

The term 'Advice' comes from Aspect-Oriented Programming (AOP) which allows us to inject cross-cutting code (called "advice") around existing methods. A controller advice allows us to intercept and modify the return values of controller methods, in our case to handle exceptions.

— From reflectoring.io

Listing 6: ControllerAdvice

```
@ControllerAdvice
public class GlobalExceptionHandler {

    /**
     * A method to handle email duplication across the whole
        application.
     */
    @ExceptionHandler({EmailAlreadyUsedException.class})
    public ResponseEntity<Map<String, String>>
       emailAlreadyUsedException(EmailAlreadyUsedException ex) {
        return ResponseEntity
                .status(HttpStatus.CONFLICT)
                .body(Map.of("message", ex.getMessage()));
    }
}
```

5.2 Database

5.2.1 Overview

This application uses **PostgreSQL** was its database. The latter was chosen firstly because it's an open source database secondly a relational database was a needed to handle all interrelations between entities and lastly because it's a feature-rich database that supports things like Full-text Search and JSON for instance.

5.2.2 Database Migrations - Flyway

Ideally its good to have well-thought-out schema of a database at the beginning of a project but with evolving requirements changes to the initial schema tend to happen quite often. A database schema literally represents the application so changes to the latter must be carefully executed to avoid losing the currently stored data. To tackle this problem **Flyway** was used a database-migration tool.

> In software engineering, schema migration (also database migration, database change management) refers to the management of incremental, reversible changes and version control to relational database schemas. A schema migration is performed on a database whenever it is necessary to update or revert that database's schema to some newer or older version.
>
> — Wikipedia

Flyway migrations can be written in SQL or Java but SQL base migrations were chosen for simplicity.

Listing 7: Booking Table

```sql
CREATE TABLE booking
(
    id                uuid            DEFAULT        PRIMARY KEY,
    cancellation_date TIMESTAMPTZ,
    booking_state     d_booking_state DEFAULT 'OPEN' NOT NULL,
    withdrawal_date   TIMESTAMPTZ                    NOT NULL,
    return_date       TIMESTAMPTZ                    NOT NULL,
    user_id           uuid                           NOT NULL UNIQUE,
    car_id            uuid,
    CHECK (return_date > withdrawal_date),
    FOREIGN KEY (user_id) REFERENCES users (id),
    FOREIGN KEY (car_id) REFERENCES cars (id)
);
```

6 Chapter VI : Security

6.1 Spring Security

What is Spring Security ?

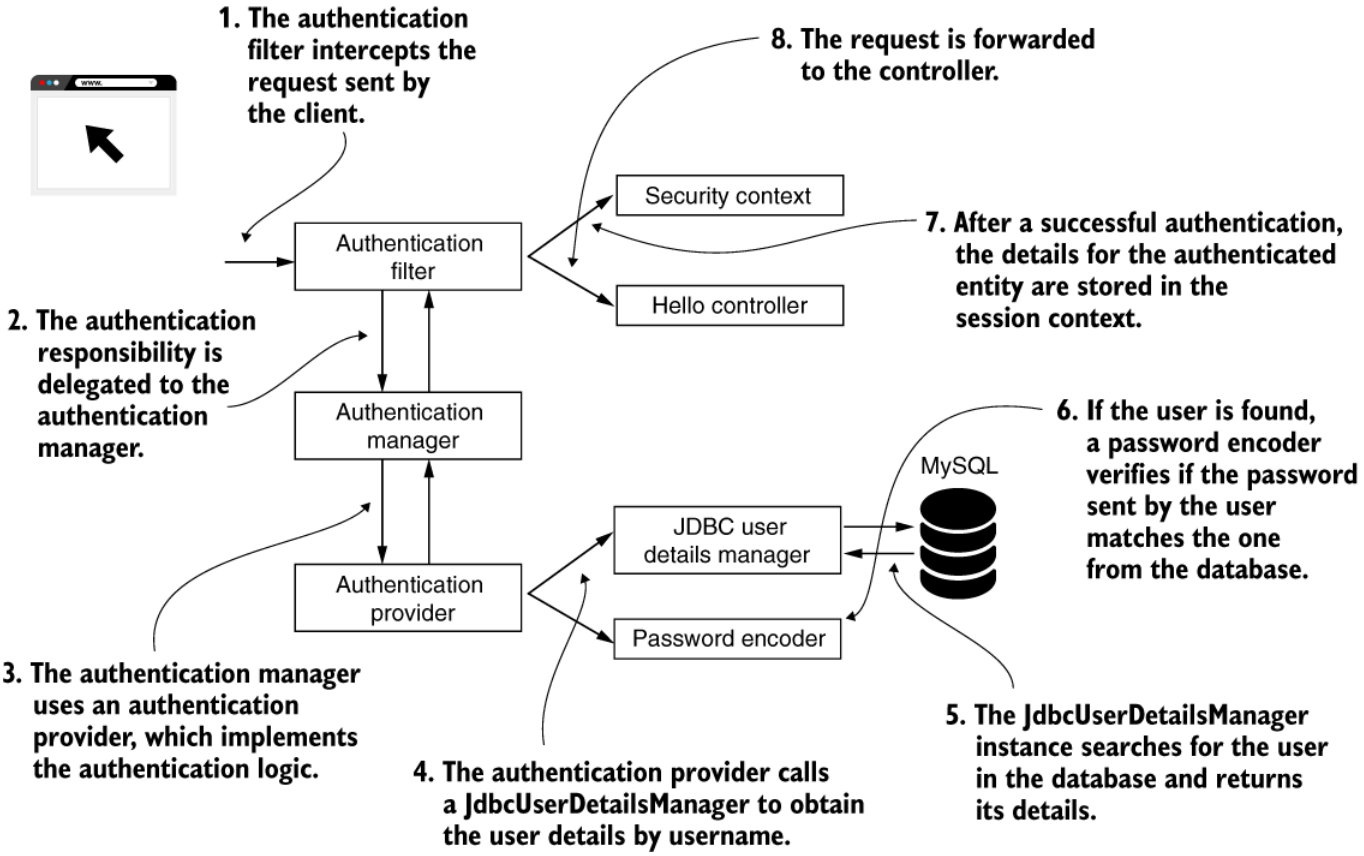
Spring Security is a powerful and highly customizable authentication and access-control framework. It is the de-facto standard for securing Spring-based applications.

Spring Security is a framework that focuses on providing both authentication and authorization to Java applications. Like all Spring projects, the real power of Spring Security is found in how easily it can be extended to meet custom requirements.

— From spring.io

6.1.1 Authentication

Figure 7: Spring Security Authentication Process

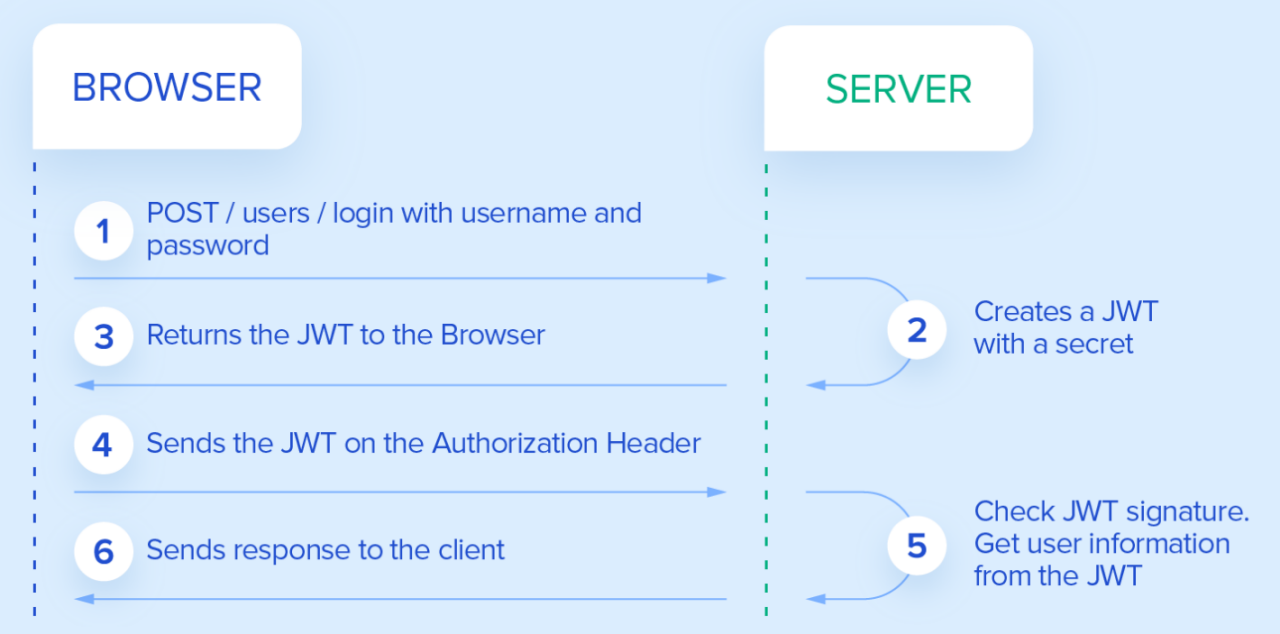


— From Spring Security in Action Book

Figure 7 demonstrates steps that executed to authenticate a user on the backend level. Obviously there is a lot of things going on under the hood during the authenticating process, but the details of the latter are out of scope of this report.

But it's important to note that upon a successful authentication, a JSON Web Token (JWT[6]) with a validity of 72h is sent to the user as shown in figure 8.

Figure 8: JWT Authentication Process



— From toptal.com

6.2 Role-based Access Control - RBAC

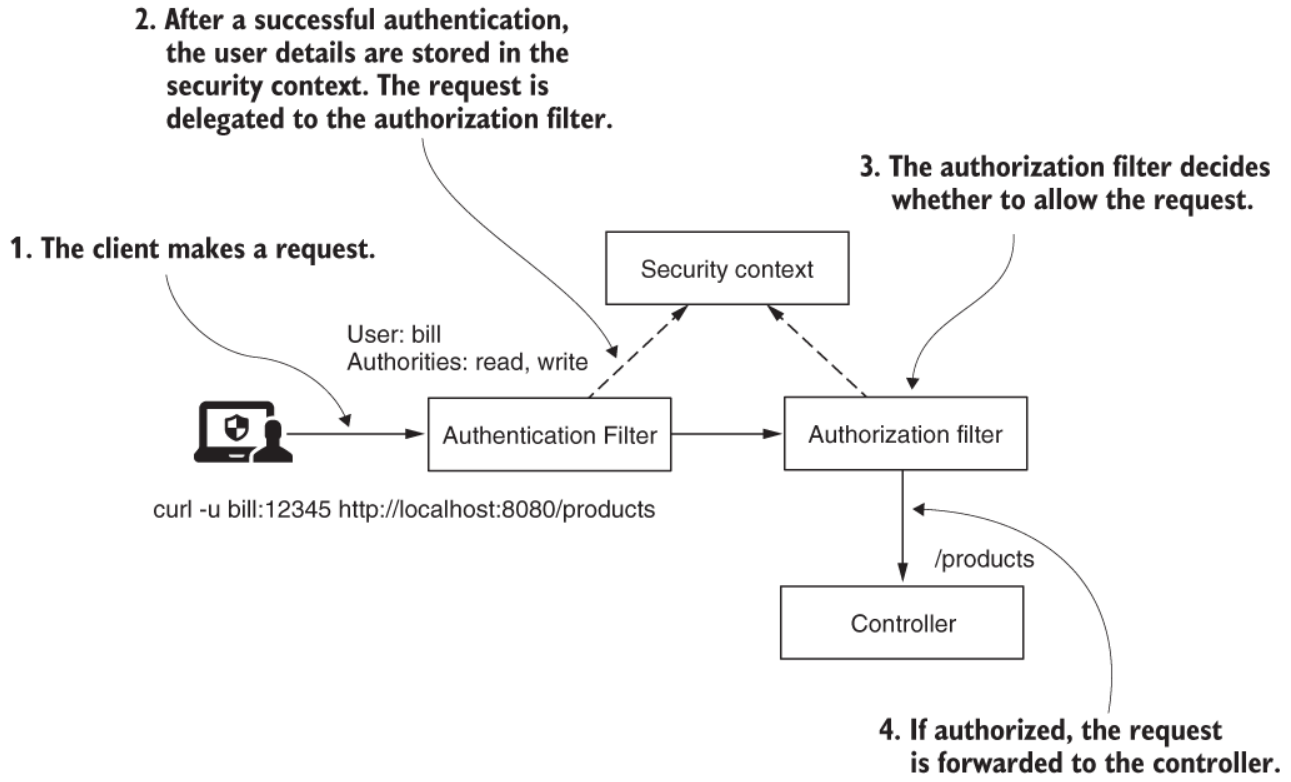
Role-based Access Control, is a mechanism that restricts an application access to users using their roles, privileges and permissions.

To limit what a customer can do, the latter has to be authenticated through a set of credentials verification process. Once successfully authenticated, the customer's role is retrieved from the HTTP Authorization header.

Within this application, roles are created for various user types (e.g., admin, user and anonymous). The permission to perform certain transactions or access a particular route, a specific role is needed. For instance, a user with a role of an `admin` is granted the permission to create, update, read, and delete any profile, whereas a user with a role of a `user` is given access to read & update their own profile.

[6] JWT

Figure 9: Spring Security Authorization Process



— From Spring Security in Action Book

Step 4 shown in figure 9 might sometimes fail so in the effort to make the application more user-friendly, a custom authorization failure handler class was created to deal with users trying to access forbidden routes.

Figure 10: Authorization Fail Message

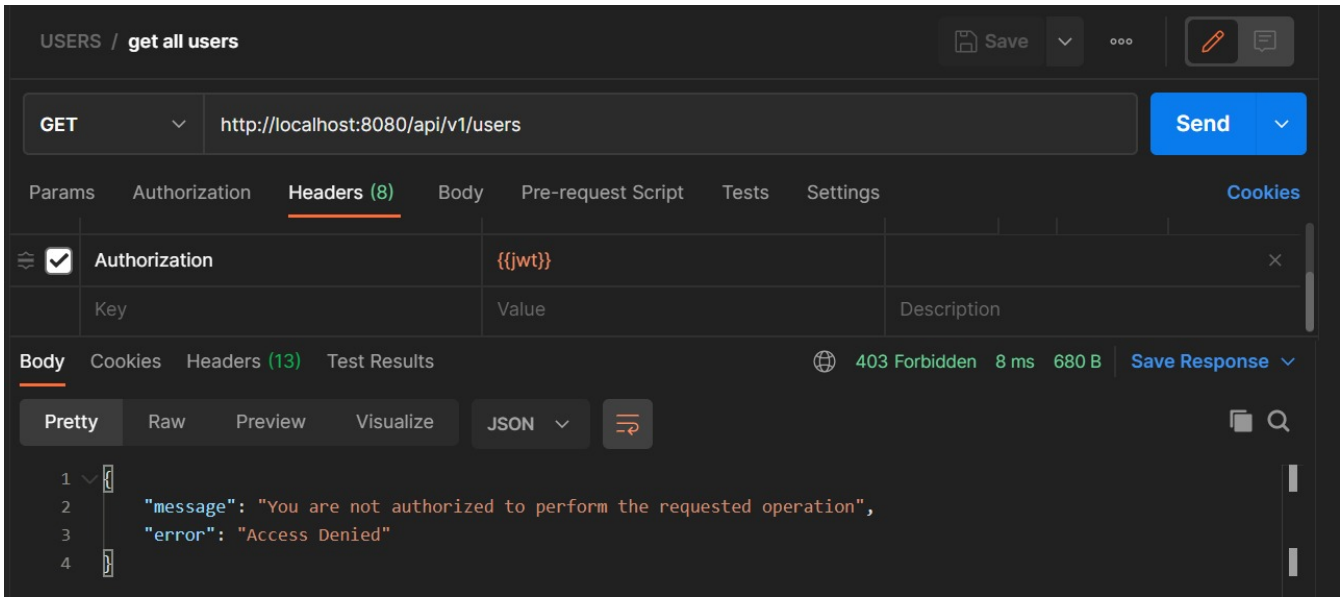


Figure 10 shows an authenticated user with no proper authorization trying to read all customers. Only a user with role of an `admin` is allowed to read all customers stored in the database.

6.3 Open Web Application Security Project - OWASP

> The Open Web Application Security Project (**OWASP**) is an online community that produces freely-available articles, methodologies, documentation, tools, and technologies in the field of web application security.
>
> — Wikipedia

6.3.1 Sensitive Data Exposure

As the name implies, this security threat occurs when the web application does not adequately protect sensitive information like session tokens, emails, passwords, credit card information, location, health data, or any other similar crucial data whose leak can be critical for the customer.

- **Precaution Taken Against Sensitive Data Exposure**

First, I needed to determine what data the application collects that could be considered sensitive. After a thorough study on that matter, information like email addresses, phone numbers, personal address, and passwords should all be considered sensitive information. One way to defend against attackers gaining access to sensitive data is through thorough review of the application code and environment.

The following are some precautions that were taken to mitigate this attack :

1. **Passwords are hashed using strong adaptive and salted hashing functions**.

Listing 8: Password Encoder

```
1     @Bean
2     public PasswordEncoder passwordEncoder() {
3         return new BCryptPasswordEncoder(10);
4     }
```

To achieve that we used a Spring Security's PasswordEncoder whose implementation uses BCrypt strong hashing functions. Bcrypt allows developers to choose the value of saltRounds. The higher this number is, the longer it takes a machine to compute the hash associated with the password.

Having that, it has to be small enough to avoid slowing down the application during the registration or authentification process. By default, the saltRounds value is 10, and that's the one that was chosen. It's good enough.

2. **Website only available via HTTPS**.

If a website accepts a connection through HTTP and redirects to HTTPS, visitors may initially communicate with the non-encrypted version of the site before being redirected, if, for example, the visitor types `http://rent-vehicle.herokuapp.com` or even just `rent-vehicle.herokuapp.com`. This creates an opportunity for a man-in-the-middle attack. The redirect could be exploited to direct visitors to a malicious site instead of the secure version of the site.

We added the HTTP Strict-Transport-Security header that informs the browser that it should never load a site using HTTP and should automatically convert all attempts to access the site using HTTP to HTTPS requests instead. Once a browser sees this header, it will only visit the site over HTTPS for the next 365 days:

Listing 9: Strict-Transport-Security Header

```
@Override
public void configure(HttpSecurity http) throws Exception {
    http
        .headers()
            .hsts()
                .includeSubdomains(true)
                .maxAge(Duration.ofDays(365));
}
```

> For the moment the site is deployed on Heroku so out of the box the site is already in HTTPS, but this header added was added in case future developers choose another hosting platform other than Heroku.

6.3.2 Referrer Policy

What's a referrer ?

The Referrer-Policy HTTP header controls how much referrer information (sent via the Referer header) should be included with requests.
For example, if you click a link on ***https://rent-vehicle.herokuapp.com/index.html*** that takes you to ***https://www.ephec.be***, Ephec's servers will see Referer: ***https://rent-vehicle.herokuapp.com***. This can have privacy implications.

- **The referrer problem**

If you are visiting a site that knows your identity (i.e. any site you're logged into), then this site may receive referrer URLs of other pages on the web that you have visited. For example, you may visit a web page about a particular medical condition, sites may associate your identity with having visited that particular medical webpage.

- **How was this fixed ?**

A Referrer-Policy header with a directive of `strict-origin-when-cross-origin` was added on the backend server to control what information is sent through the Referer header. This referer header indicates where the inbound visitor came from, but there are cases where a developer may want to control or restrict the amount of information present in this header like the path or even whether the header is sent at all.

Here is an example :

Table 14: Referrer Policy

Source	Destination	Referrer
https://rent-vehicle.be/blog1	https://facebook.com	https://rent-vehicle.be
https://rent-vehicle.be/blog1	http://facebook.com	NULL
https://rent-vehicle.be/blog1	http://rent-vehicle.be/blog2	NULL
https://rent-vehicle.be/blog1	https://rent-vehicle.be/blog2	Source

Listing 10: Referrer Policy

```
@Override
public void configure(HttpSecurity http) throws Exception {
   http
        .headers()
            .referrerPolicy(
                ReferrerPolicyHeaderWriter
                    .ReferrerPolicy.STRICT_ORIGIN_WHEN_CROSS_ORIGIN)
}
```

6.3.3 Strict Transport Security

- **The attack**

HTTPS is a secure protocol. HTTP, by contrast, is not. HTTP's users are vulnerable to person-in-the-middle attacks and nothing sent is encrypted. I like to think of insecure HTTP as "anything goes."

- **The header**

As explained above the Strict-Transport-Security HTTP header tells browsers to stick with HTTPS and never visit the insecure HTTP version. Once a browser sees this header, it will only visit the site over HTTPS for the next 365 days.

See Listing 9: Strict-Transport-Security Header

6.3.4 X Frame Options

- **The attack**

The attacker wants the end user to click on something the latter actually does not want to click on, but they may hide a button behind something else then trick the user into clicking that button. Clickjacking can be used to get you to click on anything you don't want to. These things include unintentional endorsements on social networks, clicking advertisements etc…

- **The header**

What is X-Frame-Options ?

The X-Frame-Options HTTP response header can be used to indicate whether a browser should be allowed to render a page in a <frame>, <iframe>, <embed> or <object>. Sites can use this to avoid click-jacking attacks, by ensuring that their content is not embedded into other sites.

— From developer.mozilla.org

When browsers load iframes, they'll check the value of the X-Frame-Options header and abort loading if it's not allowed. The header has three options, but the one that was chosen is `X-Frame-Options : DENY`

Listing 11: X-Frame-Options

```
1  @Override
2  public void configure(HttpSecurity http) throws Exception {
3     http
4        .headers()
5            .frameOptions()
6                .deny()
7  }
```

6.3.5 Insufficient Logging And Monitoring

To detect easily why & when the website was/is down. An **uptimerobot**[7] monitor was set up to send a message to the admin just in case the website went down.

Figure 11: Logging And Monitoring

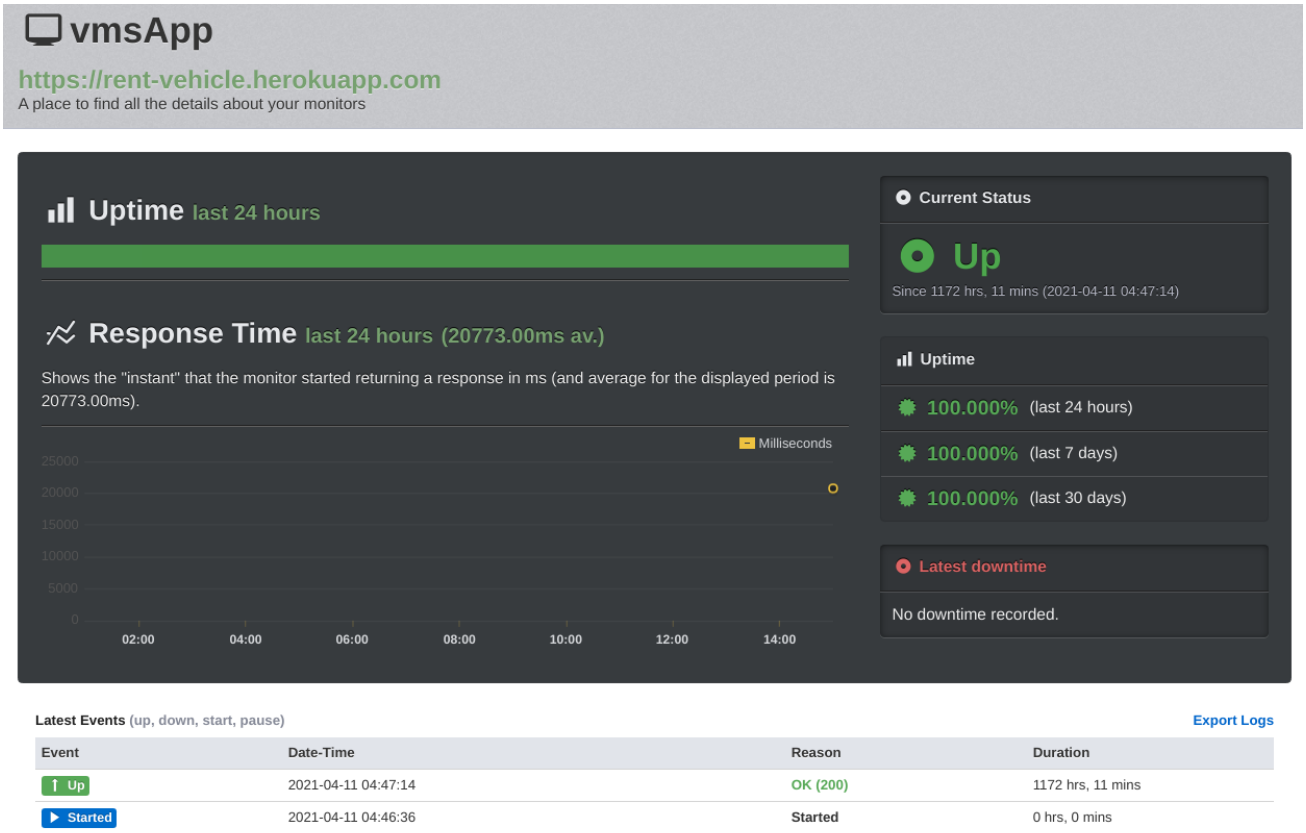


Figure 11 shows this application has been up & running for more 50 days and never went down !

[7] uptimerobot

7 Chapter VII : Application Testing

A full Tests report can be found here.

7.1 Unit Test

Unit testing an applications has many benefits such as ensuring every component of the latter works as expected, facilitates code refactoring & design improvement and more importantly reduces bug fixing cost.

JUnit was used as a unit testing tool in this application.

Listing 12: Assert username can be found

```
1  @Test
2  @DisplayName("assert that user can be found by username")
3  public void assertThatUserCanBeFoundByUsername() {
4    var userDetails = domainUserDetailsService.loadUserByUsername(
       USER_ONE_LOGIN);
5    assertThat(userDetails).isNotNull();
6    assertThat(userDetails.getUsername()).isEqualTo(USER_ONE_LOGIN);
7  }
```

7.2 Integration Test

Listing 13: Should get one car by its id

```
1  @Test
2  @Transactional
3  @DisplayName("should get one car by its id")
4  public void getCar() throws Exception {
5
6    // Initialize the database
7    var savedCar = carDAO.saveAndFlush(car);
8
9    restCarMockMvc
10     .perform(get("/api/v1/cars/{id}", savedCar.getId()))
11     .andExpect(status().isOk())
12     .andExpect(jsonPath("$.isDamaged").value(DEFAULT_IS_DAMAGED))
13     .andExpect(jsonPath("$.licensePlate").value(DEFAULT_LICENSE_PLATE))
14 }
```

Integration testing plays a major role in the application development cycle by making sure the back-end endpoints work as expected. To achieve the latter, **MockMvc**[8] was used to test the web layer and perform requests against a mocked servlet environment. There were no real HTTP request going around during the integration tests since the latter are executed in a mocked environment provided by Spring.

7.3 Architecture Testing - ArchUnit[9]

What is ArchUnit ?

ArchUnit is a free, simple and extensible library for checking the architecture of your Java code. That is, ArchUnit can check dependencies between packages and classes, layers and slices, check for cyclic dependencies and more. It does so by analyzing given Java bytecode, importing all classes into a Java code structure. ArchUnit's main focus is to automatically test architecture and coding rules, using any plain Java unit testing framework.

— From archunit.org

7.3.1 ArchUnit To The Rescue

A common problem that occurs quite often in a Software Development Life Cycle is that code implementation can often diverge from the initial design and architecture. This is where ArchUnit comes into play to test that a code implementation is consistent with the initially defined design.

One of the main reason of software architecture, when it comes to codebase, is maintainability, and to keep an application maintainable, testable, and extendable, an effort to make sure it is modular, interdependencies are as small as possible and there no loose coupling between components needs to be made.

Furthermore, in an evolving application, implementation rules evolve over time. The use of ArchUnit tests forces us developers to purposely accept deviations from predefined rules instead of haphazardly encountering them later in a test feature or actually never. ArchUnit issues warnings about deviations.

ArchUnit allowed me to test the following :

- Package dependency
- Class dependency
- Naming conventions
- Layered packages constraints

[8] MockMvc
[9] ArchUnit

Listing 14: Layer Dependencies

```
1   @Test
2   @DisplayName("layer dependencies should be respected")
3   public void layer_dependencies_are_respected() {
4
5     layeredArchitecture()
6       .layer("Controllers").definedBy("be.rentvehicle.web..")
7       .layer("Services").definedBy("be.rentvehicle.service..")
8       .layer("Persistence").definedBy("be.rentvehicle.dao..")
9
10      .whereLayer("Controllers").mayNotBeAccessedByAnyLayer()
11      .whereLayer("Services").mayOnlyBeAccessedByLayers("Controllers")
12      .whereLayer("Persistence").mayOnlyBeAccessedByLayers("Services")
13      .ignoreDependency(DomainUserDetailsService.class, UserDAO.class)
14      .check(importedClasses);
15  }
```

7.4 End-to-End Test - Puppeteer[10]

E2E tests live demo here.

An end-to-end tests are all about testing an application from the end customer's perspective and making sure business requirements are met. To achieve this feat, a Node library called Puppeteer was used.

7.4.1 Puppeteer

Puppeteer is a browser automation Node library from the Google Chrome team which enables developers to control Chrome/Chromium and execute common actions, much like in a real browser - programmatically, through a decent API. Simply put, it's a super useful and easy tool for automating, testing and scraping web pages. It runs headlessly by default, but can be configured to run in headful mode which helps while debugging.

It's important to note that puppeteer is not a testing tool. It doesn't come with built-in testing assertions, it's just a tool for controlling the browser. But if it's used along with real testing tools such as Mocha & Chai, we get a fascinating framework that can test the application as if it was being used by a customer. In other words the combination of Mocha & Chai and Puppeteer helps test an application from the costumer's perspective.

[10] Puppeteer

Listing 15: E2E Test Example

```
describe('/POST Login', async () => {

   it('it should login a user', async () => {

       const pseudo = await page.$('#pseudo'); //
       const password = await page.$('#password');
       const submit = await page.$('#submitBtn');

       await pseudo.click({ clickCount: 3 });
       await pseudo.type('john@gmail.com');

       await password.click({ clickCount: 3});
       await password.type('az#azx=3*ùâzH2é01');

       await submit.click();
       await page.waitForNavigation();

       expect(page.url()).eql('http://localhost:4200/gallery');

       const logoutBtn = await page.$('#logoutBtn');
       await logoutBtn.click();
       await page.waitForNavigation();

       expect(page.url()).eql('http://localhost:4200/login');

   });
});
```

8 Chapter VIII : General Data Protection Regulation - GDPR

8.1 Respect of the private life

As little data as possible is collected and kept on the website to respect GDPR guidelines. Therefore, inactive customers are deleted within 3 days since there is no need to keep their information stored in the database.

Listing 16: GDPR Example

```
/**
 * Not activated users should be automatically deleted after 3 days.
 * This is scheduled to get fired everyday, at 01:00 (am).
 */

@Override
@Scheduled(cron = "0 0 1 * * ?")
public void removeNotActivatedUsers() {
  userDAO
    .deleteNotActivateUsers(Instant.now().minus(3, ChronoUnit.DAYS))
    .forEach(userDAO::delete);
}
```

8.2 Future Enhancements

The GDPR requires two major things, one to be as transparent as possible about how a website collects customer's personal data, and secondly to collect only the data that is necessary.

To fully comply with GDPR, the following should be incorporate in this application website:

- **The website runs on a secure https connection**. ✓
- **Set up a privacy policy**: Make it available to customers by integrating it into the website's footer. The latter will allow customers to know exactly how their data is collected and for what purpose. It must also specify on which legal mentions this privacy policy is based on. ×
- **Implement Informed Consent** Informed consent means the customers need to know what data processing activities the company intend to conduct, for what the purpose. It should be clearly stated that customers can withdraw their consent at any time.

  It also means that the explanation of the data collecting activities, and their purpose are explained in plain language ("in an unequivocal and easily accessible page, using clear and plain language"). That means no technical jargon or legalese. Any customer accessing the company's services should be able to understand what the company is asking them to agree to. ×

9 Chapter IX : DevOps

9.1 Web Hosting Service - Heroku

What is Heroku ?

Heroku is a platform as a service based on a managed container system, with integrated data services and a powerful ecosystem, for deploying and running modern apps. The Heroku developer experience is an app-centric approach for software delivery, integrated with today's most popular developer tools and workflows.

— From heroku.com

9.2 Deployment Process

The Spring Boot application was locally built & packaged and then deployed a jar to Heroku. This is achieved using Heroku CLI Deploy Plugin[11] which deploys a jar or war file to Heroku.

Here are the steps that were followed to deploy the packaged jar on Heroku :

1. Install Heroku Deploy CLI Plugin

```
heroku plugins:install heroku-cli-deploy
```

2. Package Spring Boot application

```
mvn clean package
```

3. Deploy the jar file on Heroku

```
heroku deploy:jar vms.jar --app rent-vehicle
```

4. Port binding

   Heroku requires an application to listen on the port passed through `$PORT` environment variable. A Procfile[12] file was used to customize the command that is used to run the application. The customized command includes the server.port configuration that is used by Spring Boot to bind the app.

```
web: java $JAVA_OPTS -jar vms.jar -Dserver.port=$PORT $JAR_OPTS
```

[11] Heroku CLI Deploy Plugin
[12] Procfile

10 Conclusion

10.1 Daniel Olivier

Final Word

The development of this application taught me a lot. I had the opportunity to improve myself in the Java programming language ecosystem by using multiple libraries to develop some features. Overall im completely gratified with the final product. The application is fully operational and bug-free, customers can book their cars anytime - any day - anywhere !

"**Truth can only be found in one place: the code.**"

— Robert C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship

11 Bibliography

Books and Libraries consulted for this project

- Raoul Urma, Mario Fusco, Alan Mycroft, **Modern Java in Action**.
- Laurenţiu Spilcă, **Spring Security in Action**.
- Robert Cecil Martin, **Clean Code**.
- Elisabeth Freeman, Kathy Sierra, **Head First Design Patterns**.
- Dan Bergh, Daniel Deogun, and Daniel Sawano, **Secure By Design**.
- Craig Walls, **Spring in Action, Sixth Edition**.
- Jeremy Wilken, **Angular in Action**.
- Nicolai Parlog, **The Java Module System**.
- Joshua Bloch, **Effective Java**.

- SendGrid Java API Library
- TOPSIS Algorithm NPM Library
- Java Architecture Test Library
- Eisvogel Template to convert markdown files to PDF or LaTeX
- MapStruct Library for generating type-safe bean mappers
- FullCalendar Library for full-sized drag & drop event calendar

https://www.logicbig.com/tutorials/java-ee-tutorial/jpa.html

https://www.marcobehler.com/guides/spring-security

https://stackabuse.com/mapstruct-in-java-advanced-mapping-library

https://benjiweber.co.uk/blog/2020/10/03/sealed-java-state-machines

https://blog.scottlogic.com/2020/01/03/rethinking-the-java-dto.html

https://www.thinktocode.com/2018/01/08/repository-pattern

https://www.baeldung.com/java-streams-find-list-items

https://blogs.oracle.com/unit-test-your-architecture-with-archunit

https://spring.io/guides/gs/scheduling-tasks

https://auth0.com/blog/spring-boot-authorization-tutorial-secure-an-api-java

---

https://codecraft.tv/courses/angular/http/http-with-observables

https://dzone.com/articles/understanding-angular-route-resolvers-by-example

https://blog.scottlogic.com/2020/10/01/reducer-builder.html

https://medium.com/@gregradzio/solid-angular-smart-components-using-tdd

https://blog.thoughtram.io/angular//advanced-caching-with-rxjs.html

https://www.primefaces.org/primeng/showcase/#/fullcalendar

https://oasisdigital.com/class/rxjs-angular

https://www.digitalocean.com/community/tutorials/angular-interceptors

---

https://flywaydb.org/documentation/concepts/migrations

https://stackabuse.com/hibernate-with-spring-boot-and-postgresql

https://www.postgresql.org/docs/9.5/sql-createdomain.html

https://www.postgresql.org/docs/9.4/uuid-ossp.html

https://www.postgresql.org/docs/9.4/ddl-constraints.html