EPHEC

Development Of Car Rental Management System

Rapport de stage

Daniel Olivier

07 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 the ideas in 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 to accomplish this task.

Daniel Olivier
Louvain-La-Neuve, 10 May 2021.

Abstract

This Car Rental System project is designed to aid my client manage his cars up for rent through an online system. The latter has many features such as users can search for available cars; view profile; book the car for the time period; administrator can keep track of all the customers' information.

This system increases customer retention and maximizes efficiency when it came to automating all car reservation tasks and receiving up-to-date important business statistics.

By using this system, the admin can manage their rental, bookings, profiles, car details etc. It has a user-friendly interface which helps the user to check for cars and rent them for the period specified. Upon a successful reservation the customer receives a confirmation email detailing everything he/she needs to know about their reservation.

The main focus of this project is to help any user easily find a car that available in the system as well as help my client manage rented cars and users in the system efficiently.

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

1 Chapter 1 : 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 my client his services accessible to the public through the web and furthermore keep and maintain records about reservations.

The main functions of this application is to keep tracks of vehicles, customers and booking. It provides useful information to the admin such as making reports of vehicles to be delivered/picked up and acts as a vehicle management system by monitoring the use and price of the vehicles. From the user point of view, this application is basic to diverse individuals' arrangement to travel or move from one place to another for business or school purposes, tour, and visit occasions. Thus, making this application 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.

My client has some personal vehicles at disposal for people looking to rent a car for a given period. But for the time being my client runs fully his business on his phone(s) by receiving calls of people who want to rent some of his vehicle, or those that need a car for a certain period, but does not have the desire or opportunity to buy it.

He wants to go digital and to accept online reservations and manage his fleet with ease. In today's digital environment, users 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 to implement a well-designed car rental system to help not only him accept online reservations and manage his entire fleet but as well as his potential customers reserve available cars online. This car rental system should allow him to run his business smoothly and effectively.

1.3 Stakeholder(s)

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

1.4 Aim and Objectives

Below is 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 accepting online reservations,
- Removing the paper-based processes,
- Detailed analytics and statistics, i.e., the software should deliver an up-to-date analytics to see how his business performs,
- Avoiding risks of overbooking and the factor of human error,
- The reservations' timeline to track the status of the vehicle due for maintenance, for delivery or pick-up, or currently on the road,
- Cuts down on administration processes to improve business efficiency,
- Making 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 a system is there to define its boundaries. In other words, what is in scope and what is out of scope.

The scope can be presented from user aspect of view :

- 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 (my client).
- Rented cars should be used at the moment in one country Belgium
- No mobile application will be developed for this car rental web application.

2 Chapter 2 : 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. References are made to source from the internet.

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

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).
>
> — 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 hypnotically 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 three criteria as 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 we convert the class linguistic terms using the 5 point scale, we get the following table :

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 will be normalised. Divide the data with their own criteria performance values as shown in Table 3 to get result.

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 3 : Software Requirement Specifications - SRS

3.1 Overview

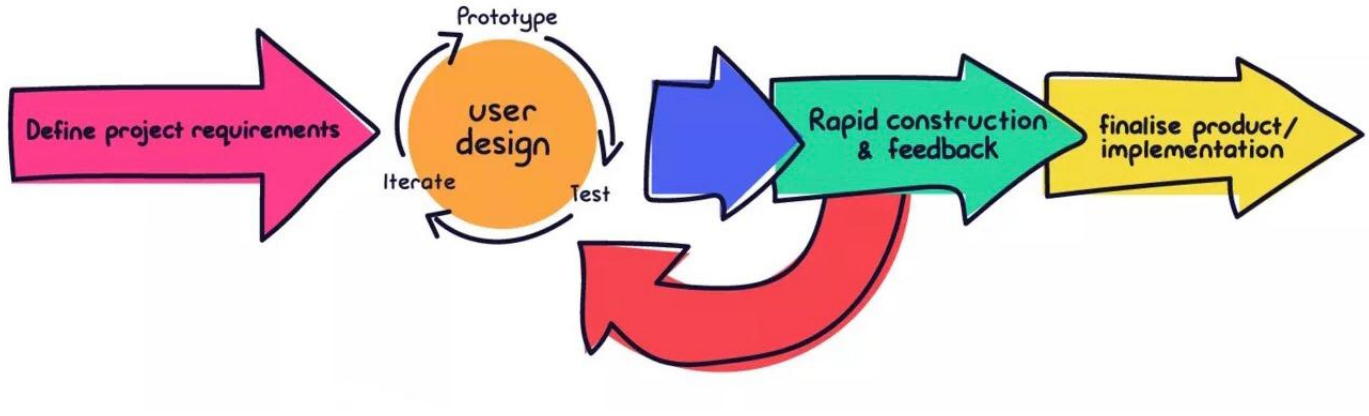
This chapter contains the analysis and design of car rental web based application system, the type of the methodology, function and non-functional requirements of this application and software tools that were used during the development phase. In addition, this chapter contains UML diagrams, the database design, application architecture to better understand how the application 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 application 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, what I did was gather all non-functional and functional application requirements are gathered from the stakeholder by talking him, then moved to the second stage by designing of the application which was presented to him as a preview of the application system in order to identify flaws of the latter. The third stage was about refining the prototype according to feedback I was able to gather from him.

3.3 Requirements Analysis

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

My analysis went through many phases such as making a difference between functional and non-functional requirements, setting up a data dictionary for database metadata, entity–relationship model to better understand how entities in the system are related and finally a relational data model that 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 to 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

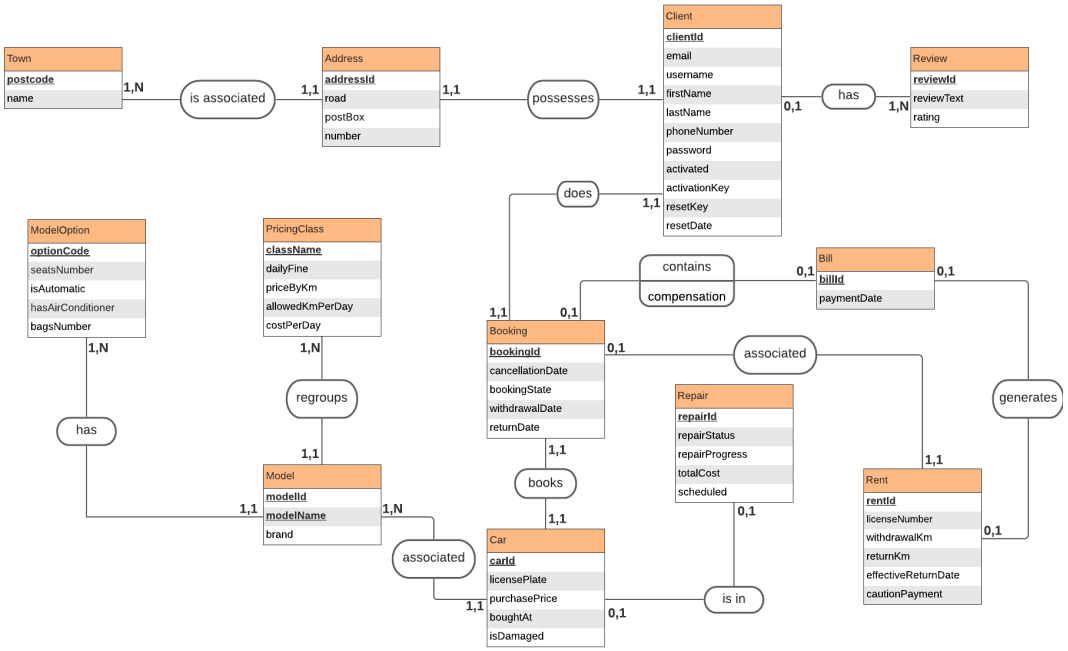
3.3.4 Entity–relationship Model

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



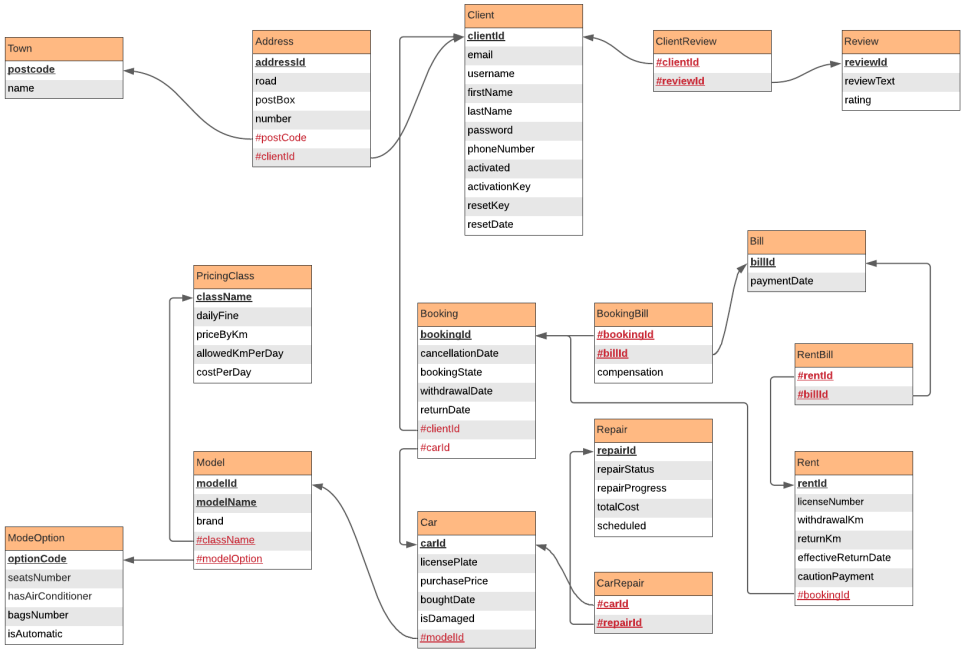
3.3.5 Relational Data Model

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

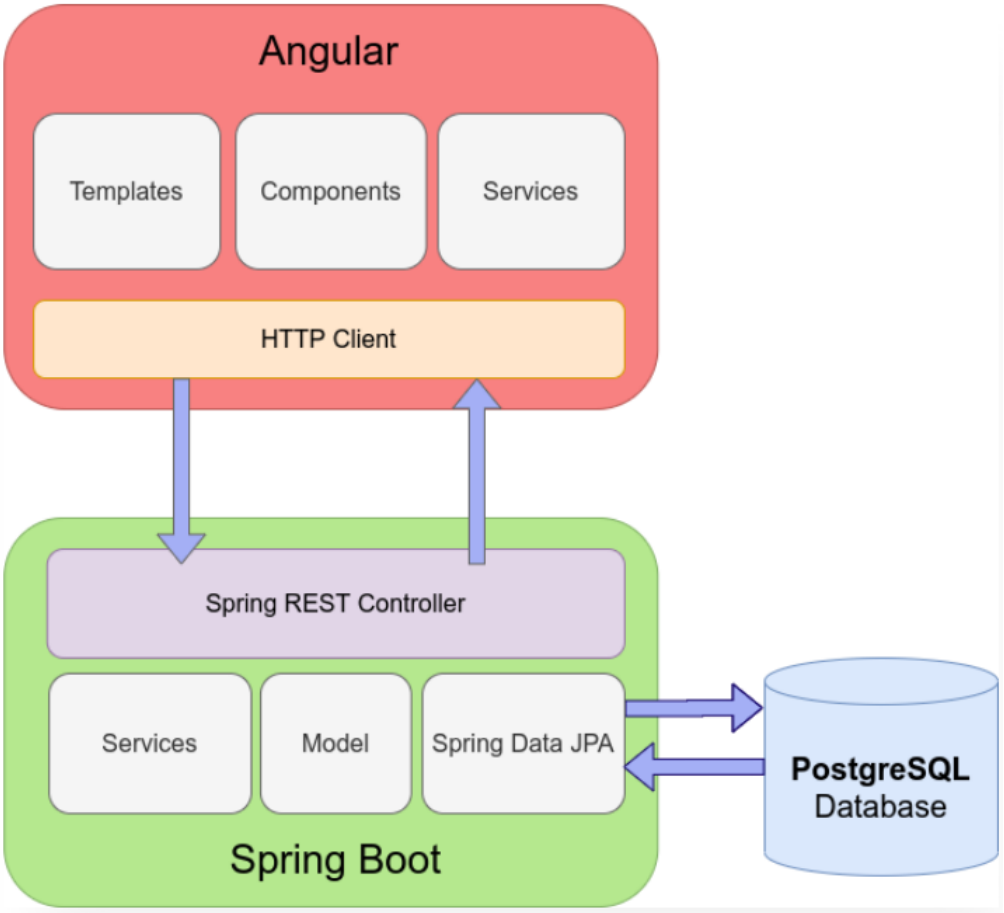
BR. No.	Description
BR-1	Only registered customers can book cars
BR-2	Every car in the system must have model
BR-3	A client must have a driving licence to rent a car
BR-4	A customer can only book one car at once
BR-5	Every rent must be linked to its reservation

4 Chapter 4 : Software Design

4.1 System Architecture

The application was developed using the following architecture:

Figure 4: System Architecture



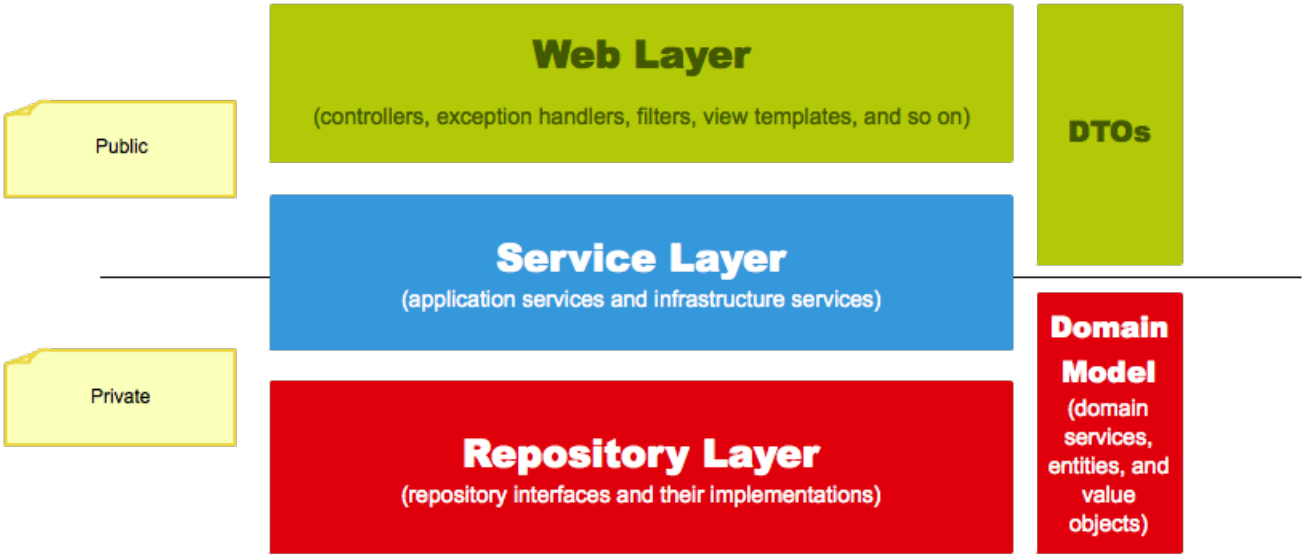
— From frontbackend.com

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

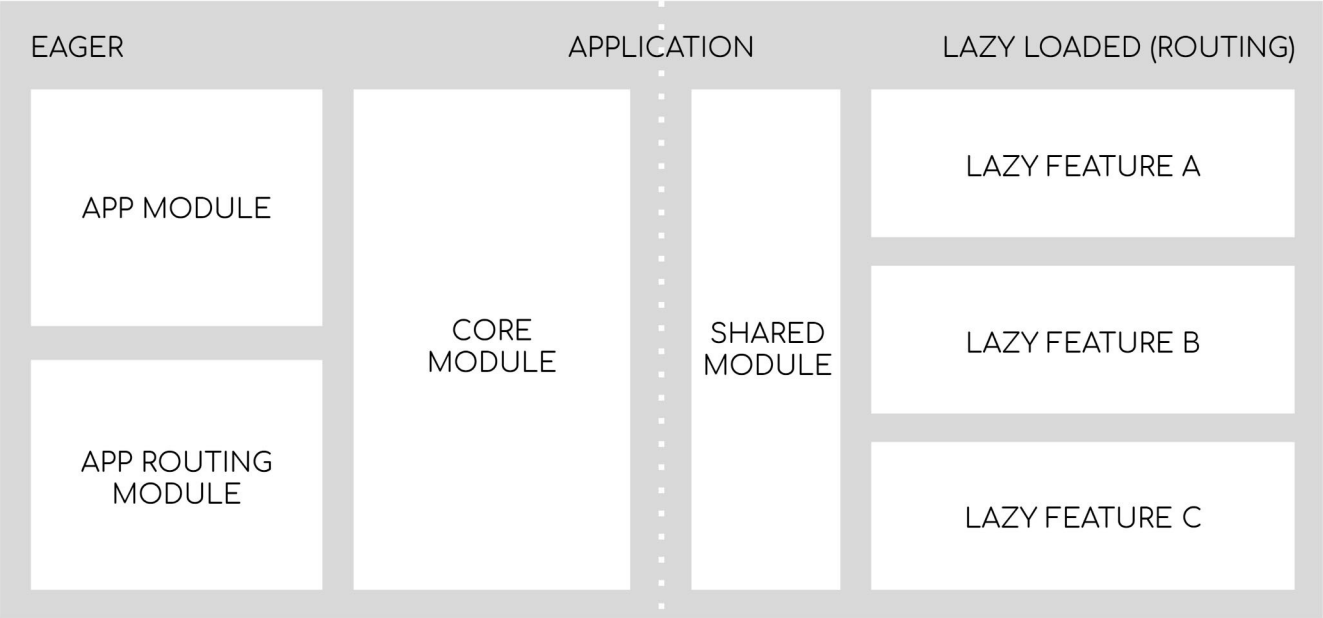
On of the reasons a dynamic typed language wasn't chosen is because I wanted my application to fail at build time rather than at runtime upon a change in a third-party method. It's completely a waste of time when you have to refer back to the implementation of a method to figure out which type(s) to 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 5 : 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**

An undocumented API is not that useful, 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 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 such scenarios 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 2, 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 our 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 maps our `Booking` fields to the `BookingDTO` fields and contrariwise.

5.1.2 Mail Service

To send mails to customers for instance 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 sendGridAPI;

    public MailService(SendGridConfiguration sendGridConfiguration) {
        this.sendGridAPI = 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(sendGridAPI);
            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 6 : Security

6.1 Spring Security

What is Spring Security ?

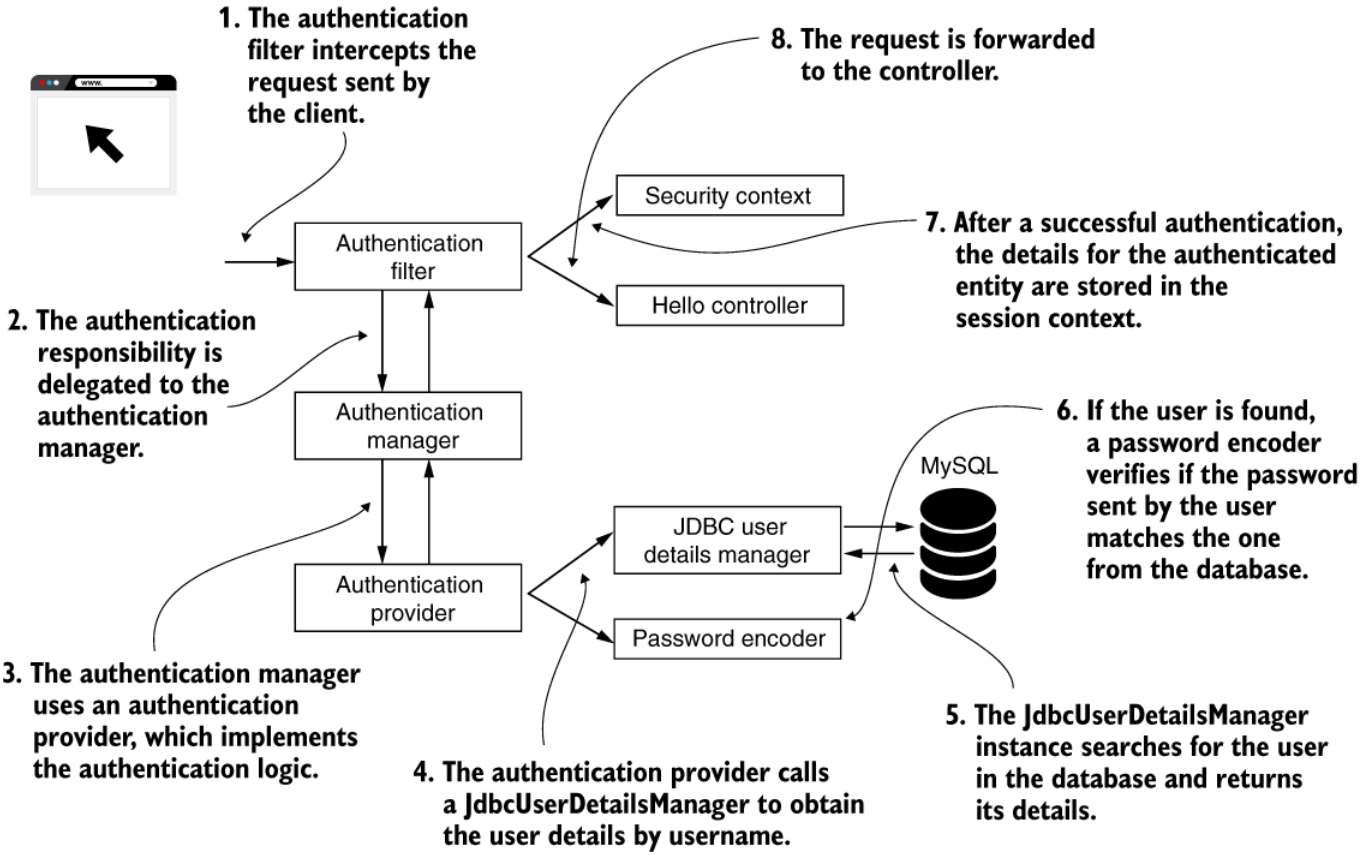
Spring Security is a powerful and highly customizable authentication and access-control framework. It is the de-facto standard for securing Spring-based applications.

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

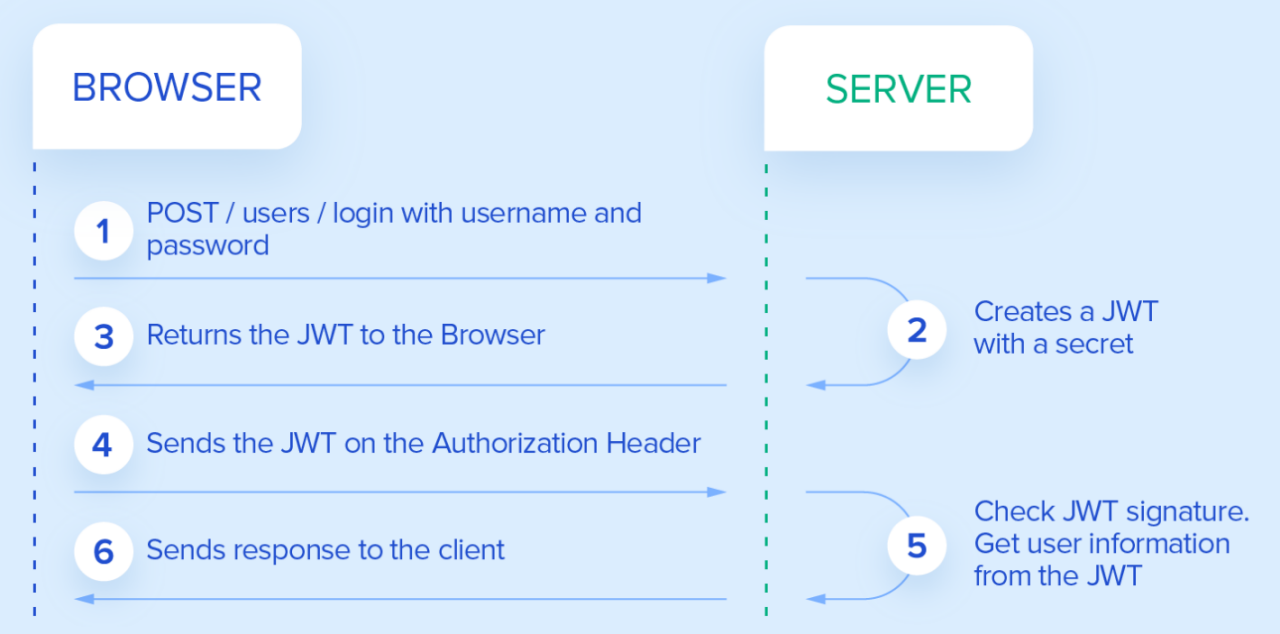


— From Spring Security in Action Book

Figure 7 demonstrates steps that executed to authenticate a user in this application. 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

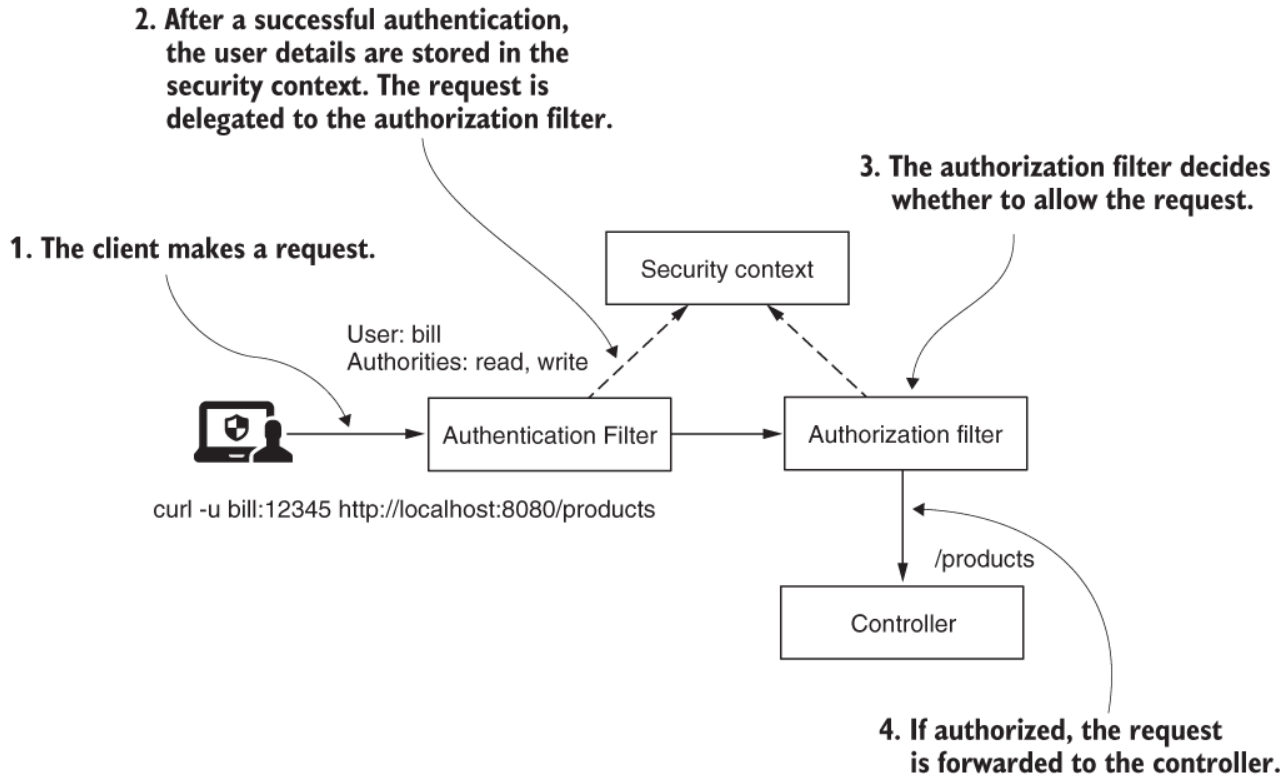
Role-based Access Control, is a mechanism that restricts an application access to users using their roles, privileges and permissions.

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

Within this application, roles are created for various user types (e.g., admin or 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 his 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

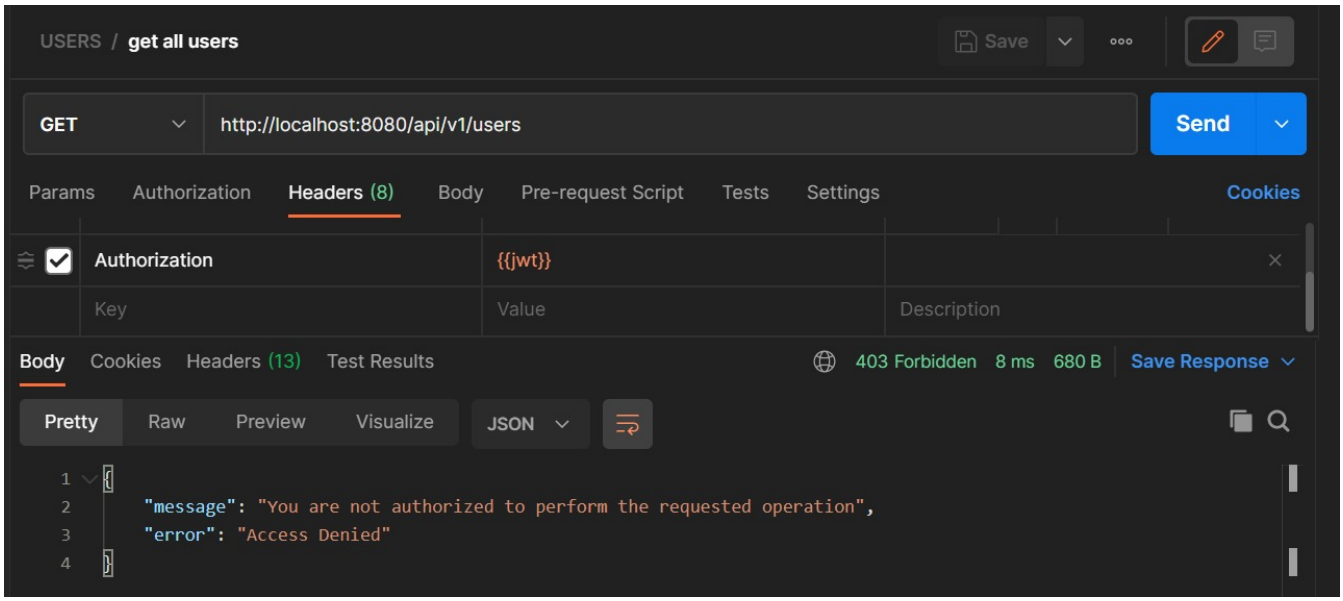


Figure 10 shows an authenticated user trying to access a route he/she doesn't have access to. Only a user with role of an `admin` is allowed to read all users stored in the database.

6.3 Open Web Application Security Project - OWASP