GER'S GARAGE

Felipe Cunha

ABSTRACT

This is a project thought and developed at the very end of the last semester of the Higher Diploma in Science Computing course at CCT College. The students were given a booking system scenario to work on and that's where Garage system was. The idea of project is to create the web system where carries out maintenance for all kinds of small to medium vehicles (i.e. motorbikes, cars, small vans and small buses).Allowing the clients make the book online will help the garage system manager the book for each mechanic has, will help the client/ staff save time given details about the problems before the vehicles arrived the garage.

ACKNOWLEDGEMENTS

I would like to express my heartfelt gratitude to Ken and Aldana, my research advisor, for their invaluable guidance, insightful feedback, and unwavering support throughout the course of this project. Their expertise and encouragement have been instrumental in shaping the direction of my research. Special thanks to classmates for their stimulating discussions and [specific assistance], which significantly contributed to the development of my ideas. I am grateful to CCT college for providing access to the library, laboratory that was essential for the successful completion of this project

TABLE OF CONTENTS

CHAPTER 1 6

I. Project context: 6

II. Why is it a good project? 7

III. Main goal 7

IV. Objectives 7

V. AREAS TO COVER 7

VI. Personal Challenges 8

PLANE 9

Chapter 2 10

Literature Review 10

DATABASE 12

VII. CHAPTER III 12

SYSTEM ANALYSIS & DESIGN 12

FUNCTIONAL REQUIRENMENTS 12

Main requirements: 13

User interface Diagrams: 33

DATA REQUIRENMENTS: 35

Diagrams & Design 36

CONCEPTUAL DESIGN 36

PHYSICAL DESIGN 39

How I decided created all those table: 40

Functional design: 41

CHAPTER IV: IMPLEMENTATION OF THE SYSTEM 44

IMPLEMENTATION OF THE SYSTEM: 47

The main logical of system 54

Book service 54

Print invoice 56

Spring security 58

VIII. CHAPTER V:TESTING & EVALUATION 59
CHAPTER VI: CONCLUSIONS & FURTHER WORK..... 67
FURTHER WORK..... 68
APPENDIX A: CLASS DIAGRAM 69
APPENDIX B: LINK GITRUB 70
REFERENCES 72

CHAPTER 1

INTRODUCTION

I. PROJECT CONTEXT:

Due to Ger has small garage and has no web system to control the book. Ger need to manage the books of car/van/bus/motorbike and make the scheduling for the staff, check how many books he has by day or week. This system will be helpful because at moment the client does not have any book online.

In contact with the customer was made a survey of requirements and needs the systems, the functionalities that will contain the system:

- The system will have tree types of user: ADM,MECHANIC,CLIENT.
- Privileges of ADM user:
  - Able to allocate vehicle's book to each mechanic;
  - Able to see/delete/ all the clients/mechanic/ADM registered on system;
  - Able to see/cancel all the vehicle's booked;
  - Able to add new ADM/MECHANIC user if there is need;
  - Able to view bookings for any particular day OR week;
  - Able to print the invoice when the book is completed.
  - Edit his profile;

- Privileges of MECHANIC user:
  - Able to update the status vehicle's book;
  - Able to add cost the vehicle's book if there is need.
  - Able to see all the vehicle's booked;

- Privileges of CLIENT user:
  - Able to register on system;
  - Able to add vehicle/van/car/bus/motorbike;
  - Able to edit/cancel the book.
  - Able to see all the book has done by himself;
  - Able to see/edit/delete all the vehicle registered by himself on the system.
- All the types of user can edit their profile/make login/make logout.
- When whoever user make the register they have to provide details/Name, phone Number, address, ,email, password.
- Only the mechanic has inform the PPSno.

II. WHY IS IT A GOOD PROJECT?

Through this garage web system it will be possible to keep the user data and make contact when necessary and pass updates on the service through the website or even accessing the customer details making direct contact.
The system will make a biting of the history of the car, this will help a lot to solve current mechanical problems of the vehicle.
With the functionality of scheduling online services will avoid an over load of demand of the mechanics being able to meet the deadline for delivery of the vehicle. With the possibility of checking the status of the service will pass a greater confidence to the customer about the service being provided.
In the future this system can provide reports with average service time, which procedure are being more expensive for the Garage. This can be able to make strategic decisions to improve the business.

III. MAIN GOAL

Develop the web system where the user can make book for his vehicle.

IV. OBJECTIVES

- Research of technologies and review of previous concepts;
- Define design and diagrams of the project;
- DB connection;
- Connect the Front-end with Back-end.;
- Complete final report.

V. AREAS TO COVER

I have to decided to develop the system in Java and the technologies that will be covered are listed below:

- Spring boot;
- Spring security;
- Model-View-Controller (MVC) as architectural pattern;
- Apache Tomcat as Web server tool, to provide a HTTP web server environment;
- Bootstrap framework;
- Java Script – Validation the label and give the message to users.
- HTML;
- Thymeleaf;
- MYsql;
- Password Hashing;
- Programming logic;

- Relational Database (MySQL), to store the data of the program;
- Project planning techniques such as Gantt Charts;

VI. PERSONAL CHALLENGES

The biggest challenge for me was thinking about how I was going to make a front-to-back connection and then tie an object into the database. Front-end is not a technology that I don't have much knowledge of.
So I started researching some framework most used in java programming. So I decided to make use of Spring Boot. My initial plan was to create a small project to get to know me the framework. After that I started the Garage project. There was a lot of difficulty in issues of configuration of the environment, entered how the dependencies work, it took a long time to be able to generate the tables in the bank due to the version of java. Every unexpected mistake I could count on colleagues and chats in the internet of people who went through error that I went through.

PLANE

It was used Gantt Charts for manager the task by the time and see how long each task took for developed. The tool provide the deadline about each task and is easy update.

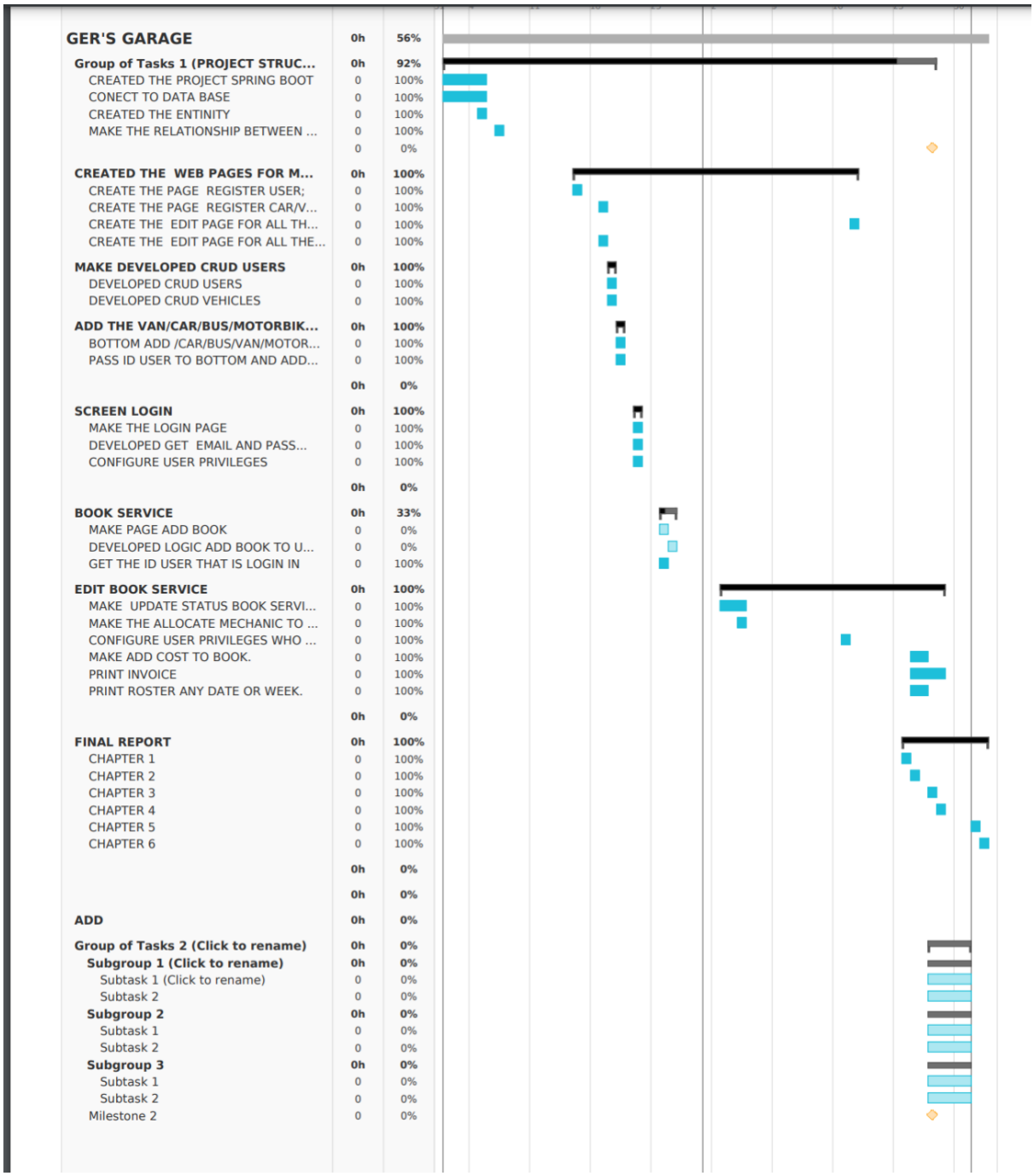


Illustration 1:plan

CHAPTER 2

LITERATURE REVIEW

The aim of this chapter is to present all academic research carried out throughout the project cycle. It is important that learners produce research that defends their justifications for choosing one from of technology or software over another, and other sources of information that have helped inform the individuals thinking, planning and delivery of the project.

- Java maven project;
- MVC Spring boot;
- Data base.

I have done a research and I have done networking with who work as developer in Java. Accordingly with I get spring boot is framework from Java really helpful building the project. Could be hard in the beginner who never has used before but will save time in the future because many configuration It is already done by Spring boot for example the server. By the way I have decide used Java because I have more skills and one of the language that was taught during the course. Due to deadline really short I thoughted was better chose some technology that I have more skills.

In the beginner I have to created small project Spring Boot just for to know how structure od project works.

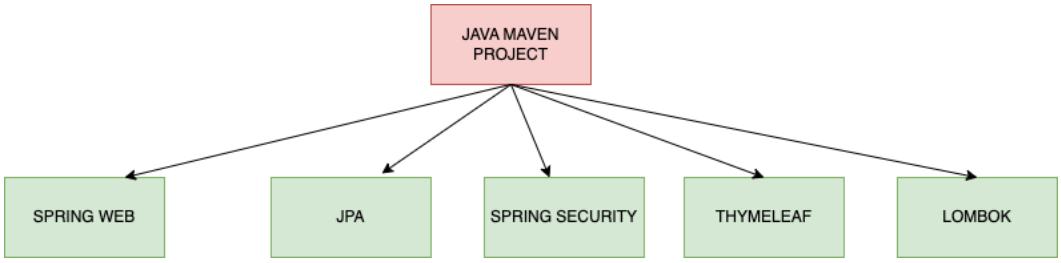


Illustration 2:Strucure of Spring Boot

- Spring WEB: belong of Spring framework that It is the platforms Java made for get easy the develop of applications. The Spring web give components for to build apps web in Java and give many library help with develop the application:

    - Spring Web includes the **Spring MVC** module, which is an implementation of the Model-View-Controller design pattern. It helps in separating concerns in web applications by dividing them into three

main components: Model (data and business logic), View (user interface), and Controller (handles user requests and coordinates the flow).

- **Web Controllers**: Spring provides annotations such as @Controller and @RestController, which simplify the process of creating request-handling controllers. These controllers can handle different types of requests (GET, POST, PUT, DELETE, etc.) and map them to corresponding methods.
- View Resolvers: Spring Web supports various view resolvers, allowing you to easily integrate different view technologies, such as JSP, Thymeleaf, and FreeMarker. These view resolvers resolve the logical view names returned from the controllers to actual view templates.
- **Data Binding**: Spring Web facilitates data binding between HTML form fields and Java objects. It automatically maps incoming request parameters to Java object properties and vice versa, simplifying form handling.

- JPA is a Java specification that provides a standard way to interact with relational databases using Object-Relational Mapping (ORM). It allows developers to map Java objects (entities) to database tables and provides an abstraction layer to perform CRUD (Create, Read, Update, Delete) operations on the database without writing explicit SQL queries. JPA implementations such as Hibernate, EclipseLink, and OpenJPA provide the concrete implementations of the JPA specification.
  Key concepts in JPA include:
  - **Entities**: Java classes annotated with @Entity that represent database tables.
  - **Persistence Context:** The first-level cache of JPA that manages the lifecycle of entities and tracks changes.
  - **Entity Manager**: An interface to interact with the persistence context and perform database operations.
  - **JPQL (Java Persistence Query Language):** A query language similar to SQL, but it operates on JPA entities rather than database tables.

How the JPA works with Spring boot?

When using JPA with Spring Boot, you can leverage the built-in support for JPA entities and data access. By adding the necessary dependencies, Spring Boot automatically configures the data source, entity manager, and transaction management, making it easy to perform database operations using JPA.

To use JPA with Spring Boot, you typically:

- Define JPA entities using @Entity annotations on Java classes that means each class that will have annotations @Entity will become tables on database.
- Create a data source configuration in the application.properties or application.yml file for make the connection java with data base.

- Use Spring Data JPA repositories or create custom data access components to interact with the database like find all the client by ID.
- Let Spring Boot handle the rest, including managing the entity manager, transaction handling, and connection to the database.

  Source: https://www.michellibrito.com/

DATABASE

I have decide to user MYsql due to had to work with it for 4 year and It easy to use the language SQL for manipulate the data and supported small project that It was necessary for Ger's garage.

Below some concept about MYsql:

- MySQL is an RDBMS, which means it organizes data into tables with rows and columns, allowing for efficient data retrieval and management.
- MySQL is an open-source database, making it freely available for use without any licensing costs. This makes it a cost-effective option, especially for projects with budget constraints.
- MySQL has a straightforward setup and configuration process. Its SQL-based query language is easy to learn and use, making it accessible to developers with varying levels of expertise.
- MySQL can be integrated with many programming languages and platforms, such as PHP, Python, Java, Ruby, and more, allowing developers to work with their preferred technologies.
- MySQL provides various security mechanisms, including user authentication, data encryption, and access control, ensuring data integrity and protection against unauthorized access.

  Source: https://www.w3schools.com/MySQL/default.asp

VII. CHAPTER III

SYSTEM ANALYSIS & DESIGN

Ger's system basic will make book online for car/vans/bus/motorbike where the system will manager the book service that means allocate the book service to each staff, print the invoice when the service is completed. This chapter will show up functional requirements and functional design. In addition, it will provide the Diagrams that support both back-end and front-end.

FUNCTIONAL REQUIRENMENTS

The application involves functional requirements for mechanic/admin/clients users:

MAIN REQUIREMENTS:

- The CLIENT will abler register on the system for book his service;
- The ADMIN will able to allocate the book to Mechanic and print the invoice;
- The MECHANIC will able to update the status of service and make notes and add extra cost if there is need.;
- All the users are able to edit their profile;
- All the user are able make login and log out.

ACCESS TO THE SYSTEM:

When the system run the ADMIN user is created automatic and he can make the login.
The first the page on the system is login page where the client will have two option, make the login or register on the system.
Client have to provide those information for register;

- First Name;
- Last name;
- Phone number;
- Address;
- Email;
- Password;

If the client has already account created just have to type email and password and make login.

Mechanic/ADMIN user are able just for make the login. The ADMIN user will have option for register MECHANIC USER/ADM

Log-in: MECHANIC/ADMIN/CLIENT ABLE MAKE LOGIN.

- Email;
- Password

Use-case access the system:

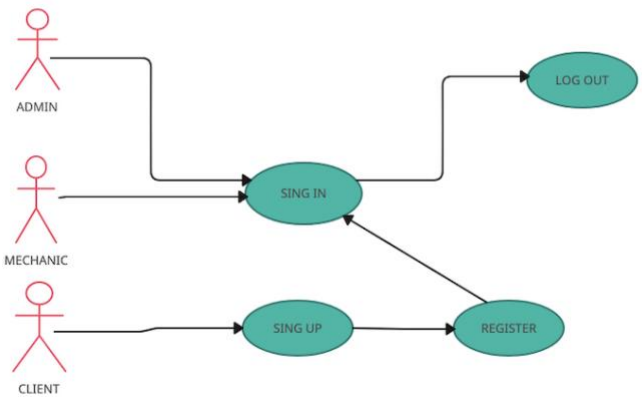


Illustration 3:acess the system

Register vehicles:

When the client make login on system will have all those option:

- CAR
  - New Car;
  - List Cars;
- VAN
  - New Van;
  - List Vans;
- BUS
  - New Bus;
  - List Buses;
- Motorbike
  - New Motorbike;
  - List Motorbikes;
- Book service;
  - Book Car;
  - Book Van;
  - Book Bus;
  - Book Motorbike;

When the user want register whatever vehicles have to provide details:

- Makes;
- Types;
- Engine types;
- Capacity passenger;
- Cylinder Capacity(This label just show up for register Motorbike)
- Numbers of doors(this label does not show up for register motorbike);
- Year;
- Plate;
- Color;

Use-case register vehicles:

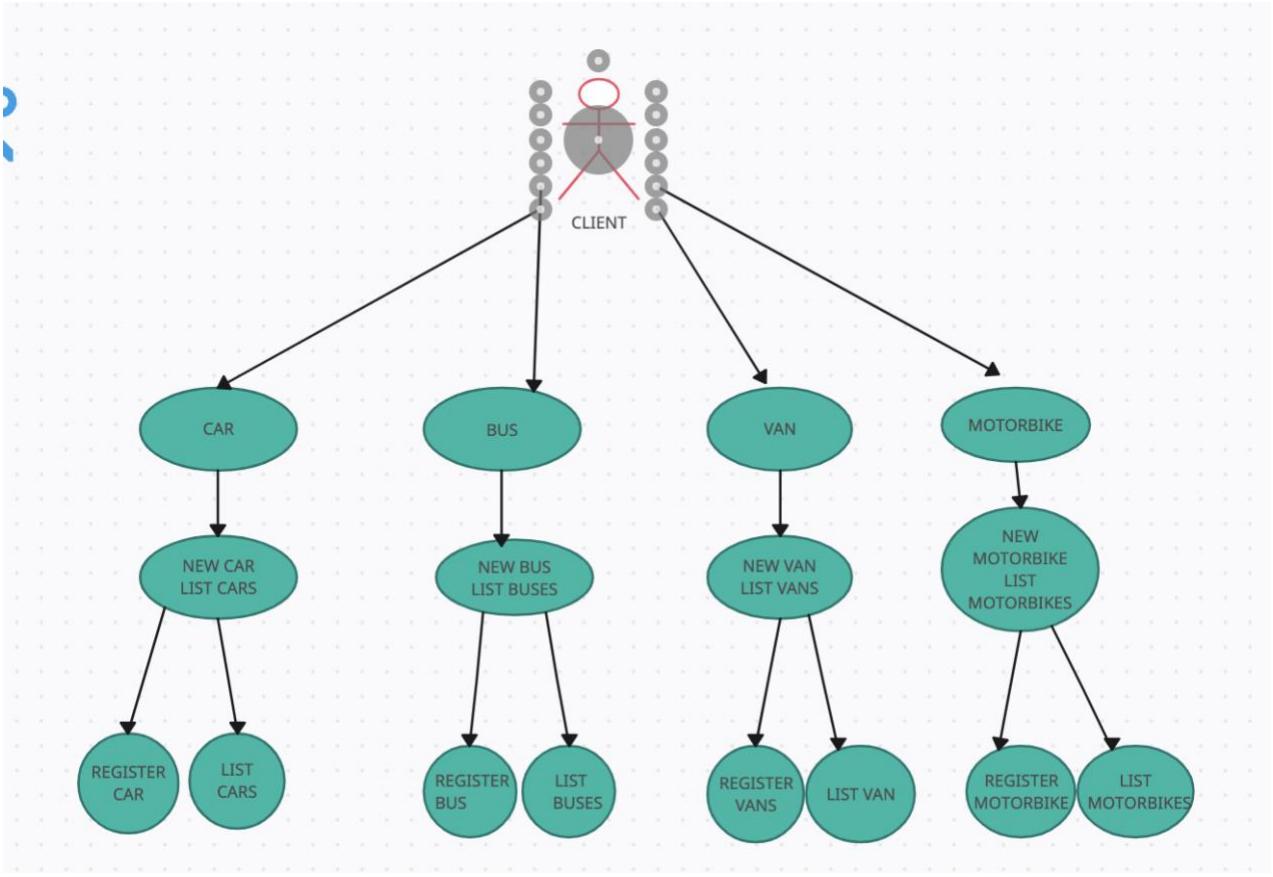


Illustration 5: register vehicles

MECHANIC/ADIMIN/MECHANCI are able to edit their profile:

Use-case edit profile:

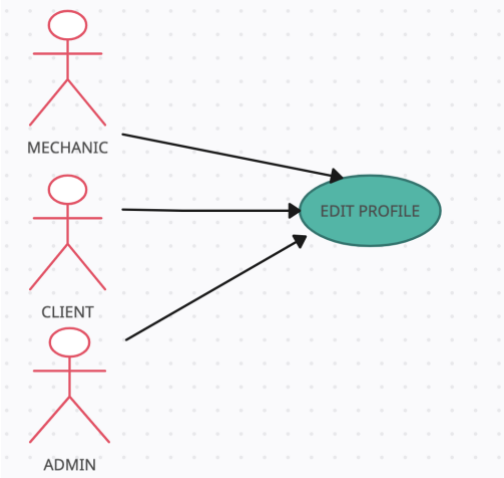


Illustration 6: edit profile

Edit details vehicles:

Clients are able to edit/delete/list the details about their vehicles that It is registered on system:

Use-case edit/delete/list vehicles:

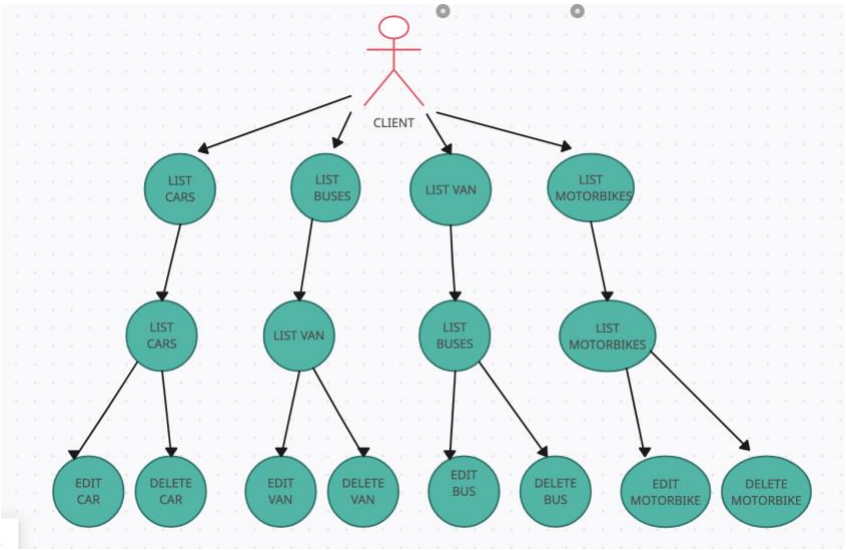


Illustration 7: Use-case edit/delete/list vehicles

Book a service vehicles:

The client will able to book Service for his vehicles. It will show up to client those option:

- Book Service:
  - Book car
  - Book van;
  - Book Bus;
  - Book motorbike;

The client gave to provide all those details for book the service:

- Type of service;
- Date of book;
- Time of book;
- Details about the problems.

Use-case register Book:

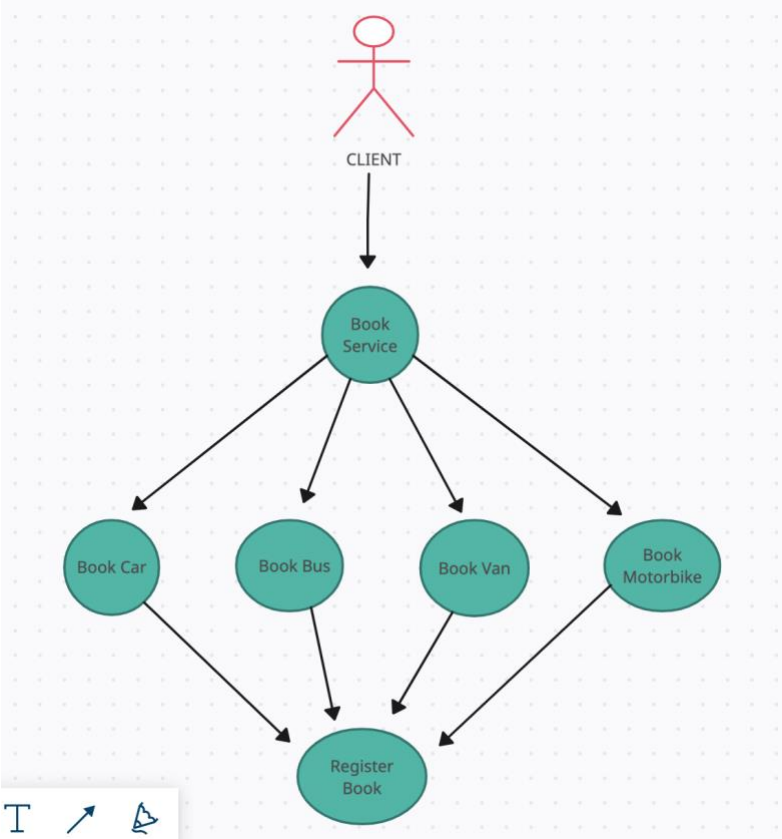


Illustration 8: Use-case register book

Client able to see/edit/cancel all the book that they have.

Use-Case see/edit/Cancel books:

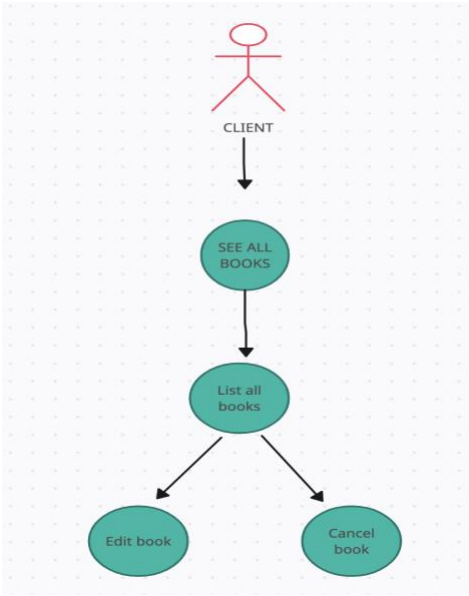


Illustration 10: Use-case see/edit/Cancel books

ADMIN USER:

Admin user will have option:

- Client;
  - List all the clients registered on the system;
- Admin;
  - List all the Admin registered on the system;
  - Add new admin user;
- Mechanic;
  - List all the mechanics registered on the system;
  - Add new mechanic user;

Use-Case side-menu of admin:

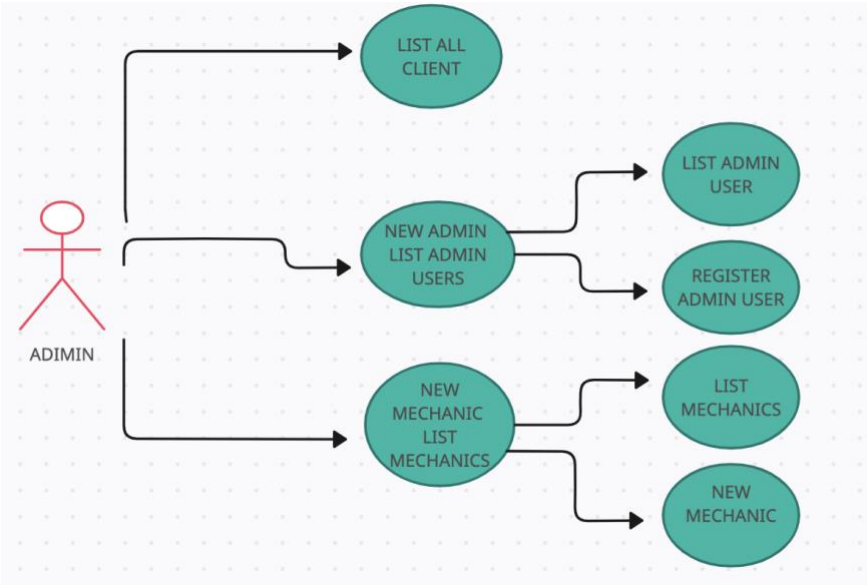


Illustration 11: Use-Case side-menu of admin

Menu book's vehicles on the system for ADMIN USER:

The admin user will have for option for check the books by the type of vehicles van/car/bus/motorbike where each option will show up all the books that was done on the system and will be possible allocate the book to mechanic.

- Check the car's books;
  - Allocate book to mechanic;
- Check the van's books;
  - Allocate book to mechanic;
- Check the bus's books;
  - Allocate book to mechanic;
- Check the motorbike's books;
  - Allocate book to mechanic.

Use-case allocate book:

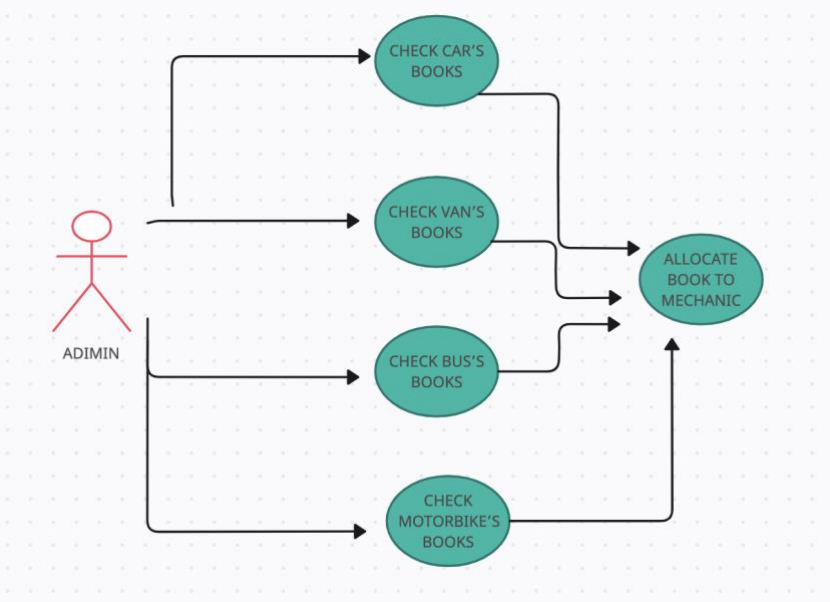


Illustration 12: Use-case allocate book

- **Generate invoice;**

Use-Case side-menu generate invoice:

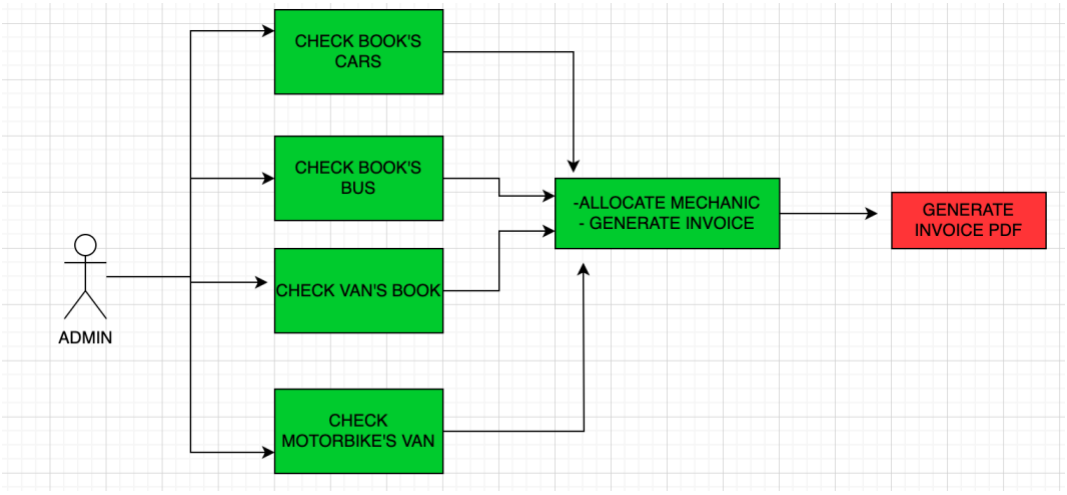


Illustration 13: Use-Case side-menu generate invoice

Menu book's vehicles on the system for MECHANIC user:

The mechanic user will have for option for check the books by the type of vehicles: van/car/bus/motorbike where each option will show up all the books that was done on the system and will be possible update the status of book and add extra cost and give some details about what was done in vehicles.

- Check the car's books;
  - Update status of book;
- Check the bus's books;
  - Update status of book;
- Check the van's books;
  - Update status of book;
- Check the motorbike's books;
  - Update status of book;

Use-case Mechanic's menu:

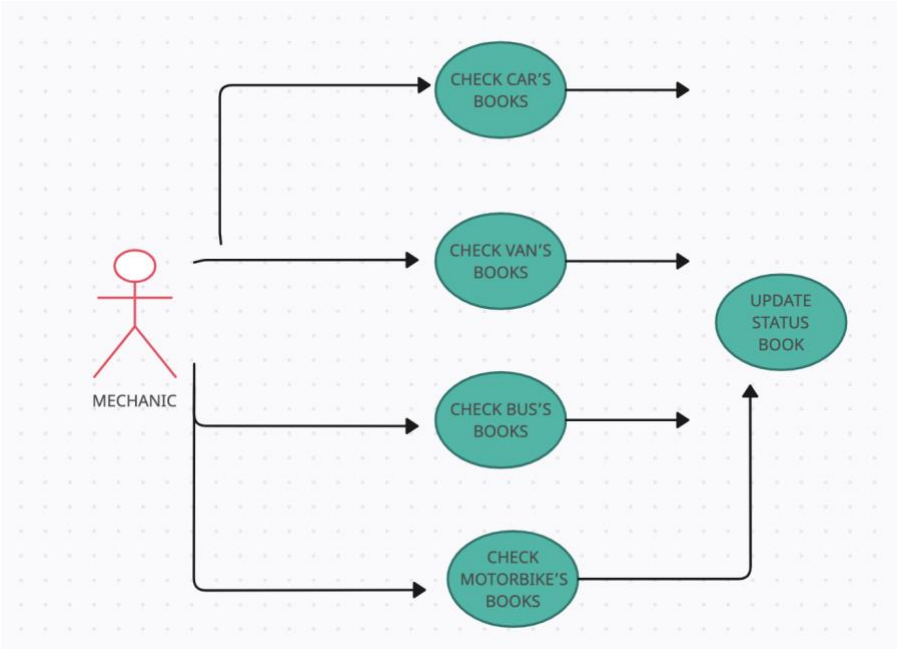


Illustration 14: Use-case Mechanic's menu

Wireframes:

The following Wireframes represents the skeletal framework of the website, taking into consideration the above requirements.

Login page: will be available for the user make sing in or sing up if the one of the user will not have account, if the user put something different that format of email will show up error message:

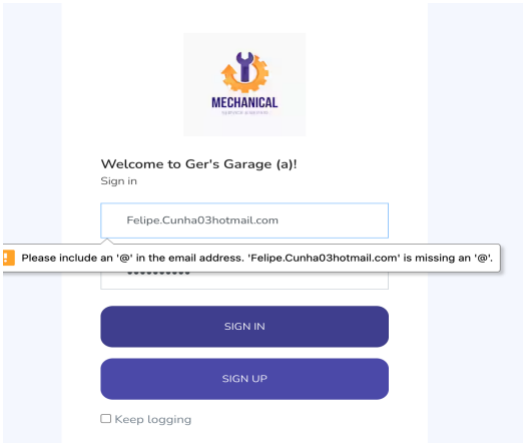


Illustration 15: login page

Register client: If the client does not have account will press bottom sing up and fill the form , all the labels are mandatory, if the user put some number instead letter for example will show up the message, All the label from all the page has validation as well and make sure the user will type the right data.

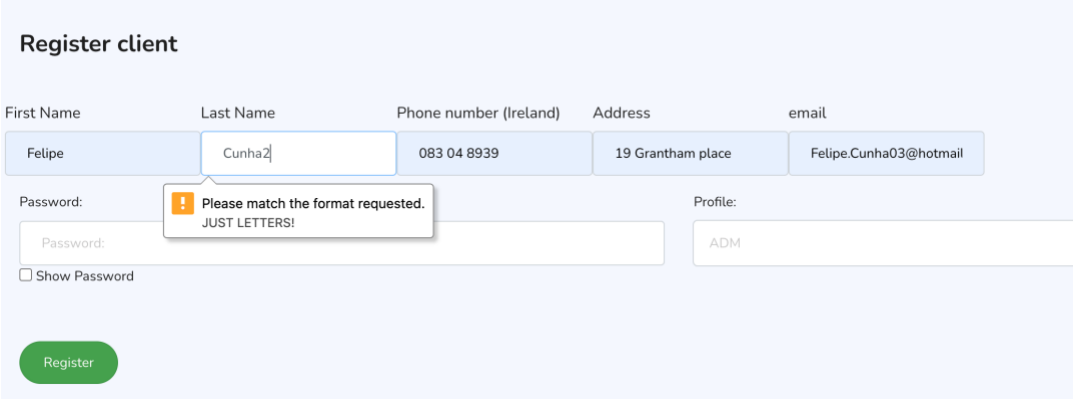


Illustration 16: registerClient

Once registered, the Client will access :

- Edit his profile, this option show up all users client/admin/mechanic:

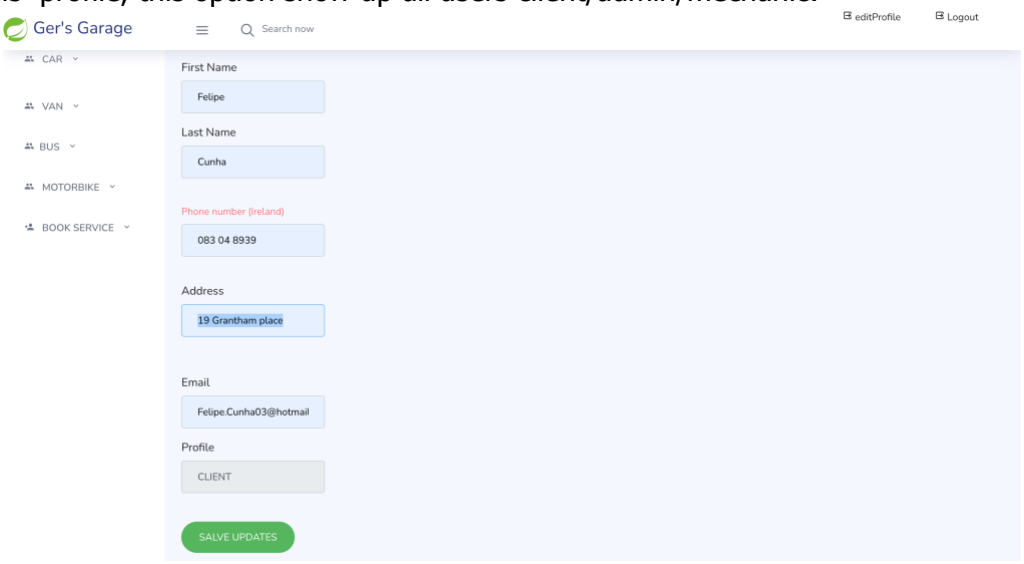


Illustration 17: edit profile

- Add vehicles /list/edit all the types:

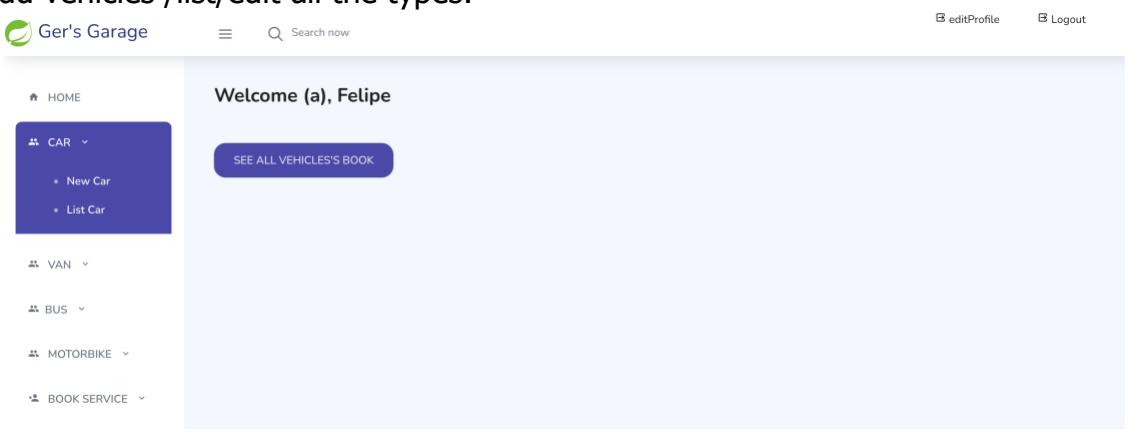


Illustration 18: add vehicles list

- Register car for example:
  There is one validation for make sure the user type the right plate in format of Ireland:

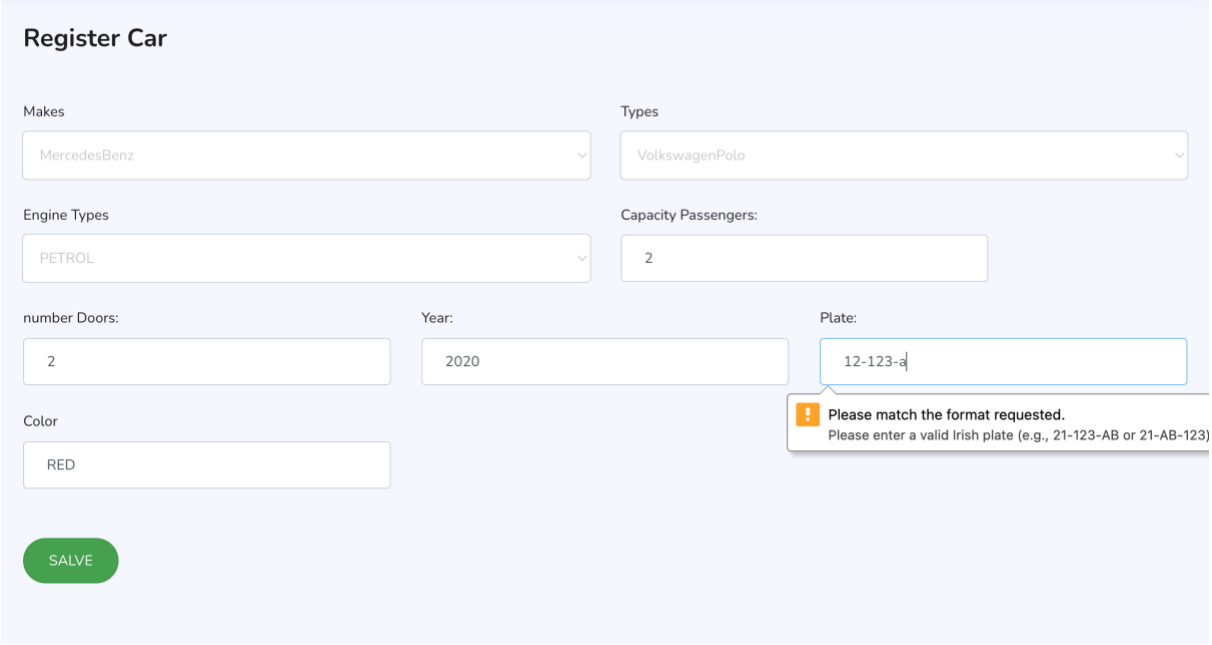


Illustration 18: register car

- List car for example:

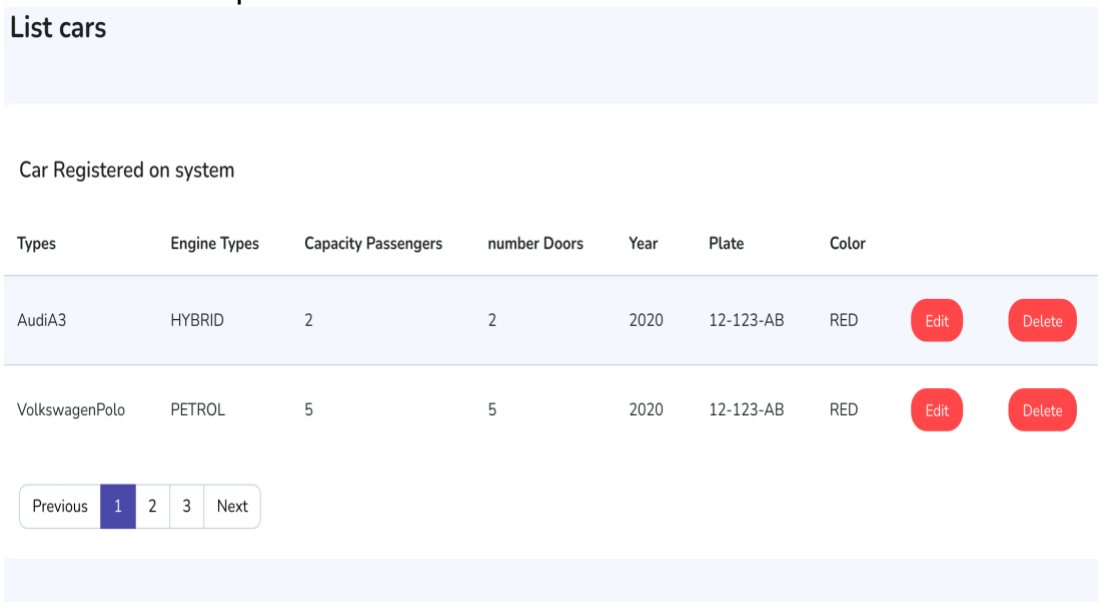


Illustration 19: list of car

❖ Edit car evidence that the all the labels are mandatory to fill:

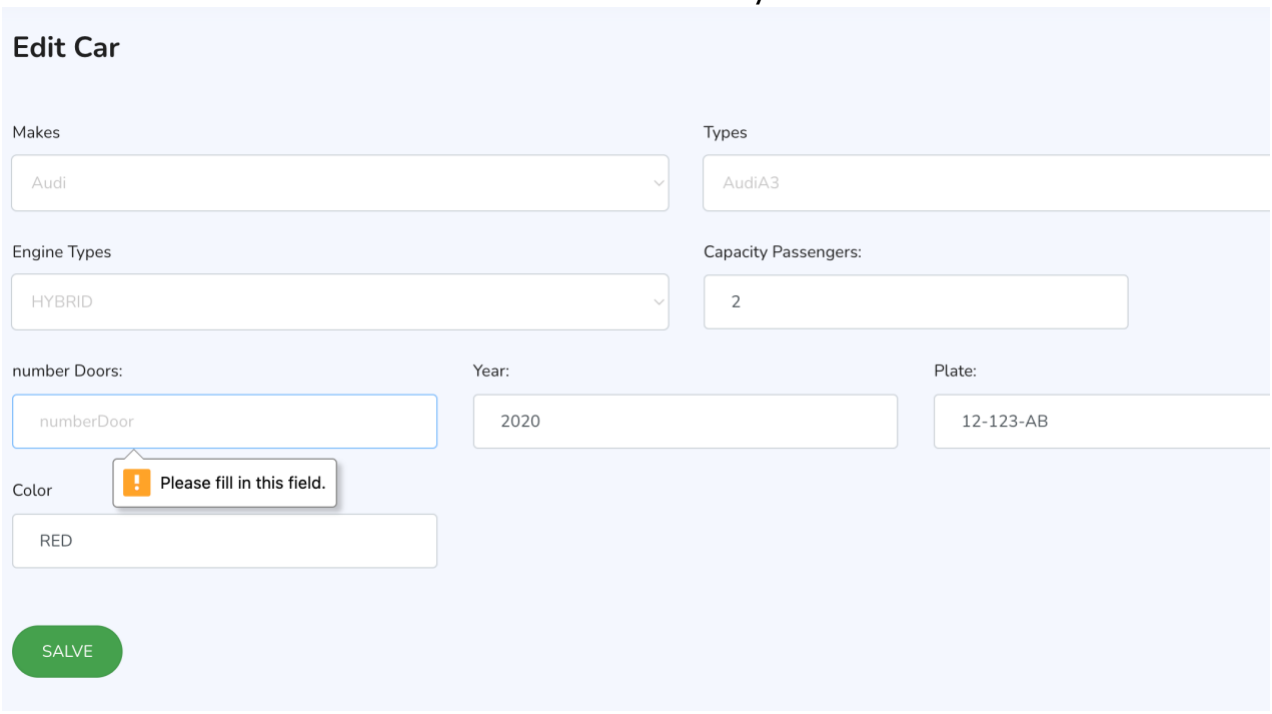


Illustration 19: edit car

- Book the service, on the side-menu the client will have access book the service for Car/Van/Bus/Motorbike:

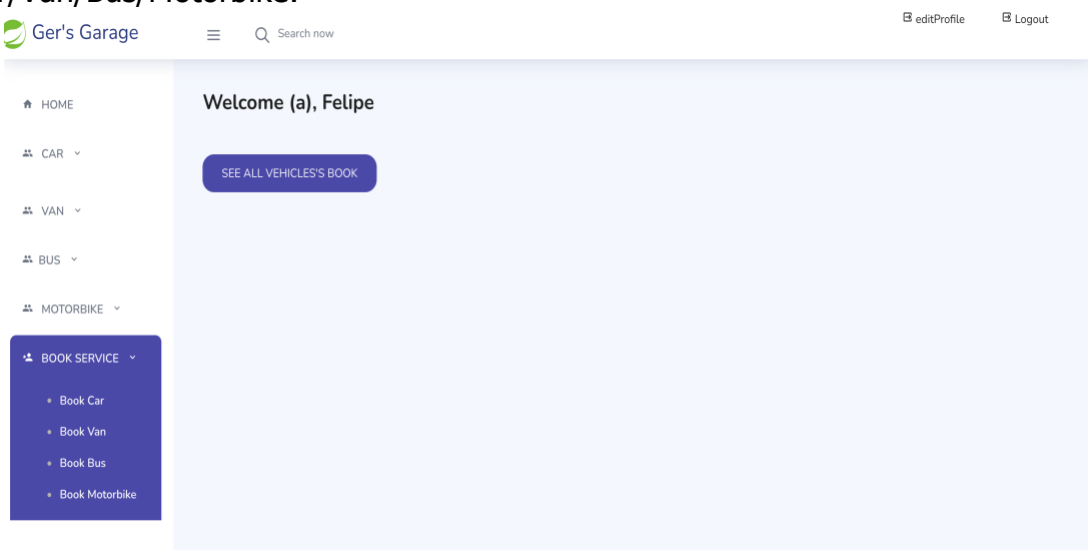


Illustration 20: Book service

- List of cars registered on the system for make the Book Service:

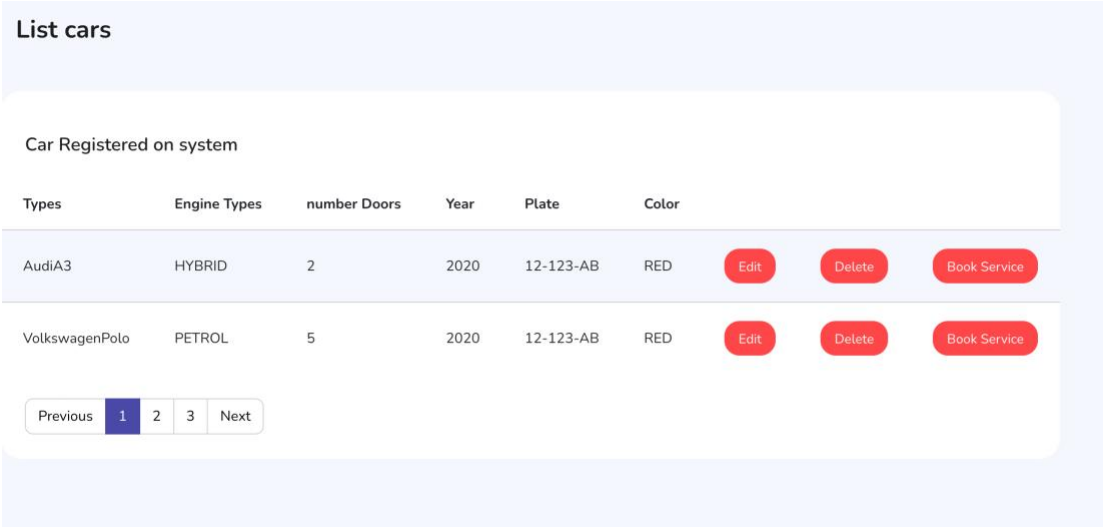


Illustration 20: list the cars for make book

- Register book car the user is not allow book the service Sunday and Book the service just between 9:00 to 17:00 Monday to Saturday.

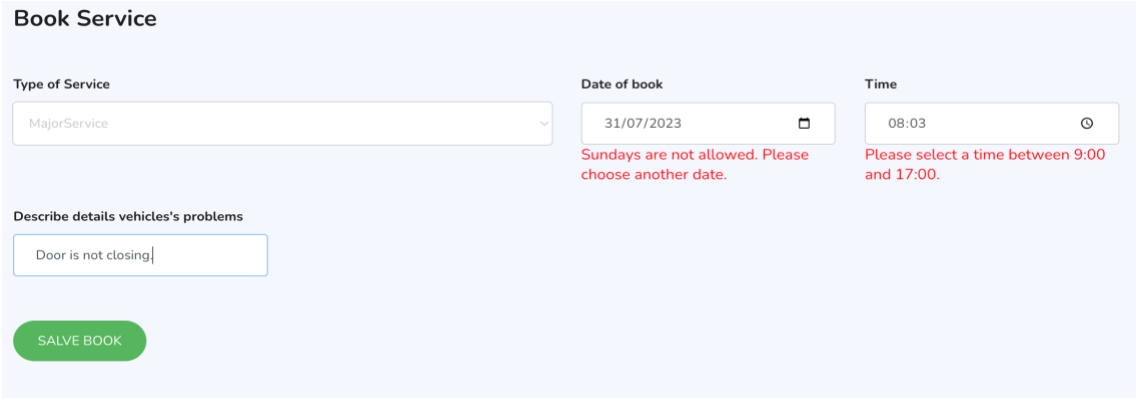
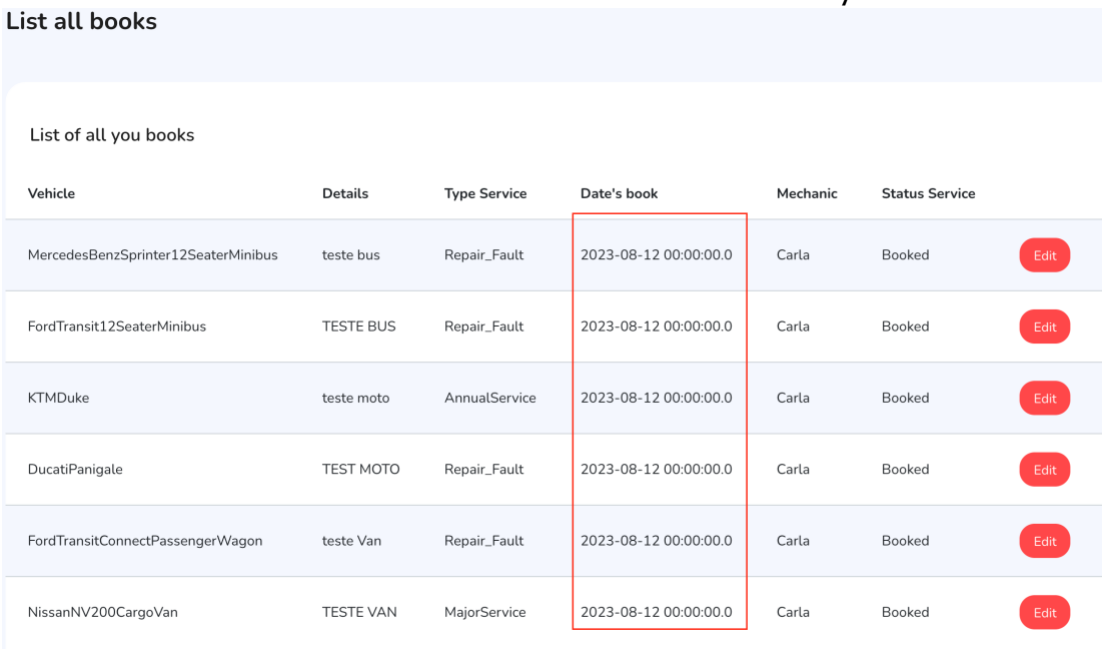


Illustration 21: register book

When there is no book available for the day:



```
Hibernate: select motorbike0_.id as id1_9_0_, motorbike0_.col
Hibernate: select motorbike0_.id as id1_9_0_, motorbike0_.col
Hibernate: select car0_.id as id1_6_, car0_.color as color2_6
Hibernate: select client0_.id_user as id_user1_7_0_, client0_
2023-08-12 18:25:19.914  WARN 16202 --- [nio-8080-exec-5] .w.
Hibernate: select car0_.id as id1_6_, car0_.color as color2_6
Hibernate: select client0_.id_user as id_user1_7_0_, client0_
Hibernate: select van0_.id as id1_10_, van0_.color as color2_
Hibernate: select client0_.id_user as id_user1_7_0_, client0_
Hibernate: select count(bookcar0_.id) as col_0_0_ from book_c
Hibernate: select count(bookvan0_.id) as col_0_0_ from book_v
Hibernate: select count(bookbus0_.id) as col_0_0_ from book_b
Hibernate: select count(bookmotorb0_.id) as col_0_0_ from boo
There are no available booking slots for this day
```

- See all the book that It was done:

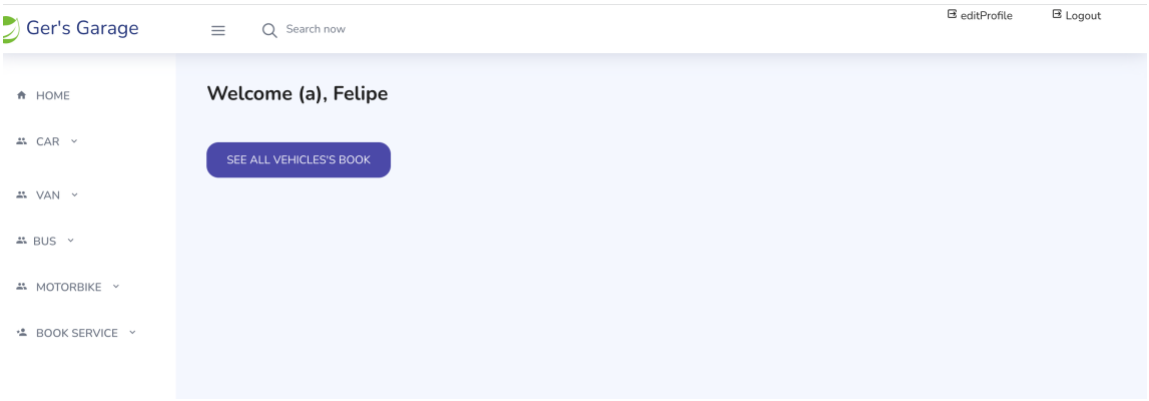


Illustration 22: list of books

- List all the books:

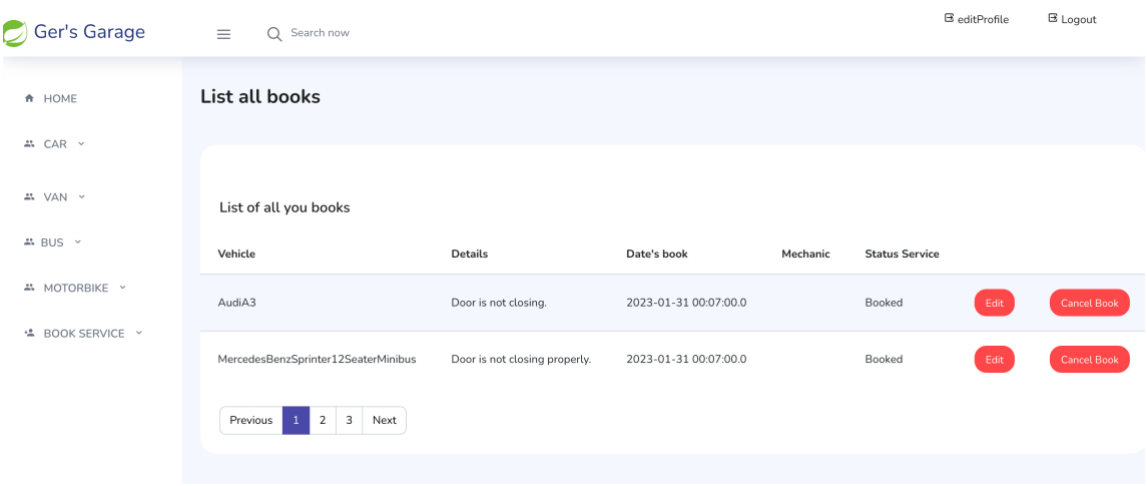


Illustration 23: list books

- **Edit book:**

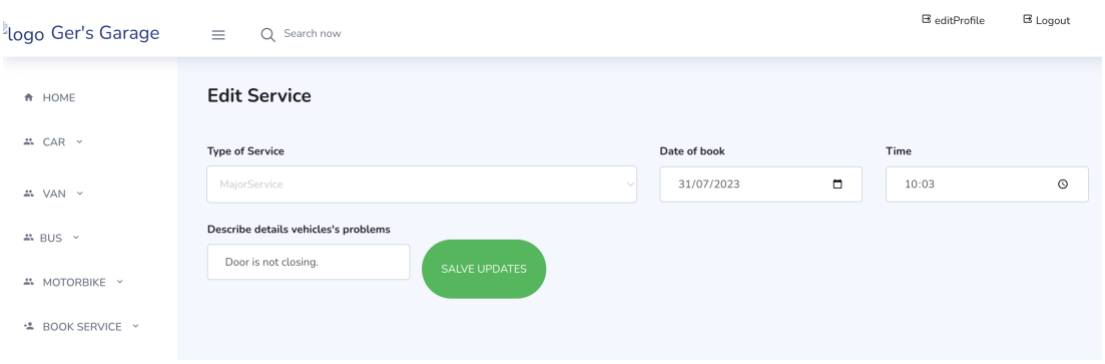


Illustration 24: edit book

- Admin user menu:
  - The ADMIN user will have permission to see all the books by type:

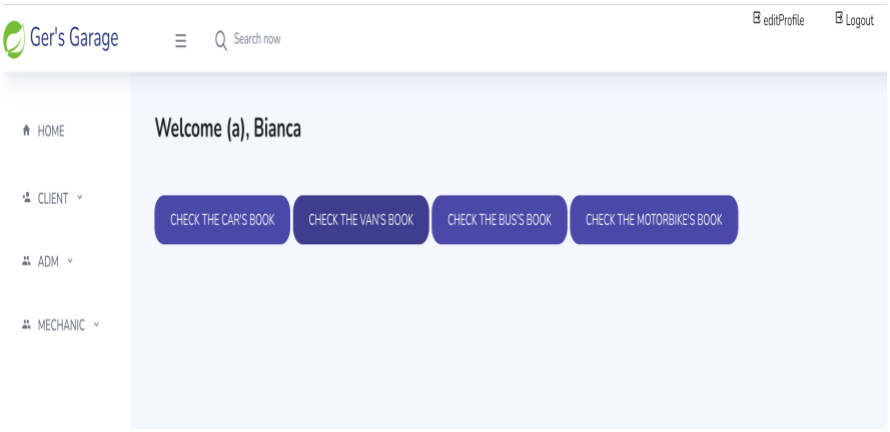


Illustration 25: list books admin user

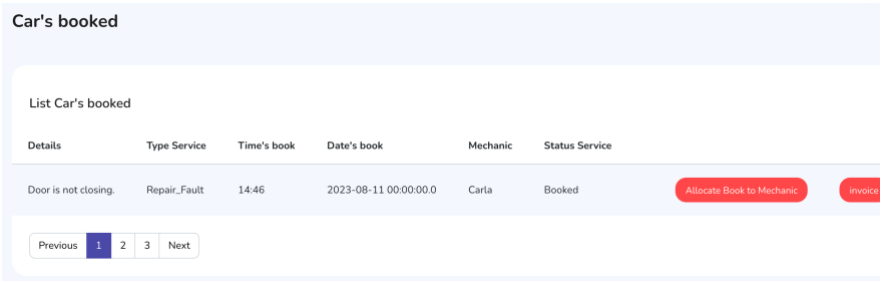


Illustration 26: list books admin user

- The ADMIN user will have permission to allocate the book to Mechanic:

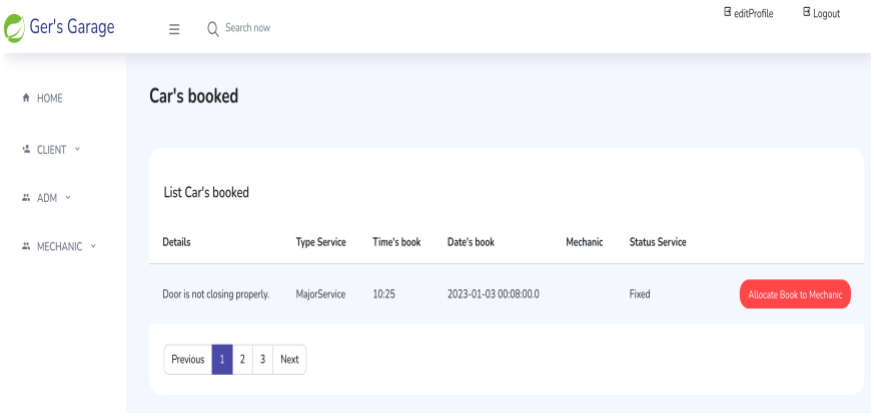
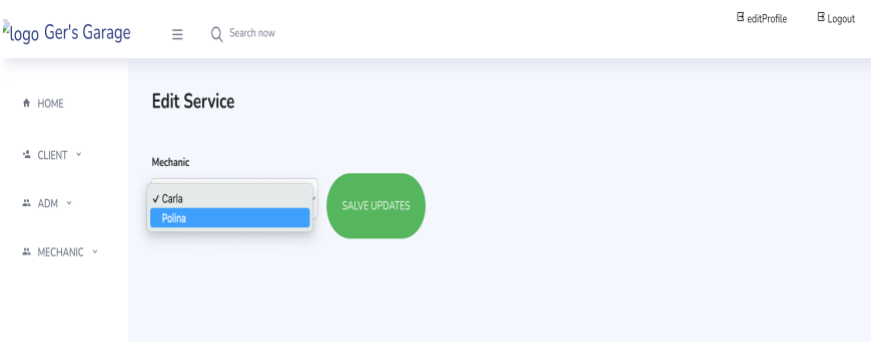
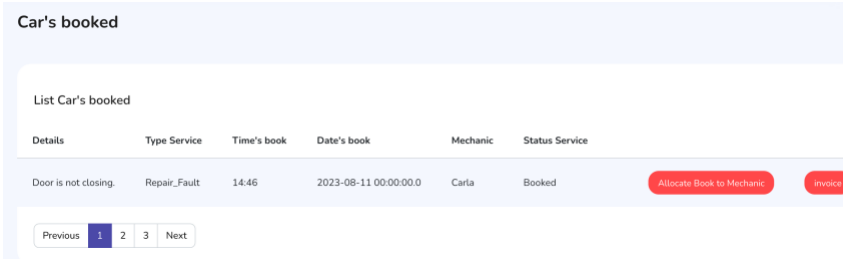


Illustration 27: allocate book

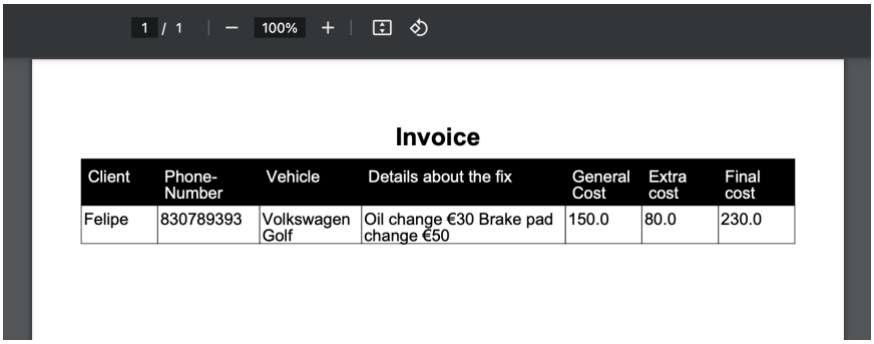


Ilustration 28: allocate book

- The ADMIN user will have permission to generate invoice:

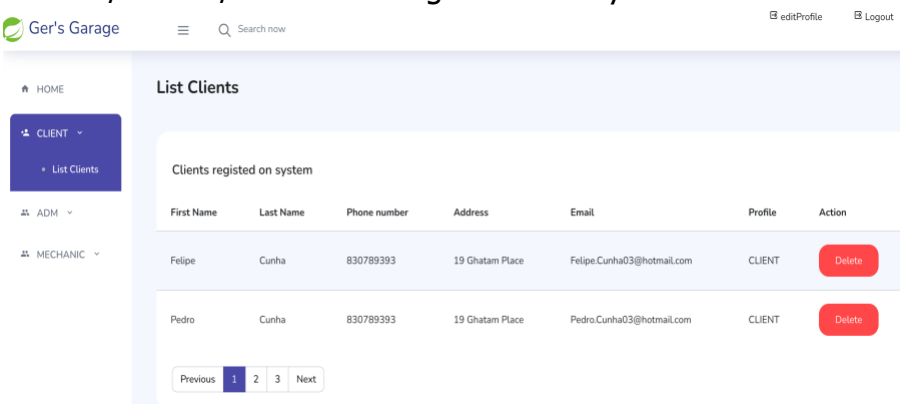


Ilustration 29: generate invoice



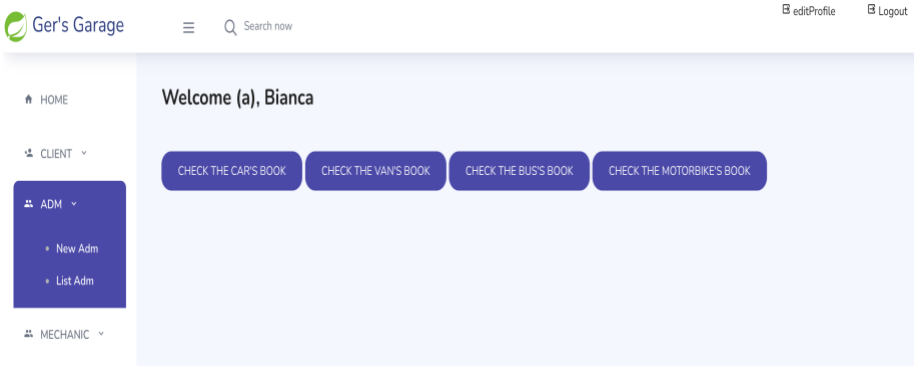
Ilustration 30: invoice

- The ADMIN user will have permission list/delete all the CLIENT/ADMIN/MECHANIC registered on system:



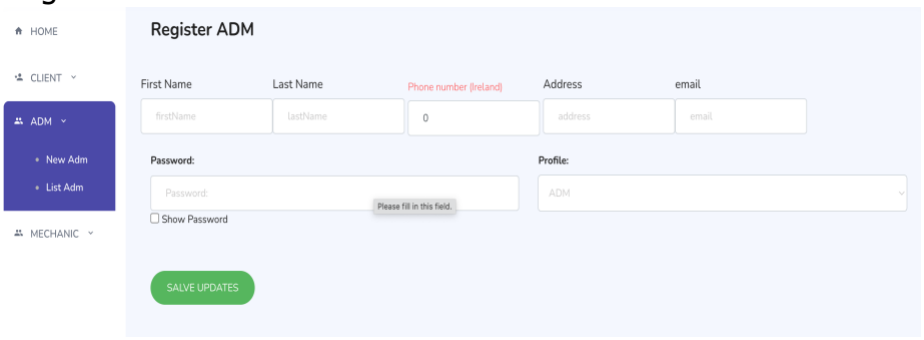
Ilustration 32: delete client/admin/mechanic

- The ADMIN user will have permission edit/delete/register the ADMIN/MECHANIC registered on system:



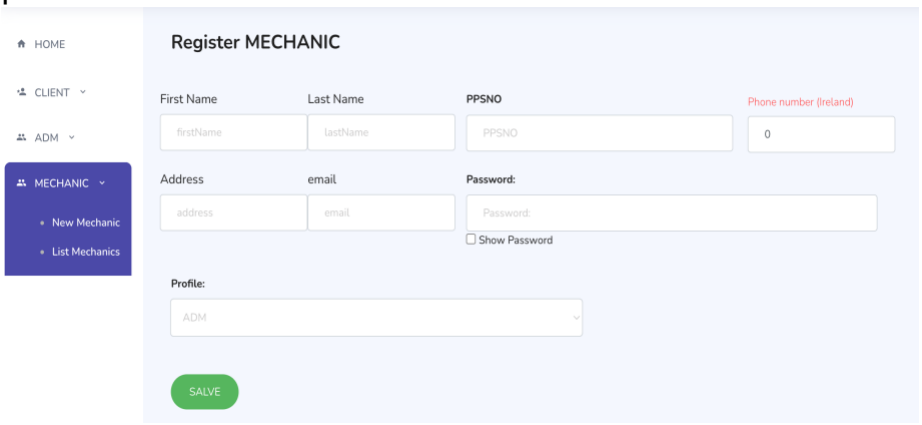
Ilustration 33: edit /delete/register users

Register ADMIN user:



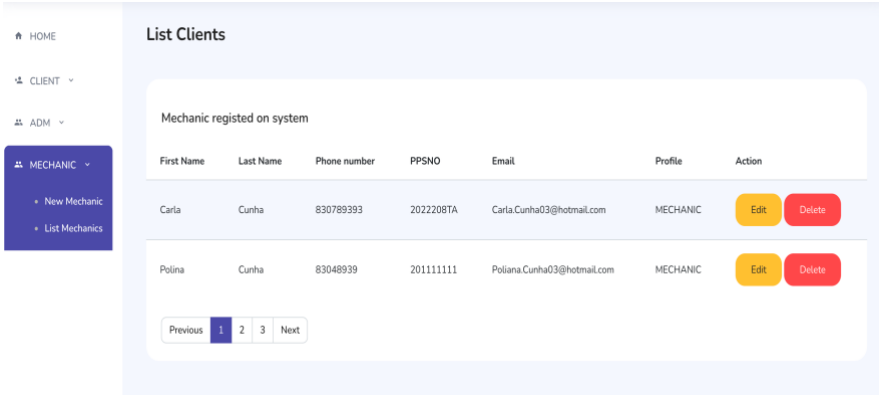
Ilustration 34: register admin

OBS- When the admin user register the mechanic user he has to provide PPSNO:



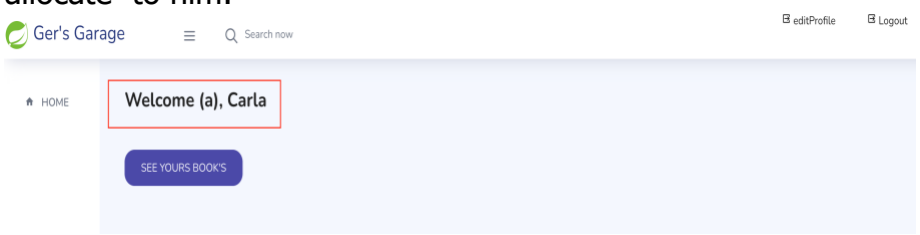
Ilustration 35: register admin

Edit/delete mechanic/Admin user:

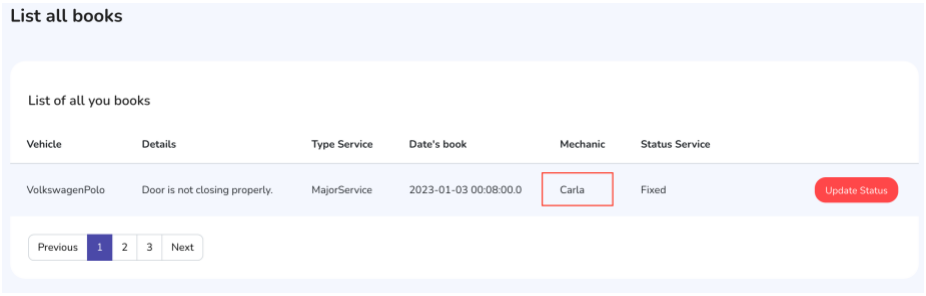


Ilustration 36: register admin

- The mechanic will have permission to add any extra cost if there is need on the book:
  - List all the books allocate for the mechanic that is login in the system. That means the mechanic can see all the books that only allocate to him:

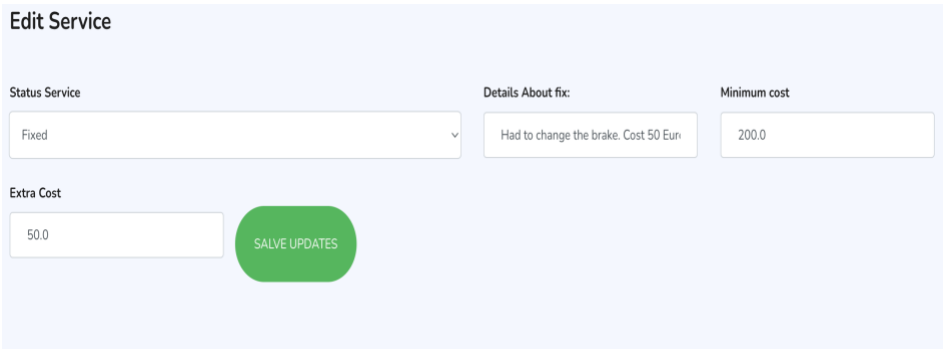


Ilustration 37: mechanic menu



Ilustration 38: mechanic menu

- Just the mechanic will able to update the status and add any extra cost for book that It is booked for him:



Ilustration 39: edit book by mechanic user

USER INTERFACE DIAGRAMS:

Bootstrap is a popular open-source front-end framework used for building responsive and visually appealing websites and web applications. It was developed by Twitter and is now maintained by the Bootstrap community. Bootstrap makes it easier for developers to create consistent and responsive user interfaces by providing a set of pre-designed HTML, CSS, and JavaScript components.

- Database design:
  Each functionality is interact with DB for storge the information.

  Access to system:

  - Register user: storage the data provide of user;
  - Log in: read the information provide of user from DB.

Register/List/edit/delete Admin/Client/Mechanic user:

- All those functionality is done by read the data from DB that was storage when the user provided the data.

Book service:

- Storage the service from de data that the user provide and get the user/vehicles that already storage from DB.
- Allocate mechanic to service from the list of mechanic from DB.
- Update the status and add any extra cost to book service from DB and update those information in DB.

Check book service:

- Admin user can read all the book that was done from any all the client.
- Client user can read all the books that was done by himself;
- Mechanic can read all the book that was allocated just for him.

DATA REQUIRENMENTS:

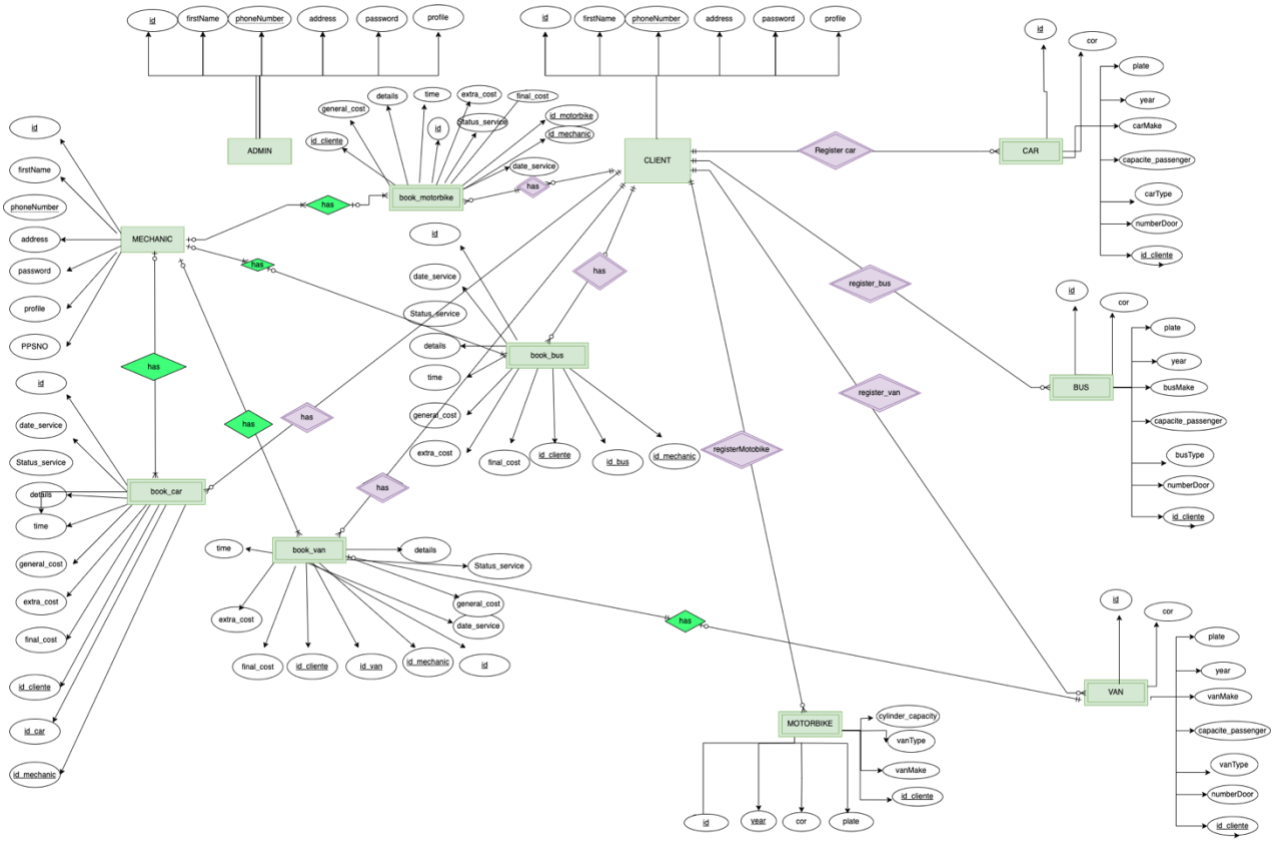
Explain how will work:

ADMIM	Can allocate the book each mechanic Can print the roster by any particular date List all the books. ADD/EDIT/DELETE mechanic/Admin users List all the client register on the system PRINT INVOICE ANY BOOK SERVICE
CLIENT	Can book many service Can cancel their book Can edit some details about book. Can add/list/delete many veichles Can edit his profile
MECHANIC	Can edit his profile Update the status of book that only It is booked for himself List the books any type viechles but just book that It is booked for himself. Can update the status of book and add extra cust if there is need.
BOOK	Mechanic can has 2 book per day. Ger's garage can have 6 book per day max. Client is not allow make book Sunday. Client can just make the book from 9:00 to 17:00
VEHICLES	Each veihicles belong just one user Clinent can edit/list/delete their vehicles

DIAGRAMS & DESIGN

CONCEPTUAL DESIGN

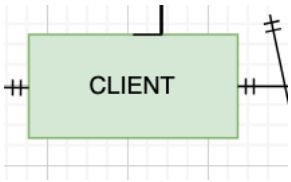
The following diagram is part of the early design process of the Database, which shows the interactions between the entities previously described, based on the data requirements.



Ilustration 40: conceptual design

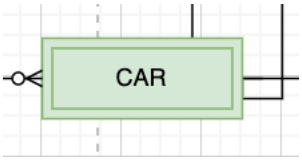
LOGICAL DESIGN

Client/Mechanic/ADM is strong entity because can exist without any book service, means the user can make register on the system and do not make the book.



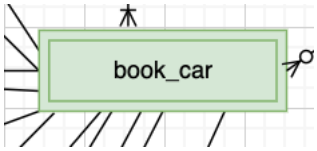
Ilustration 41: entity client

Car is week entity because for add new car have to exist one client registered on system.



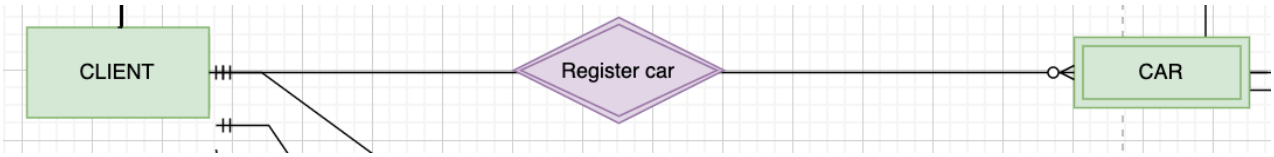
Ilustration 42: entity car

Book Car/Book Van/Book Bus/Book Motorbike are week entity because if the client wants make a book for his car depends of two another entity CLIENT/CAR:



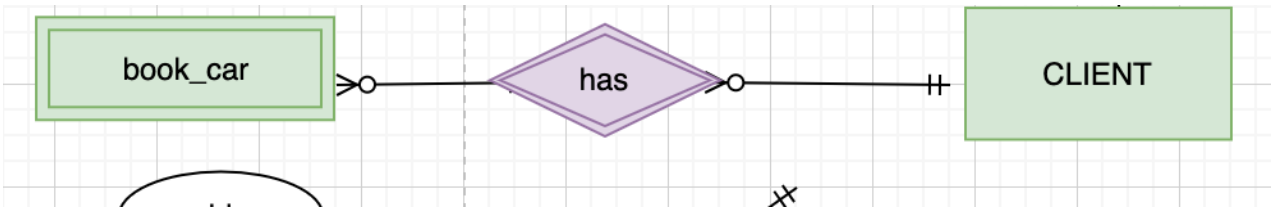
Ilustration 43: entity book car

Relationship between **client** and **car/bus/van/motorbike** are ONE TO MANY because the client can have many cars registered on the system. This relationship is week because for register a car on the system the client have exist.



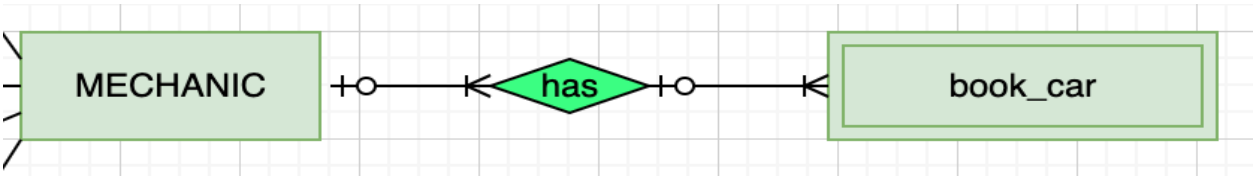
Ilustration 44: relationship between car and client

Relationship between **client** and **bookCar/bookVan/bookBus/BookMotorbike** are ONE TO MANY because one client can have many books on the system. This relationship is week because if the client wants make a book have to exist client and car registered on the system.



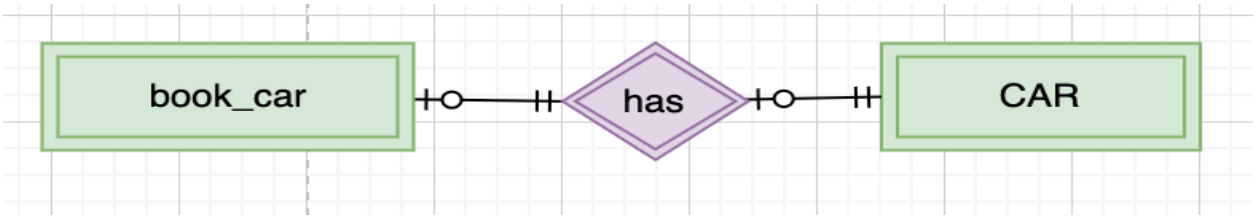
Ilustration 45: relationship between book and client

Relationship between MECHANIC and bookCar is ONE TO MANY because the mechanic can have two book per day. It is strong relationship because the client can make a book and the allocate of the service will happen after the book done.



Ilustration 46: relationship between book and mechanic

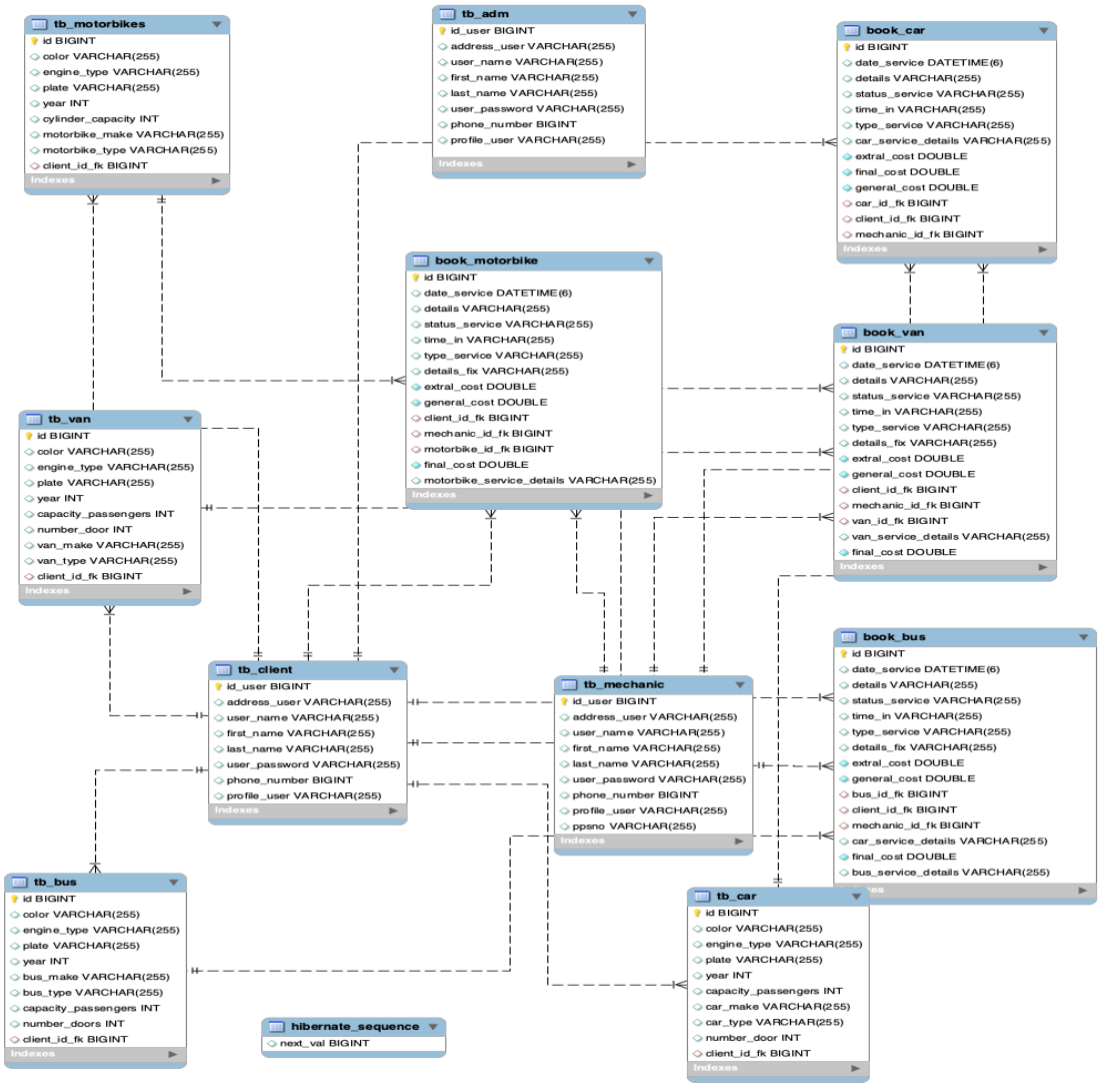
Relationship between bookCar/bookVan/bookBus/BookMotorbike and Car/Van/Bus/Motorbike are ONE TO ONE because one car belong just one book per time. Relationship is week because if the client wants book a service for his car this car have exist on system:



Ilustration 47: relationship between book and car

PHYSICAL DESIGN

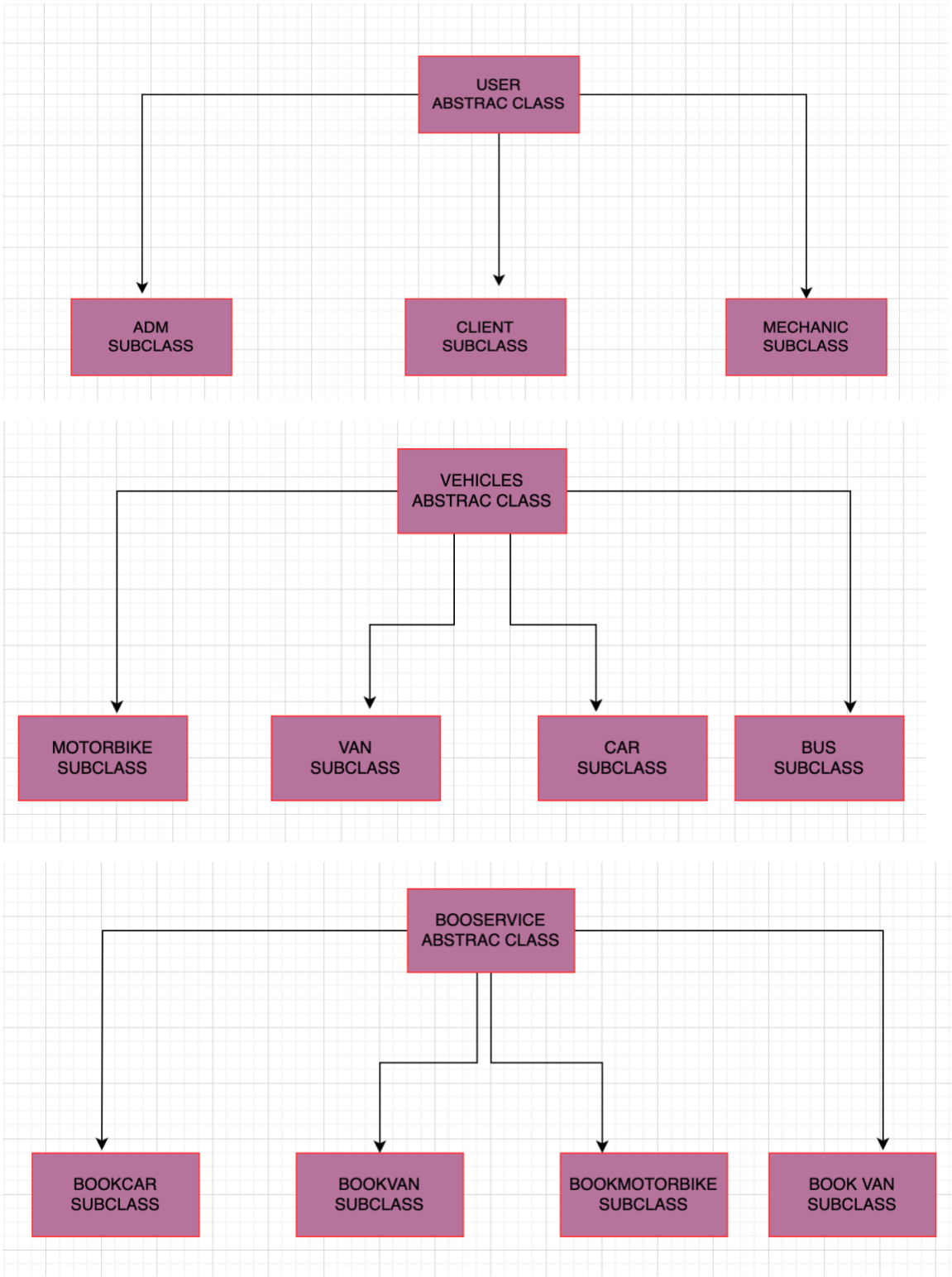
The following image shows the final physical design of the database:



Ilustration 47: PHYSICAL DESIGN

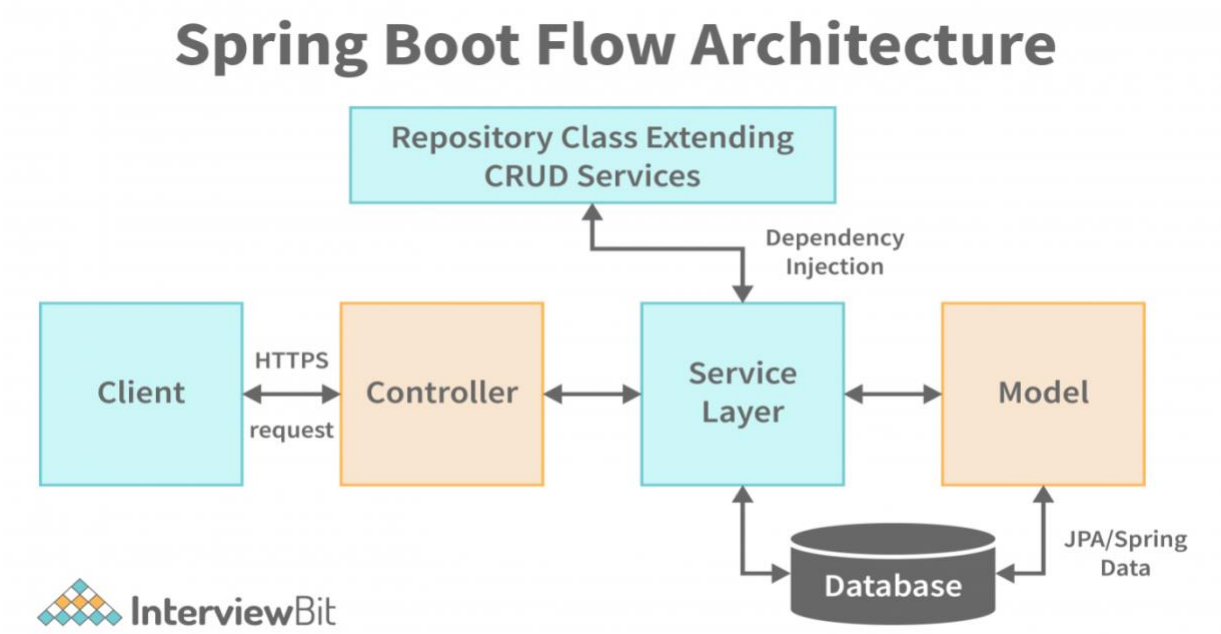
HOW I DECIDED CREATED ALL THOSE TABLE:

Following the structure of object oriented programming:



Ilustration 48: structure of class

FUNCTIONAL DESIGN:



Ilustration 50: Spring Book Architecture

As I mention before the Spring Boot is a popular Java-based framework used to build web applications and microservices. Here's a step-by-step overview of how Spring Boot typically works from the front-end to the database:

- Front-end Interaction:
  - The client (front-end) sends an HTTP request to the Spring Boot application. This request could be triggered by a user interacting with a web page, a mobile app, or any other client application.
  - The request can contain parameters, headers, and possibly a request body, depending on the nature of the operation being performed.
- Controller Layer:
  - In Spring Boot, the DispatcherServlet is the front controller that handles incoming HTTP requests.
  - It receives the HTTP request and routes it to the appropriate controller based on the URL mapping defined in the application.
- Service Servlet:
  - The DispatcherServlet routes the request to the appropriate controller class, which is responsible for processing the request.
  - The controller interacts with the business logic of the application to perform the required operations.
- Data Access Layer:

- The data access layer, also known as the persistence layer, interacts with the database.
- Spring Boot provides various options for data access, such as Spring Data JPA (Java Persistence API), JDBC (Java Database Connectivity), or other ORM (Object-Relational Mapping) frameworks.
- The data access layer typically performs CRUD (Create, Read, Update, Delete) operations on the database.

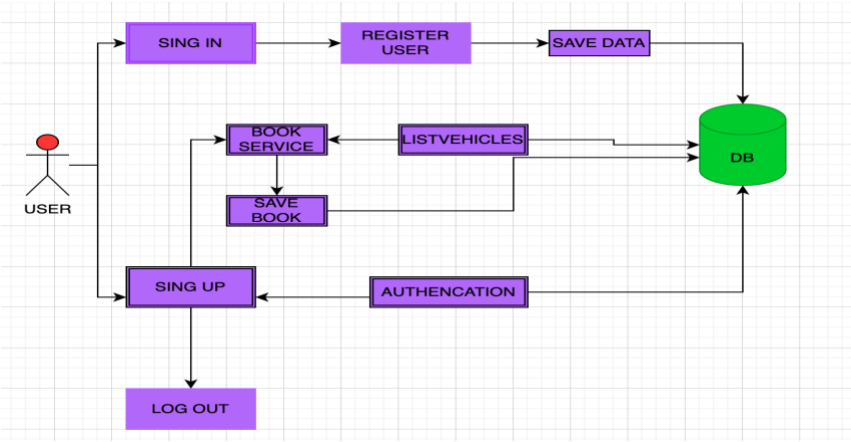
- Database Interaction:
  - The data access layer communicates with the database to perform the required operations;
  - The database processes the received SQL or other query languages to perform data manipulation or retrieval;
  - If the operation is successful, the database returns the requested data or an acknowledgment to the data access layer;

- Service and Controller Interaction:
  - If required, the service layer can perform additional processing on the data returned by the data access layer before passing it back to the controller.
  - The controller prepares the response to be sent back to the client.

- Response to Client:
  - The controller generates an HTTP response containing the result of the operation or the requested data.
  - The DispatcherServlet sends the response back to the client.

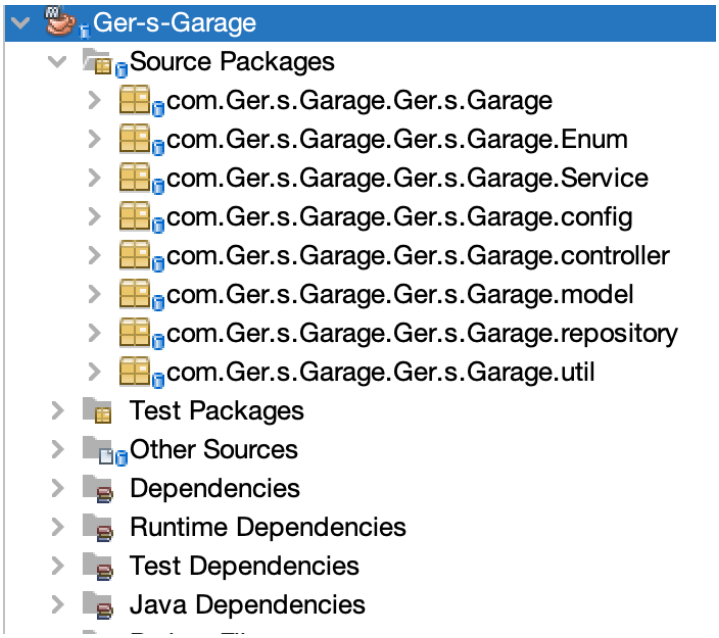
- Front-end Display:
  - The client (front-end) receives the HTTP response and processes it to display the data or perform further actions as required.

The following diagram shows a general idea of the functional design of the Application based on the user actions and taking into consideration the requirements of the Application:



Ilustration 52: general idea of project

Follow the structure of project using Spring boot and the accomplishing of all the package:



Ilustration 53: packages

CHAPTER IV: IMPLEMENTATION OF THE SYSTEM

This chapter is meant to explain the Development phase of the project.

ARCHITECTURE:

As it was already mentioned, the 'GER'S GARAGE' was developed using Spring Boot framework from Java witch support all the idea of project and keep the good structure as MVC.

TECHNOLOGIES IMPLEMENTED

FRONT END:

- HMTL;
- Java Script;
- CSS;
- Bootstrap framework;
- Thymeleaf;
- Apache Tomcat as Web server tool, to provide a HTTP web server environment;

BACK END

- **Spring Data JPA**: This module simplifies database access by providing an abstraction layer over the JPA (Java Persistence API). It allows you to interact with relational databases using object-oriented approaches.
- **Hibernate**: Often used as the JPA provider, Hibernate is an Object-Relational Mapping (ORM) framework. It maps Java objects to database tables, making database interaction more intuitive.
- **Spring Security**: For handling authentication and authorization, Spring Security offers a wide range of features to secure your application, including user authentication, role-based access control, and more.
- **Spring Boot Starter Web**: This starter module provides essential components for building web applications, including embedded web servers like Tomcat, Jetty, or Undertow, and tools for handling HTTP requests and responses.
- **Spring Boot Starter Data**: This starter module simplifies database operations by providing pre-configured data sources and JPA support, which makes it easier to interact with databases.
- **iText** is a powerful and versatile Java library for creating and manipulating PDF documents. It offers various signing functionalities including creating PDF documents, adding text, images, tables, charts, form filling, digital and much more.

DATABASE

- Relational Database (MySQL), to store the data of the program;
- Storage engine for MySQL;

DEVELOPMENT

The table below show up step by step how was implemented the system:

Component	Action	IDE	Technology	Issue/Challenge	Solution
Spring boot	Created the project Spring book to run in NetBeans	NetBeans	Sprint Boot	In the beginner I had problems about the version of Java	It was research the message that show up in some forum of developed. Finally I find out that only run in Java 17.
Database	Create the DB and make the connection with my project in NetBeans	Workbench	Workbench		
Back end	CREATE all the entity(class) and became them in to table in DB	NetBeans	Java	problems in generate the tables when the programs run	It was done the downgrade of Spring Boot Version.
Front-end	Created the pages that was necessary for each functionality	NetBeans	Thymeleaf		
Back-end	Implement the CRUD's users(client/admin, mechanic)	NetBeans	Java/ Thymeleaf	Problems about passing the object from front to back	It was done research how the thymeleaf works with back.
Back-end	Implement the CRUD's vehicles(car/van, motorbike/bus)	NetBeans	Java/ Thymeleaf		

Back-end/front-end	Implement the Spring security for make the permission of users.	NetBeans	Java/ Thymeleaf	Problems about the version of Spring Security.	It was done downgrade of version.
Back-end/front-end	Implement the login of client/admin/mechanic	NetBeans	Java/Spring Security		
Back-end	Implement the connection between the client and vehicle when the user add new vehicles	NetBeans	Java	It hard find out how to passing the ID of the user to method register car vehicles	It was using the method from Spring security that get the id of user that It is login in the application.
Back-end/front-end	Implement the permission about all the users like just the client can add/edit/delete	NetBeans	Java/Spring Security		
Back-end	Implement the book Service for van/bus/motorbike/car it was done the CRUD Book Service.	NetBeans	Java	It hard get the client that was table vehicles as FK and storage as FK in table book Service	It passed the id of client as hidden and get that id in DB and storage as FK on the table book service.
Back-end/front-end	It created the bottom allocate book to mechanic	NetBeans	Java/ Thymeleaf		
Back-end/front-end	Implement the bottom for edit the book service and add some labels as :extra cost, minimum cost, details about the service	NetBeans	Java/ Thymeleaf		

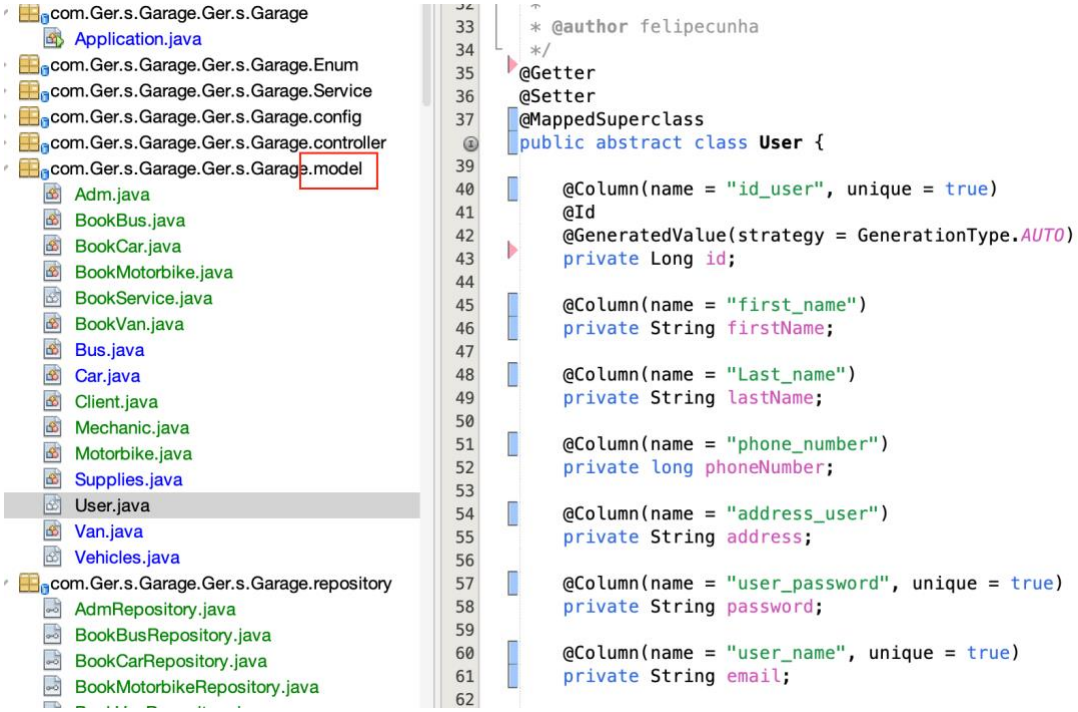
Back-end	Implement the method that set up the minimum cost for the service accordingly of type of service.	NetBeans	Java		
Back-end	Implement the logical for sum the minimum cost plus extra cost .	NetBeans	Java		
front-end	Implement the validation about all the label for make sure the user type the right information for each label.	NetBeans	Java Script		
front-end	Implement the function for do not allow the user book service on Sunday/and allow the user book just 9:00 to 17:00	NetBeans	Java Script		
Back end	Implement the generate invoice for each book service	NetBeans	Java/ Thymeleaf		

IMPLEMENTATION OF THE SYSTEM:

I had decide using abstract class for created the project.

As I decide created Abstract class as USER and the subclass as CLIENT/ADMIN/MECHANIC the database behaviour the same logical:

User- Abstract class:



Ilustration 54: Abstract class

I had used the notation @MappedSuperclass that means when the system run the Java know that It is a superclass and will not generate the table in data base, Will generate the table just in sub class through notation @Entity that means every tables that has this notation will become table on DB and all attributes of this class will became labels in DB.

Ilustration 55: subclass class

It was used the same logical for Book Service/vehicles:

Book service Abstract class:

```
*/

@Getter
@Setter
@MappedSuperclass
public abstract class BookService {

    @Column
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;

    @Enumerated(EnumType.STRING)
    private TypeService typeService;

    private String details;

    private String timeIn;


    @DateTimeFormat(pattern="yyyy-MM-dd")
    private Date dateService;
```

Ilustration 56: Book service Abstract class

One of the Subclass from book service:

```
@Entity
@Getter
@Setter
public class BookCar extends BookService {

    @ManyToOne
    @JoinColumn(name = "car_id_fk")
    private Car car;

    @ManyToOne
    @JoinColumn(name = "mechanic_id_fk")
    private Mechanic mechanic;

    @ManyToOne
    @JoinColumn(name = "client_id_fk")
    private Client client;

    private  double generalCost;
    private  double extralCost;
    private  String CarServiceDetails;
    private  double finalCost;
```

Ilustration 57: subclass class

Abstract class vehicle :

```java
@Getter
@Setter
@MappedSuperclass
public abstract class Vehicles {

    @Column
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;

    @Column
    private Integer year;

    @Column
    private String color;

    @Column
    private String plate;

    @Enumerated(EnumType.STRING)
    private EngineTypes engineType;

    public Vehicles(Integer year, String color, String plate, EngineTypes engineType) {
        this.year = year;
        this.color = color;
        this.plate = plate;
        this.engineType = engineType;
    }

    public Vehicles() {
    }
```

Ilustration 58: Abstract vehicle class

One of subclass from vehicle class:

```java
@Entity
@Getter
@Setter
@Table(name = "TB_CAR")
public class Car extends Vehicles {

    private Integer numberDoor;
    private Integer capacityPassengers;

    @Enumerated(EnumType.STRING)
    private CarMakes carMake;

    @Enumerated(EnumType.STRING)
    private CarTypes carType;

    @ManyToOne
    @JoinColumn(name = "client_id_fk")
    private Client client;


    public Car(Integer numberOfDoor, Integer capacityPassengers, CarMakes carMake, CarTypes carT
        super(year, color, plate, engineType);
        this.numberDoor = numberOfDoor;
        this.capacityPassengers = capacityPassengers;
        this.carMake = carMake;
        this.carType = carType;
        this.client = client;
    }
```

Ilustration 59: Abstract car subclass

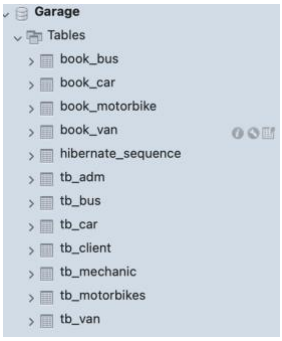
Connection with database:

As was mentioned before in your **application.properties** or **application.yml** file, It is define properties related to the database connection. These properties include information such as the database URL, username, password, driver class, and any other database-specific configuration:

```
spring.datasource.username=root
spring.datasource.password=Positivo09
spring.datasource.url=jdbc:mysql://localhost:3306/Garage?userTimezone=true&serverTimezone=UTC
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL8Dialect
spring.jpa.show_sql=true
spring.jpa.hibernate.ddl-auto = update
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

Ilustration 60: connection with DB

Tables created in database:



Ilustration 61: Table in DB

Query Methods: Spring Boot's auto-generated query methods in repository interfaces allow to perform common CRUD operations without writing explicit SQL queries. These methods are based on the method names and follow specific naming conventions.

How is possible to use CRUD ?

First was created the object in front-end for pass this object to back and finally save in DB.

For show up the form below It was used the method GET in CLASS ClientController.

ClientController- Each entity will have one Controller and this class will have the annotation @Controller When annotate a class with **@Controller**, it indicates that this class will handle incoming HTTP requests. Each public method within the controller class is typically responsible for handling a specific URL or request mapping.

In addition to the **@Controller** annotation, will often see methods within the controller annotated with **@RequestMapping** (or other specific mapping annotations like **@GetMapping**, **@PostMapping**, etc.). These annotations define the URLs or paths that the methods should handle.

@GetMapping – show up the forms to user:

```
@GetMapping("/registerClient")
public ModelAndView registerClientGet(Client client) {

    ModelAndView mv = new ModelAndView("client/registerClient");
    mv.addObject("client", new Client());
    mv.addObject("profiles", Profile.values());
    return mv;
}
```

Ilustration 62: Get forms from front-end

When the user press SAVE the object 'client; will sent to method POST and save in data base.

```
<!-- conteúdo principal -->
<form th:action="@{/client/registerClient}" method="post" th:object="${client}" enctype="multipart/form-data">
    <div class="row">
        <!-- Use <label> elements with "for" attribute instead of enclosing them in <code> -->
        <div class="mb-3">
            <label for="name" class="form-label">First Name</label>
            <input type="text" class="form-control" autocomplete="off" th:field="*{firstName}" id="firstName" placeholder
        </div>

        <div class="mb-3">
            <label for="lastName" class="form-label">Last Name</label>
            <input type="text" class="form-control" autocomplete="off" th:field="*{lastName}" id="lastName" placeholder="
        </div>

        <!-- Move the <code> element outside of the <div> -->
        <div class="mb-3">
            <label for="phoneNumber" class="form-label">Phone number (Ireland)</label>
            <input type="text" class="form-control" autocomplete="off" th:field="*{phoneNumber}" id="phoneNumber" placeho
        </div>

        <div class="mb-3">
            <label for="address" class="form-label">Address</label>
            <input type="text" class="form-control" autocomplete="off" th:field="*{address}" id="address" placeholder="ad
        </div>

        <div class="mb-3">
            <label for="userName" class="form-label">email</label>
            <input type="email" class="form-control" autocomplete="off" th:field="*{email}" id="email" placeholder="email
        </div>

        <div class="form-group col-6">
            <label for="password" class="lblCliente">Password:</label>
            <div class="password-container">
                <input type="password" class="form-control" name="password" id="password" placeholder="Password:" require
                <input type="checkbox" id="showPassword" onclick="togglePasswordVisibility()">
                <label for="showPassword">Show Password</label>
            </div>
        </div>
```

Ilustration 63: Page HTML register user

@PostMapping – Save the object in DB:

```java
@PostMapping("/registerClient")
public RedirectView registerClient(@ModelAttribute Client client) {

    ModelAndView mv = new ModelAndView("login/login");

    mv.addObject("client", client);
    clientRepository.save(client);
    System.out.println("Salved : " + client.getFirstName() + " " + client.getLastName());

    return new RedirectView("/");
}
```

Ilustration 64: storage the client on DB by JPA.

How is possible save the client in data base ?

For each entity was created the interface called repositoryClient for example, where this class extends JPARepository where has all the method like save/delete/list and etc:

```java
@Repository
public interface ClientRepository extends JpaRepository<Client,Long> {
   /**
    * find the user by UserName in dataBase.
    * @param email
    * @return
    */
  Optional<Client> findByEmail(String email);
```

Ilustration 65: Client repository.

Example of another methos that was used for make the CRUD:

```java
@GetMapping("/editClient/{id}")
public ModelAndView editClient(@PathVariable("id") Long id) {

    ModelAndView mv = new ModelAndView("client/editClient");
    mv.addObject("client", clientRepository.findById(id));

    return mv;
}

@PostMapping("/editClient")
public RedirectView editClient(Client client) {
    ModelAndView mv = new ModelAndView("/index");
    clientRepository.save(client);
    return new RedirectView("/");
}

@GetMapping("/deleteClient/{id}")
public RedirectView deleteCliente(@PathVariable("id") Long id) {

    ModelAndView mv = new ModelAndView("/index");
    clientRepository.deleteById(id);
    return new RedirectView("/");
}

@GetMapping("/listClient")
public ModelAndView listUsers() {
    ModelAndView mv = new ModelAndView("/client/listClient");
    mv.addObject("clients", clientRepository.findAll());
    return mv;
}

@GetMapping("/editClientProfile")
public ModelAndView editProfileClient(@RequestParam("id") Long id) {

    ModelAndView mv = new ModelAndView("/client/editClientProfile");
    mv.addObject("client", clientRepository.findById(id));

    return mv;
}
```

Ilustration 66: Methods from JPA

It was used the same structure for implement the CRUD of BOOKSERVICE/VECHIELS/.

THE MAIN LOGICAL OF SYSTEM

BOOK SERVICE

Page hmtl:

In the page below It was passing the id of user/idCar hidden to method post:

```
<form method="post" th:action="@{/car/registerBookServiceCar}" th:object="${bookCar}">
    <input type="hidden" th:name="${_csrf.parameterName}" th:value="${_csrf.token}">
    <input type="hidden" name="id" th:value="${#authentication.getPrincipal().getId()}">
    <input type="hidden" name="idCar" th:value="${idCar}">

    <div class="row">
        <!-- field nome -->

        <!-- ... código anterior ... -->

        <div class="form-group col-6">
            <label for="typeService" class="lblCliente">Type of Service </label>
            <select name="" id="" class="form-control" th:field="*{typeService}" required>
                <option th:each="typeService : ${typeServices}" th:text="${typeService}" th:value="${typeService}"
            </select>
        </div>

        <div class="form-group col-3">
            <label for="nome" class="lblCliente">Date of book</label>
            <input th:field="*{dateService}" type="date" class="form-control" autocomplete="off" id="dateService"
            <span id="warningMessage" style="color: red; display: none;">Sundays are not allowed. Please choose an
        </div>



        <div class="form-group col-3">
            <label for="timeIn" class="lblCliente">Time</label>
            <input th:field="*{timeIn}" type="time" class="form-control" autocomplete="off" id="timeIn" required m
            <span id="errorMessage" style="color: red; display: none;">Please select a time between 9:00 and 17:00
        </div>


        <div class="form-group col-3">
            <label for="nome" class="lblCliente">Describe details vehicles's problems</label>
            <input th:field="*{details}" type="text" class="form-control" autocomplete="off"  id="details" placeho

        </div>


        <div class="form-group col-6">
            <label for="typeService" class="lblCliente"></label>
            <select name="" id="" class="form-control" th:field="*{statusService}" hidden required>
                <option value="Booked">Default Value</option>
                <option th:each="statusService : ${statusServices}" th:text="${statusService}" th:value="${statusS
            </select>
```

Ilustration 67: Page register book

```
@PostMapping("/registerBookServiceCar")
public RedirectView registerNewService(@ModelAttribute BookCar bookCar, @RequestParam("id") Long idClient,
        @ModelAttribute("idCar") Long idCar) {

    Client client = clientRepository.findById(idClient)
            .orElseThrow(() -> new UsernameNotFoundException("User not found"));
    bookCar.setClient(client);

    Car car = carRepository.findById(idCar)
            .orElseThrow(() -> new UsernameNotFoundException("Car not found"));
    bookCar.setCar(car);

    String typeService = bookCar.getTypeService().getTypeService();

    bookCar.setGeneralCost(bookServiceImp.checkService(typeService));

    ModelAndView mv = new ModelAndView("/index");
    bookCarRepository.save(bookCar);

   return new RedirectView("/");
}
```

Ilustration 68: Storge the book in DB.

After get those IDs It was called the method find ClientbyId from JPARepository for get the client that who wants book the service. It was done the same for get the Car.

Before save the object to BookCar in DB It was set up the client/car as FK in the table BookCar.

There is two method that was created for set up the minimum cost of type of service and another one for update the final cost if there is extra cost in service, All those methos are in the picture from method post.

Final result in DB:

id	date_service	details	status_service	time_in	type_service	car_service_details	extral_cost	final_cost	general_cost	car_id_fk	client_id_fk	mechanic_
16	2023-08-10 23:00:00.000000	Door is not closing.	Booked	14:46	Repair_Fault	Oil change €30 Brake pad ch...	80	230	150	7	1	4
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Ilustration 68: Table book service.

PRINT INVOICE

For print the invoice was used the library from java itext as was mentioned before.

Class that build the PDF :

```java
public class GenerateBookCarInvoicePDF {

    private BookCar bookCar;

    public GenerateBookCarInvoicePDF(BookCar bookCar) {
        this.bookCar = bookCar;
    }

    private void writeTableHeader(PdfPTable table) {
        Font font = FontFactory.getFont(FontFactory.HELVETICA);
        font.setColor(Color.WHITE);

        addHeaderCell(table, "Client", font);
        addHeaderCell(table, "Phone-Number", font);
        addHeaderCell(table, "Vehicle", font);
        addHeaderCell(table, "Details about the fix", font);
        addHeaderCell(table, "General Cost", font);
        addHeaderCell(table, "Extra cost", font);
        addHeaderCell(table, "Final cost", font);
    }

    private void addHeaderCell(PdfPTable table, String text, Font font) {
        PdfPCell cell = new PdfPCell(new Phrase(text, font));
        cell.setBackgroundColor(Color.BLACK);
        cell.setPadding(5);
        table.addCell(cell);
    }

    private void writeTableData(PdfPTable table) {

        table.addCell(bookCar.getClient().getFirstName());
        table.addCell(String.valueOf(bookCar.getClient().getPhoneNumber()));
        table.addCell(bookCar.getCar().getCarType().toString());
        table.addCell(bookCar.getCarServiceDetails());
        table.addCell(String.valueOf(bookCar.getGeneralCost()));
        table.addCell(String.valueOf(bookCar.getExtralCost()));
        table.addCell(String.valueOf(bookCar.getFinalCost()));

    }
```

```java
public void export(HttpServletResponse response) throws DocumentException, IOException {
    Document document = new Document(PageSize.A4);
    PdfWriter.getInstance(document, response.getOutputStream());

    document.open();
    Font font = FontFactory.getFont(FontFactory.HELVETICA_BOLD);
    font.setSize(18);
    font.setColor(Color.BLACK);

    Paragraph p = new Paragraph("Invoice", font);
    p.setAlignment(Paragraph.ALIGN_CENTER);

    document.add(p);

    PdfPTable table = new PdfPTable(7);
    table.setWidthPercentage(100f);
    table.setWidths(new float[]{1.5f, 2.0f, 2.0f, 4.0f, 1.5f, 1.5f, 1.5f});

    table.setSpacingBefore(10);

    writeTableHeader(table);
    writeTableData(table);

    document.add(table);

    document.close();

}
```

Ilustration 69: Logical print the invoice.

Method generate invoice:

```
@GetMapping("/generateInvoiceBookCar/{id}")
public void exportToPDF(HttpServletResponse response, @PathVariable("id") Long id) throws DocumentException,
    response.setContentType("application/pdf");
    DateFormat dateFormatter = new SimpleDateFormat("yyyy-MM-dd_HH:mm:ss");
    String currentDateTime = dateFormatter.format(new Date());

    String headerKey = "Content-Disposition";
    String headerValue = "attachment; filename=invoice" + currentDateTime + ".pdf";
    response.setHeader(headerKey, headerValue);

    BookCar booKCar;

     booKCar =bookCarRepository.findById(id)
    .orElseThrow(() -> new UsernameNotFoundException("User not found"));

    GenerateBookCarInvoicePDF exporter = new GenerateBookCarInvoicePDF(booKCar);
    exporter.export(response);

}
```

Ilustration 70: Method generate Invoice.

How was done relationship between the table form back-end:

Many-to-One Relationship: A many-to-one relationship is the reverse of a one-to-many relationship. Many instances of one entity are associated with a single instance of another entity. Use the **@ManyToOne** annotation.

```
@Entity
@Getter
@Setter
public class BookCar extends BookService {

    @ManyToOne
    @JoinColumn(name = "car_id_fk")
    private Car car;

    @ManyToOne
    @JoinColumn(name = "mechanic_id_fk")
    private Mechanic mechanic;

    @ManyToOne
    @JoinColumn(name = "client_id_fk")
    private Client client;

    private  double generalCost;
```

Ilustration 70: relationship between the tables.

This example show up was set up relationship between BookCar to Client/mechanic/Car.

I was done the same logical for another relationship in diagrama.

SPRING SECURITY

As was mentioned before I had used the Spring security for make the permission of user.

The method below show up how to get the user that It is login:

```
@Override
public Collection<? extends GrantedAuthority> getAuthorities() {

    Profile profile = client.getProfile();

    if (profile == Profile.ADM) {
        profile = Profile.ADM;

    } else {
        profile = Profile.CLIENT;
    }

    return AuthorityUtils.createAuthorityList(profile.toString());
}
```

Ilustration 71: get the profile user.

Through the method getAuthorities from Spring Boot It possible set up the permission accordingly profile of user.

Example how was implement the permission just the mechanic user is allow update the status of service:

```
<td>
    <a  th:hidden="${#authorization.expression('hasAuthority(''CLIENT'') or  hasAuthority(''ADM'')')}" class="
        <i class ="fas fa-edit"></i>update Status</a>

        <td>
    <a  th:hidden="${#authorization.expression('hasAuthority(''CLIENT'') or  hasAuthority(''MECHANIC'')')}"cl
```

Ilustration 72: make the permission by user.

In this case It hidden the bottom update Status for CLIENT/ADM users.

Challenges:

The front-end implied one of the biggest challenges, in terms of connecting it with the back end. The hard part for me It was how pass the understand how to pass the object and their attributes

to back-end. But after done the first one It was easy implemented to other because the logical was same.

VIII. CHAPTER V:TESTING & EVALUATION

This chapter will cover the final test and evaluation process of project Spring Boot. During the development of the application, it was necessary to make changes to the structure according to the system requirements.

This chapter will cover the input and output for each functionality.

- **ACESS TO THE SYSTEM:**

Test input	Expected result	Actual Result	Comment
Fill the form register new client	Storage the client on Table and back to page login	Correct.	It is not show up any message to user like "You have been registered". In additional It show up message of error if the user put some wrong information in any labels.
Make the login	Authentication of user in the system and access the menu of system.	Correct.	

- **EDIT USER'S PROFILE:**

Test input	Expected result	Actual Result	Comment
Edit profile client/admin/mechanic	Make the update that the user want and save in DB.	Correct.	It is not show up any message to user like "You have been registered". In additional It show up message of error if the user put some wrong information in any labels.

- **MAKE LOGOUT:**

Test input	Expected result	Actual Result	Comment
Users make the logout	Return to page login	Correct.	It is not show up any message to user.

- **CAR:**

Test input	Expected result	Actual Result	Comment
Press the bottom list the cars.	List all the cars registered for the user.	Correct.	
Fill the form car's details	Get the id of user who is login and car's details that was provide from user and storage Client as FK in CAR'S table and storge the car in table as well.	Correct.	It is not show up any message to user like "You have been registered". In additional It show up message of error if the user put some wrong information in any labels.
Edit the form car's details	Salve the updates car's tables.	Correct.	It is not show up any message to user like "You have been registered". In additional It show up message of error if the user put some wrong information in any labels.
Delete car	Delete the car in CAR's table.	Correct	It is not show up any message to user.

- **VAN:**

Test input	Expected result	Actual Result	Comment
Press the bottom list the vans.	List all the vans registered for the user.	Correct.	
Fill the form van's details.	Get the id of user who is login and van's details that was provide from user	Correct.	It is not show up any message to user like "You have been registered". In

	and storage Client as FK in VAN'S table and storge the van in table as well.		additional It show up message of error if the user put some wrong information in any labels.
Edit the form van's details	Salve the updates van's tables.	Correct.	
Delete the van	Delete van in van's table.	Correct	It is not show up any message to user.

- **BUS:**

Test input	**Expected result**	**Actual Result**	**Comment**
Press the bottom list the buses.	List all the buses registered for the user.	Correct.	
Fill the form bus's details.	Get the id of user who is login and bus's details that was provide from user and storage Client as FK in BUS'S table and storge the bus in table as well.	Correct.	It is not show up any message to user like "You have been registered". In additional It show up message of error if the user put some wrong information in any labels.
Edit the form bus's details	Salve the updates bus's tables.	Correct.	
Delete bus	Delete bus in bus's table.	Correct	It is not show up any message to user.

- **MOTORBIKE:**

Test input	**Expected result**	**Actual Result**	**Comment**
Press the bottom list the motorbikes.	List all the **motorbikes** registered for the user.	Correct.	
Fill the form motorbike's details.	Get the id of user who is login and bus's details that was provide from user	Correct.	It is not show up any message to user like "You have been registered". In

	and storage Client as FK in MOTORBIKES'S table and storge the motorbike in table as well.		additional It show up message of error if the user put some wrong information in any labels.
Edit the form motorbike's details	Salve the updates motorbike's tables.	Correct.	It is not show up any message to user like "You have been registered". In additional It show up message of error if the user put some wrong information in any labels.
Delete motorbike	Delete motorbike in motorbike's table.	Correct	It is not show up any message to user.

- **BOOK SERVICE VAN/BUS/MOTORBIKE/CAR:**

Test input	**Expected result**	**Actual Result**	**Comment**
Fill the form book service.	Get the idUser/idVehicle and all the details provided by the user storage in vehicle's table.	Correct.	It is not show up any message to user like "You have been registered". In additional It show up message of error if the user put some wrong information in any labels.
Press the bottom for see all the books that was done by the user.	Lista all the book by user that It is log in in the system .	Correct.	
Fill the forms edit book van/bus/car/motorbike	Salve the updates book in van/bus/car/motorbike table.	Correct.	It is not show up any message to user like "You have been registered". In additional It show up message of error if the user put some wrong information in any labels.

Delete book van/bus/car/motorbike	Delete van/bus/car/motorbike in DB.	Correct	It is not show up any message to user.

- **ADMIN MENU:**

Test input	Expected result	Actual Result	Comment
Press the bottom list all the client/admin/mechanic registered on system.	List all client/admin/mechanic from DB.	Correct.	
Fill the form mechanic/admin user.	Get the details provided by the ADMIN user and storge in DB.	Correct.	It is not show up any message to user like "You have been registered". In additional It show up message of error if the user put some wrong information in any labels.
Edit the form bus's details	Salve the updates bus's tables.	Correct.	
Delete bus	Delete bus in bus's table.	Correct	It is not show up any message to user.

- **ADMIN MENU ABOUT THE BOOK:**

Test input	Expected result	Actual Result	Comment
List all the books that was done.	List all van/car/motorbike /bus from DB.	Correct.	
Choose the mechanic that It is registered on system and allocate to book.	List all the mechanics from DB and storage the mechanic that was choose by Admin in the Book's table as FK.	Correct.	It is not show up any message to user like. and It not limited the amount of book by mechanic.
Press generate invoice for the book.	Generate the document as PDF with some details from book.	Correct.	

		Correct	It is not show up any message to user.

- **MECHANIC MENU ABOUT THE BOOK:**

Test input	Expected result	Actual Result	Comment
Press the bottom check all the books that It allocated for him.	List all the books that was allocated for him from DB.	Correct.	
Change the Status of book and fill extra cost and provide some details about the fix like list of service that It was done and how much cost those.	Get the details (extra cost/list of services) and save those information in DB. The label extra cost It add to final cost in DB .	Correct.	

- **TEST ABOUT THE REQUIMENT:**

Test input	Expected result	Actual Result	Comment
Client try book the service Sunday.	Show up a message to user Sunday It is not available.	Correct.	
Client try book the service when there is no book available. Limit of books is 6 per day.	Show up the message to user that there is no book available for this day.	Correct.	It is show up the message that there is no book available just on console. I did have type to send the message to front.
Type the plate wrong when the client register the new car .	Show up the message how the correct format of plate for Ireland.	Correct.	
Type email wrong	Show up the message to user that Email It is wrong.	Correct	
Type number instead letter for any label.	Show up the message to user that label accept just letters.	Correct	
Type letter instead number.	Show up the message to user that label accept just number.	Correct	

Type wrong phone number	Do not allow the user type letter.	Correct	
User do not fill some label.	Show up the message that label have to be fill.	Correct	This validation It works all the label for any form that the user try fill.
Admin user allocate the book to mechanic.	Do not allow allocate the book to mechanic if the mechanic has two books for the day.	I did have time for implement It.	
Option other if there is no type/makes for vehicles on list	Allow the user type makes/types vehicles.	I did have time for implement It.	
Generate roster for any particular date.	Generate the PFD document	I did have time for implement It.	
Choose type of service when the client book the service	Add minimum cost accordingly type of service.	Correct.	
Choose the time of book before 9:00 or after 17:00	Show up the message to user that time of book It just 9:00 to 17:00.	Correct.	

Extra functionality:

- Method encodes (hashes) the given password using BCrypt

```
public class PasswordUtil {

    // This method encodes (hashes) the given password using BCrypt
    public static String encoder(String password){
        // Create an instance of BCryptPasswordEncoder
        BCryptPasswordEncoder encoderPasswod = new BCryptPasswordEncoder();

        // Encode the password and return the hashed result
        return encoderPasswod.encode(password);
    }
}
```

- Admin user has :
    - List all the clients registered on system.
    - Delete client/mechanic/admin
    - List all the mechanic registered on system.
    - List all the admin registered on system.

CHAPTER VI: CONCLUSIONS & FURTHER WORK

I'm truly proud of the journey I've undertaken with my Spring Boot project in Java. Initially, I believed that the primary challenge would be adapting to new technical concepts in English, but I soon discovered that wasn't the case. The real test was balancing the demanding schedule, work responsibilities, and managing college commitments, which proved to be the most significant obstacle.

Although I didn't achieve every goal I set for this project, I am confident that I gave it my all and performed exceptionally well given the constraints of time. The most daunting aspect for me was integrating the diverse technologies I had learned—front-end, back-end, and database—all of which I had studied separately. It was a challenge to understand how these components could seamlessly work together.

Opting for Spring Boot as my framework of choice was a decision that initially filled me with apprehension due to my lack of knowledge about it. However, through dedicated networking and effort, I gradually grasped how Spring Boot functions, and I'm convinced that this investment in learning will pay off by streamlining future project development. This framework has sparked my curiosity, motivating me to delve even deeper into it. I've even made plans to replicate the project using the Angular framework.

My journey to Ireland initially began as an opportunity to improve my English, but midway through, I took the life-changing decision to embark on a Higher Diploma in Computing Science. This choice has been the best decision I've made. It has allowed me to surpass my own expectations and has given me the confidence to tackle any technological challenge that comes my way.

In conclusion, I've learned a great deal from this experience, not only in terms of technical skills but also in terms of time management, adaptability, and personal growth. The journey has been challenging, but it's equipped me with the tools and knowledge to face future endeavors with confidence and determination.

FURTHER WORK

Keeping going ahead with this project I would like complete the functionality It is missing:

- Print the roster for any particular date;
- Show up the message to user to front-end that day It not available.
- Make list of service;
- Improve the layout of invoice.
- Add option other in list makes/type;

Extra functionality :

- Find the client/book by name;
- Find the car by client;

Make the same project using Angular Framework.

APPENDIX A: CLASS DIAGRAM

APPENDIX B: LINK GITRUB

https://github.com/FelipeCunha03/GARAGE-2022250

Instructions:

When the program run It is created 2 types of user just user Admin/mechanic. I used this for make the test and save time for record the demo. Bus you are free for make new register whatever user for test.

Mechanic user:

- Email: Carla.Cunha03@hotmail.com
- Password:Positivo11

Admin user:

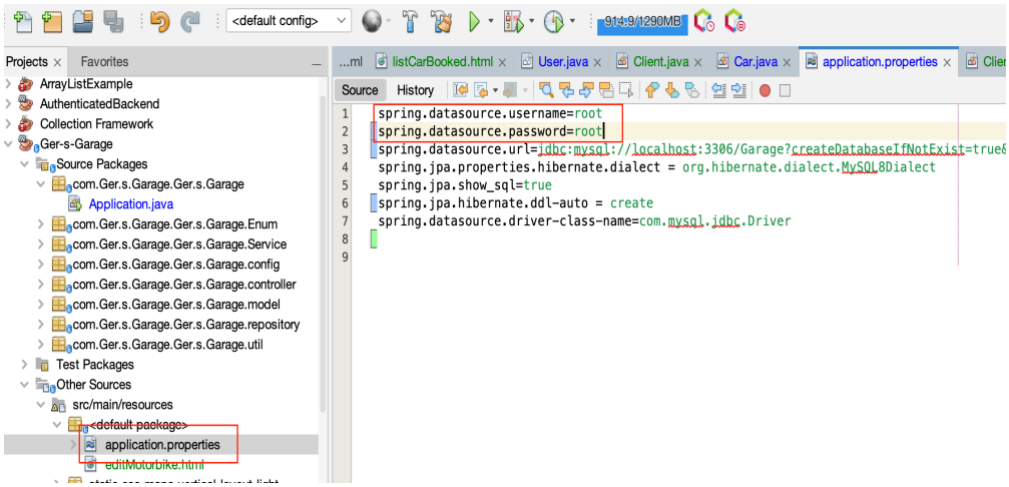
- Email: Bianca.Cunha03@hotmail.com
- Password: Positivo10

The program have to be execute in Java17 for avoid bug because version;

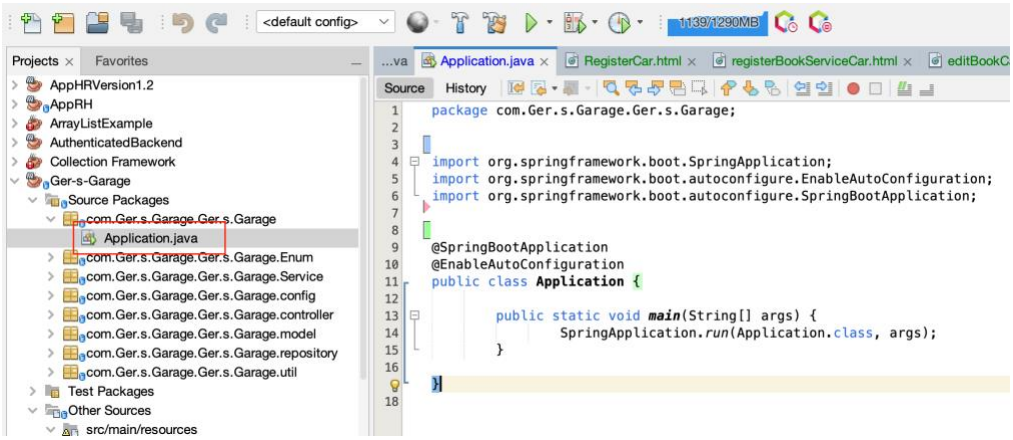
LinkWeb: http://localhost:8080/login;

Connection to DB: When the program run the DB and all the table are created automatic.

Just update password and user name in file below and save:



The program have to be run main class below:



There is one side menu on system and whatever page user go through this side-menu go as well, if the user try choose some option from side menu after went through whatever page It is not work, this happens sometimes just. I did not have time for fix it.

REFERENCES

Java point (2020). *Learn Spring Boot Tutorial - javatpoint*. [online] www.javatpoint.com. Available at: https://www.javatpoint.com/spring-boot-tutorial.

Maven (2020). *Maven Repository: Search/Browse/Explore*. [online] mvnrepository.com. Available at: https://mvnrepository.com.

Packtpub.com. (2019). *Packt | Programming Books, eBooks & Videos for Developers*. [online] Available at: https://www.packtpub.com/.

Stack Overflow. (2014). *unidirectional vs bidirectional in one-to-one relationship*. [online] Available at: https://stackoverflow.com/questions/21661856/unidirectional-vs-bidirectional-in-one-to-one-relationship [Accessed 12 Aug. 2023].

www.youtube.com. (2020). *Spring Boot | Curso Completo 2022*. [online] Available at: https://www.youtube.com/watch?v=LXRU-Z36GEU&t=6554s [Accessed 12 Aug. 2023].

www.youtube.com. (2022). *Spring Security without the WebSecurityConfigurerAdapter*. [online] Available at: https://www.youtube.com/watch?v=s4X4SJv2RrU [Accessed 12 Aug. 2023].