

UNIVERSITÀ DI PISA



Department of Information Engineering(DII)

Master of Science in Computer Engineering

**Second-hand Car Advertisement application
Based on Erlang distributed server**

Lecturer(s):

Professor Bechini

Student(s):

**Kamran Mehravar
Sultan Mahmud**

**Final Project for Distributed
Systems and Middleware Technologies Course.**

Academic Year 2022/23

Contents

1.System Overview	3
1.1System Requirements	3
1.1.1Functional Requirements	3
1.2.1 Non-Functional Requirements	4
2.System Architecture	5
2.1Components:	6
2.1.1 User Interface (UI):	6
2.1.3 MySQL Database:	10
2.1.4 Mnesia Database:	10
2.1.5 Apache Web Server:	11
2.2 Architecture Flow:	11
2.2.1 User Interaction:	11
2.2.2 Frontend-Backend Communication:	11
2.2.3 User Authentication and Role-Based Authorization:	12
3. Functional Modules	12
3.1 Security Module	12
3.2 Front-end Module:	12
3.2.1 User Management Module:	12
3.2.2 Advertisement Module:	12
3.2.3 Car Module:	12
3.2.4 Order Module:	13
4. Data Storage:	13
4.1 MySQL Database:	13
4.2 Mnesia Database:	13
5. Inter-Module Communication:	14
6. Project Structure:	15
6.1 Front end project:	15
6.2 Back end project:	15
7. Erlang project:	16
7.1 operations in the Erlang Module	16
7.2 Core Erlang	18
8.Deployment and Configurations	21
9. Conclusion:	22
10.User Manual	23
11.Functionality Test and System Engineering	27
11.1 Tomcat status	27
11.2 Apache Web server	27
11.3 Erlang System Management	28
11.4 Erlang functionality test	28
11.5 My SQL functionality test	29

1.System Overview

The distributed system is a web-based application designed to handle various functionalities related to user management, advertisements, cars, and orders. It consists of multiple modules, including a front-end developed using Angular, a backend REST API built with Spring Boot, and two separate databases: MySQL for user information and Mnesia for advertisement and order data. The system employs JWT-based authentication and role-based authorization to ensure secure access to different functionalities.

1.1 System Requirements

1.1.1Functional Requirements

The functional requirements use case has the following actors.

1.1.2 Anonymous User

These users are encouraged to register a user account in the system in order to participate in the advertising system.

The System must provide all of the pages and capabilities required to create a new user.

1.1.3 A *standard user*

A. Add (Create) New adverts:

As a user, you may quickly create and publish new adverts. You may post your listings with all important information, whether you're selling items, services, or anything else. The procedure is simple and ensures that your advertisements reach the intended demographic.

B. Read (View) adverts and Browse Listings:

Our platform allows you to easily view all accessible adverts. You may easily find a broad selection of possibilities by browsing through a comprehensive list of advertisements without providing any search parameters.

C. Change (Edit) Advert Information:

We recognise that information about your advertisements may change over time. As a result, we allow you to update and change the information in your published adverts. You have complete control over keeping your advertisements up to date, whether it's by changing the pricing, description, or any other pertinent info.

D. Delete (delete) Advertisements:

You may quickly delete an advertisement from the platform if it is no longer relevant or if you have successfully completed the transaction. The remove option allows you to easily manage your active listings.

E. Search Advertisements with extensive Filters:

We provide extensive filtering tools to help you expedite your search process. You may search for advertising by location, manufacturer (for example, Ford, Chevrolet, or BMW), and production year. This guarantees that you locate what you're searching for quickly and easily.

F. Send Buy (Order) Requests: When you see an advertising that piques your attention, you may simply begin a buy or order request using our platform. This allows for more fluid communication between buyers and sellers, helping the purchase process go more smoothly.

1.1.4 The administrator

A. Create (Add) a New User:

- CRUD Operation: Create
- Description: This function allows authorized administrators to add new users to the system by providing necessary details like name, email, and phone number.

B. Search Users by Their Phone Number:

- CRUD Operation: Read
- Description: This operation enables authorized users to search and retrieve user accounts based on the provided phone number. It helps to find specific users efficiently.

C. Update (Edit) Information Associated with a Specific Advert:

- CRUD Operation: Update
- Description: This function allows authorized users to edit and update the information associated with a specific advert. Users can modify details such as price, description, and other relevant data.

D. View an Advert and Perform a Remove Action:

- CRUD Operation: Read, Delete
- Description: This function allows users to view an advert and decide whether to remove (delete) it from the platform if it is no longer relevant or the item has been sold.

E. View all the Buy(Order) requests:

- CRUD Operation: Read
- Description: can do fillers for each Region, by Manufacturer (like Ford, Chevrolet, BMW, ..., by Production Year, by Price, and browse them

1.2.1 Non-Functional Requirements

Performance:

We want our users to have a pleasant and smooth experience when using our programme. One critical part of doing this is ensuring that the application responds as quickly as possible. Users should receive rapid and seamless replies to their actions or requests while interacting with our platform. This includes eliminating any delays or lags in system performance that might cause user displeasure. To achieve the fastest possible response times, we concentrate on optimising our application's code, simplifying database queries, and making the most use of our server resources.

Usability:

Our major objective is to develop an application that is simple to use and comprehend for all of our users. We believe on offering a user-friendly experience so that even inexperienced users may easily use the site. This includes creating a clear and intuitive user interface, establishing logical procedures, and ensuring that design components are consistent throughout the programme. We regularly seek user feedback through usability testing to discover areas for improvement and make required modifications to improve the overall user experience.

Database Information:

To optimise data storage and retrieval, our programme makes use of two separate databases, MySQL and Mnesia. MySQL is used to store vital and structured data that need extensive searching and reporting. It provides a strong solution for efficiently organising such information.

Mnesia, on the other hand, is a distributed, in-memory, and fault-tolerant database system that precisely meets the requirements of our application. It excels at managing high-speed reads and writes, making it suitable for storing time-sensitive or frequently accessed data.

We can improve performance and scalability for different types of data and use cases by deliberately using the characteristics of each system. Maintaining data consistency and integrity across both databases, on the other hand, is critical, and we take the required steps to guarantee easy synchronisation and administration.

Overall, the speed and usability of our application are top priority for us, and we are always striving to give the best experience possible for our valued users.

2.System Architecture

The distributed system is designed to provide a seamless user experience by displaying the User Interface (UI) in the browser and communicating with the backend Spring Boot REST API. The system utilizes multiple databases, namely MySQL and Mnesia, to store and manage different types of data efficiently. The architecture is built upon a distributed model, allowing the system to scale and handle high volumes of user interactions.

Our finished deployable object is a war. The war is deployed on the java application server *GlassFish/Tomcat*.

Spring framework has been used in the Java EE project to make the code easy and faster. Spring is a De-facto framework in Java EE development.

Reasons for using spring framework:

1. It simplifies the database operation like insertion, query etc.
2. Security implementation

By integrating Spring into our Java EE application, we can leverage its powerful features to build ***robust*** and ***modular*** enterprise applications more efficiently. The key is to set up the Spring context correctly and make sure that our Spring beans are properly wired into your Java EE components.

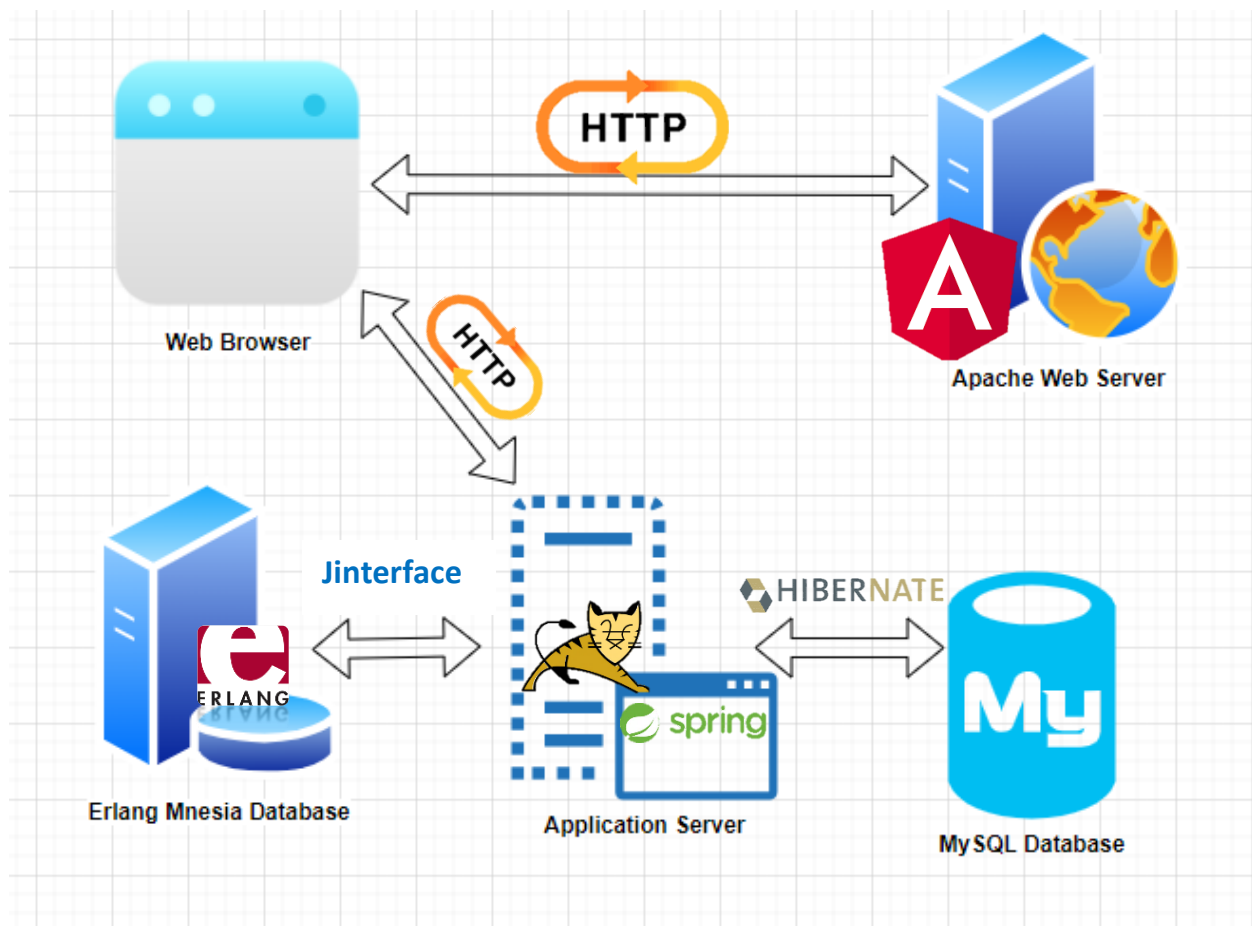


Figure 1: Simplified Architecture

2.1 Components:

2.1.1 User Interface (UI):

The UI is developed using Angular, a popular front-end framework, and is presented to users in their web browsers. It provides an intuitive and interactive interface through which users can interact with the system's functionalities.

2.1.2 Spring Boot REST API:

The backend of the system is implemented using Spring Boot, a powerful Java-based framework for building web applications. The REST API acts as an intermediary between the UI and the databases,

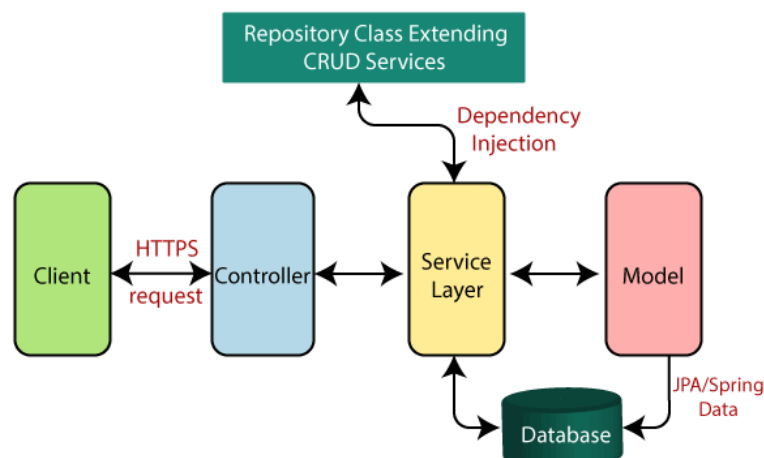


Figure 2: Spring Framework Architecture

handling user requests and responses. It follows RESTful principles and provides endpoints that respond to HTTP requests made by the UI.

- Spring Boot uses all the modules of Spring-like Spring MVC, Spring Data, etc. The architecture of Spring Boot is the same as the architecture of Spring MVC, except one thing: there is no need for **DAO** and **DAOImpl** classes in Spring boot.
- Creates a data access layer and performs CRUD operation.
- The client makes the HTTP requests (PUT or GET).
- The request goes to the controller, and the controller maps that request and handles it. After that, it calls the service logic if required.
- In the service layer, all the business logic performs. It performs the logic on the data that is mapped to JPA with model classes.

Spring API Controller implementation:

```
@ApiController
@RequestMapping(UrlConstants.AdvertisementManagement.ROOT + UrlConstants.AdvertisementManagement.GET_ALL)
public class AdvertisementController {

    6 usages
    private final AdvertisementService advertisementService;

    public AdvertisementController(AdvertisementService advertisementService) {
        this.advertisementService = advertisementService;
    }

    @PostMapping
    public Response create(@RequestBody AdvertisementDto advertisementDto, HttpServletRequest request, HttpServletResponse response) {
        return advertisementService.create(advertisementDto);
    }

    @GetMapping
    public Response getAll(HttpServletRequest request, HttpServletResponse response, Pageable pageable,
        @RequestParam(value = "export", defaultValue = "false") boolean isExport,
        @RequestParam(value = "search", defaultValue = "") String search,
        @RequestParam(value = "status", defaultValue = "") String status) {
        return advertisementService.getAll(pageable, isExport, search, status);
    }

    @GetMapping(UrlConstants.AdvertisementManagement.GET)
    public Response get(@PathVariable(value = "id") Long id, HttpServletRequest request, HttpServletResponse response) {
        return advertisementService.get(id);
    }

    @DeleteMapping(UrlConstants.AdvertisementManagement.GET)
    public Response delete(@PathVariable(value = "id") Long id, HttpServletRequest request, HttpServletResponse response) {
        return advertisementService.delete(id);
    }
}
```

Some Advantages of Spring Boot as follow:

- ✓ It is very easy to develop Spring Based applications with Java
- ✓ It provides Embedded HTTP servers like Tomcat, Jetty etc. to develop and test our web applications very easily.

- ✓ It provides lots of plugins to develop and test Spring Boot Applications very easily using Build Tools like Maven and Gradle.
- ✓ It provides lots of plugins to work with embedded and in-memory Databases very easily.
- ✓ It provides different data format like JSON, text, HTML, and XML.

The following figure shows the relationship between classes and interfaces.

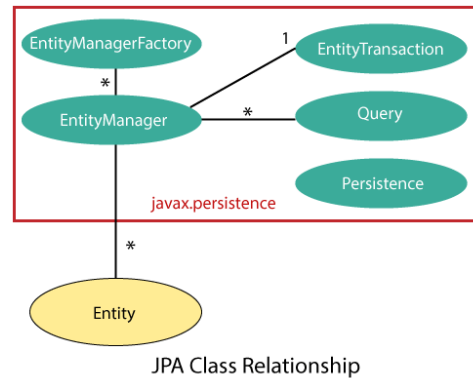


Figure 3: JPA Relation

- The relationship between EntityManager and EntityTransaction is **one-to-one**. There is an EntityTransaction instance for each EntityManager operation.
- The relationship between EntityManagerFactory and EntityManager is **one-to-many**. It is a factory class to EntityManager instance.
- The relationship between EntityManager and Query is **one-to-many**. We can execute any number of queries by using an instance of EntityManager class.
- The relationship between EntityManager and Entity is **one-to-many**. An EntityManager instance can manage multiple Entities.

There are some popular JPA implementations frameworks such as Hibernate, EclipseLink, DataNucleus, etc. It is also known as Object-Relation Mapping (ORM) tool. **We are using JPA. Hibernate which is the implementation of the JPA**

To be more specific on spring JPA Hibernate and Hibernate XML mapping as follow

Spring Data JPA:

- Uses annotations like **@Entity**, **@Table**, etc., for mapping Java classes to database tables.
- Provides higher-level abstraction and simplifies data access with repository interfaces and query methods.
- Eliminates the need for XML configuration by relying mainly on Java-based annotations.
- Reduces boilerplate code and offers convenient methods for common CRUD operations.

Hibernate XML Mapping:

- Requires separate XML files (e.g., **hibernate.cfg.xml**, ***.hbm.xml**) for mapping Java classes to database tables.
- Offers explicit control over mappings and relationships through XML configurations.
- Can become verbose and cumbersome as the number of entities and relationships increases.
- Requires XML parsing during application startup, which can add some overhead.

Spring JPA Hibernate implementation:

```
@Service("orderService")
@SLF4j
public class OrderServiceImpl implements OrderService {

    2 usages
    private final OrderRepository orderRepository;
    2 usages
    private final ModelMapper modelMapper;

    2 usages
    private final UserRepository userRepository;
    2 usages
    private final AdvertisementRepository advertisementRepository;
    4 usages
    private final EntityManager entityManager;

    public OrderServiceImpl(OrderRepository orderRepository, ModelMapper modelMapper, UserRepository userRepository, AdvertisementRepository advertisementRepository, EntityManager entityManager) {
        this.orderRepository = orderRepository;
        this.modelMapper = modelMapper;
        this.userRepository = userRepository;
        this.advertisementRepository = advertisementRepository;
        this.entityManager = entityManager;
    }

    @Override
    public Response create(OrderDto orderDto) {
        try {
            String username = AuthUtil.getUserName();
            Order order = new Order();
            BeanUtils.copyProperties(orderDto, order);
            UserEntity user = userRepository.getOne(username);
            order.setUser(user);
            order.setStatus(OrderStatusConstants.PENDING);
            order.setOrderDate(new Date());
            Advertisement advertisement = advertisementRepository.findByIdAndIsActiveTrue(orderDto.getAdvertisement().getId());
            order.setAdvertisement(advertisement);
            order = orderRepository.save(order);
            return ResponseUtils.getSuccessResponse(HttpStatus.CREATED, content: null, message: "Order created successfully");
        } catch (Exception e) {
```

```
spring.datasource.url=jdbc:mysql://localhost:3306/car_ad?useSSL=false
spring.datasource.username=root
spring.datasource.password=K@mi09373734241
```

```
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect
spring.datasource.driver.class=com.mysql.cj.jdbc.Driver
```

Spring Data JPA with Hibernate benefits:

- Simplified data access and reduced boilerplate code.
- Annotation-based configuration for concise code.

- Automatic query generation and custom query support.
- Seamless integration with the Spring ecosystem.
- Transaction management and data integrity.
- Portability and flexibility with database independence.
- Performance optimization through caching and lazy loading.
- Active communities with extensive support and resources.

2.1.3 MySQL Database:

The Spring Boot REST API communicates with the MySQL server to handle user authentication and role assignment. User login information and role data are retrieved and updated in real-time through the REST API.

2.1.4 Mnesia Database:

Mnesia is an Erlang distributed database used to store advertisement and order information.

The Spring Boot REST API interacts with the Mnesia database using the jinterface, a Java library that enables communication with Erlang nodes.

Erlang nodes find each other via the EPMD daemon. Nodes based on the Jinterface library are no exception, but Jinterface doesn't start EPMD by itself. Whenever you start an Erlang node, however, it makes sure EPMD is running on the host machine. This is the simplest approach to solving this problem: start an Erlang node before you start any Jinterface-based code. (Even if that Erlang node is stopped again, EPMD will keep running on the host machine until it's killed or the machine reboots.)

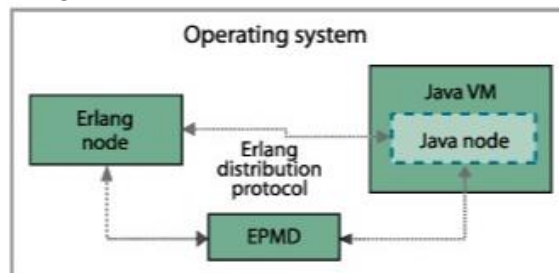


Figure 4: *Simplified Jinterface Workflow*

using the same EPMD daemon to find each other, and communicating over the Erlang distribution protocol.

Interface Implementation as follow:

```
private String request_handler(OtpErlangTuple request) throws RemoteException {
    String result = "";
    try {
        // This is a unique reference to identify a particular request.
        OtpErlangRef otpErlangRef = erlangNode.makeRef();
        // get a mail-box for the client.
        OtpMbox clientMailBox = erlangNode.getMbox();
        //
        // Send Request to Erlang Server.
        String serverMailBox = erlangServerMailBox;
        String serverName = erlangServerName;
        //
        OtpErlangObject[] content = new OtpErlangObject[3];
        content[0] = clientMailBox.self();
        content[1]=otpErlangRef;
        content[2]= request;
        OtpErlangTuple message = new OtpErlangTuple(content);
        //
        // {serverMailBox, serverName} ! {clientMailBox.self(),otpErlangRef,request}
        //where request = {insert,Name,Surname,Contestid,Tally}
        //
        clientMailBox.send(serverMailBox,serverName,message);
        // Receive response.
        Integer time_out = erlangTimeout;
        OtpErlangObject response = clientMailBox.receive(time_out);
        // close the clientMailBox
        clientMailBox.close();
        //
    }
}
```

2.1.5 Apache Web Server:

With respect the course subject, In the past, the front end and back end were combined and deployed together in the application server, with **JSP** being the primary front-end technology. However, modern trends have shifted towards using front-end frameworks like **Angular or React**, which are now deployed separately in the web server. As part of our project, we have embraced **Angular** to keep up with current technologies and implement a more modern approach to the user interface.

2.2 Architecture Flow:

2.2.1 User Interaction:

Users access the system's UI through their web browsers, where they interact with the provided functionalities. They can create advertisements for cars, view the list of available cars, place orders, and perform other actions.

2.2.2 Frontend-Backend Communication:

When users perform actions in the UI, such as submitting a login form, creating advertisements, or placing orders, HTTP requests are sent to the Spring Boot REST API. The API receives and processes these requests, performing the required operations.

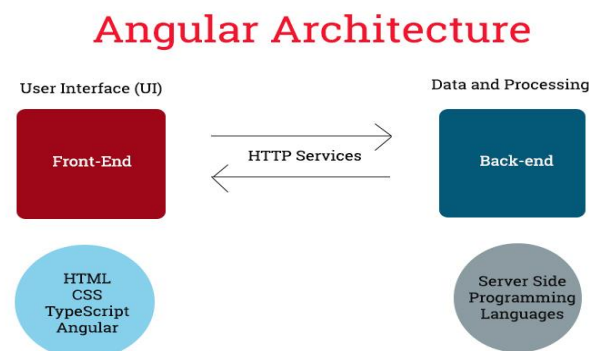


Figure 5: Simplified Architecture of Angular

2.2.3 User Authentication and Role-Based Authorization:

For login requests, the REST API communicates with the MySQL database to authenticate user credentials. Upon successful login, the API generates a JWT token, which is sent back to the UI and stored in a cookie. For subsequent requests, the UI includes the JWT token in the HTTP header to provide secure authentication. The REST API enforces role-based authorization to ensure users have access to authorized functionalities.

3. Functional Modules

3.1 Security Module

Responsible for user authentication and authorization.

The front end provides a login page where users enter their credentials.

Upon successful login, a JSON Web Token (JWT) is generated by the backend and stored in a cookie on the user's browser.

JWT is used to authenticate subsequent requests from the front end to the backend, ensuring only authenticated users can access protected resources.

Role-based authorization is enforced to control access to specific functionalities and resources.

3.2 Front-end Module:

Developed using Angular and deployed on an Apache server, serving as the user interface for the distributed system. Users interact with the front end to access various functionalities provided by the system.

3.2.1 User Management Module:

Admin users have access to this module, allowing them to create new users and assign roles. Role assignment determines the level of access users have to different functionalities within the system.

3.2.2 Advertisement Module:

Users can create advertisements for cars, which are stored in the Mnesia database. The module provides a list of advertisements, visible to all users, showcasing available cars for potential customers.

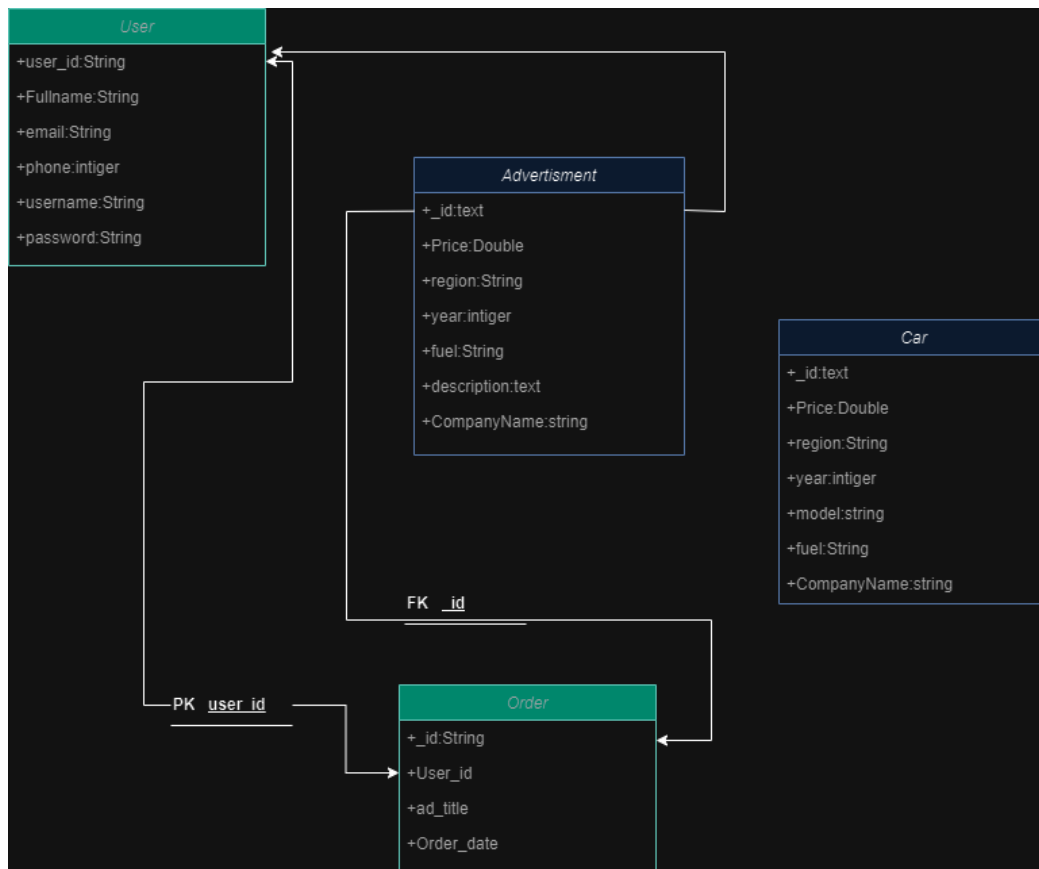
3.2.3 Car Module:

Displays a list of cars, accessible to all users. Users can view the available cars and their details.

3.2.4 Order Module:

Users can view car advertisements and place orders for their desired cars. Orders are processed and stored in the Mnesia database.

4. Data Storage:



4.1 MySQL Database:

Used for storing user information such as *usernames, passwords, and role assignments*. The Spring Boot backend interacts with the MySQL database to handle user authentication and role assignment.

4.2 Mnesia Database:

An Erlang distributed database used to *store advertisement and order information*. The Spring Boot backend communicates with the Erlang node via the jinterface to access and manage data in the Mnesia database. As we see it in system architecture.

Notice that we consider the Car table as a Global/Universal dataset for our model and storing live new data based on car table into the Advertisement table into the Mnesia DB.

5. Inter-Module Communication:

The front end communicates with the backend REST API via HTTP requests. The backend handles user authentication, authorization, and serves as an interface to the data stored in the Erlang Mnesia database. The jinterface enables communication between the Spring Boot backend and the Erlang node, facilitating seamless data exchange between the two modules.

Let's dive deeper into the explanation:

5.1. Front End and Backend Communication:

The application follows a client-server architecture, where the front end and backend components communicate with each other. The front end, which is the user interface, interacts with the backend using HTTP requests. These requests are sent to specific endpoints (URLs) exposed by the backend REST API.

5.2. Backend Responsibilities:

The backend serves as the core of the application, handling critical functionalities such as user authentication and authorization. When a user attempts to log in or access certain parts of the application, the backend verifies their identity and permissions, ensuring that only authorized users can perform specific actions.

5.3. Data Storage and Interface:

The backend also serves as an interface to interact with the data stored in the Erlang Mnesia database. The backend is responsible for managing data access, allowing the front end to request and update information stored in the Mnesia database.

5.4. Seamless Communication with jinterface:

To enable communication between the Spring Boot backend and the Erlang node (where the Mnesia database resides), the application leverages the jinterface. The jinterface is a Java library that facilitates communication between Java-based applications, like Spring Boot, and Erlang-based applications. It acts as a bridge between the two modules, enabling seamless data exchange and coordination.

5.5. Benefits of jinterface Integration:

By utilizing the jinterface, the application benefits from the strengths of both Spring Boot and Erlang. Spring Boot provides a robust and scalable backend framework, while Erlang's Mnesia database offers fast and reliable in-memory data storage. The jinterface enables the backend to communicate efficiently with the Mnesia database and utilize its capabilities to enhance data management and retrieval.

5.6. Ensuring Data Consistency:

With the jinterface facilitating data exchange between Spring Boot and Erlang, the application ensures that data consistency is maintained. When the backend receives requests from the front end to update or

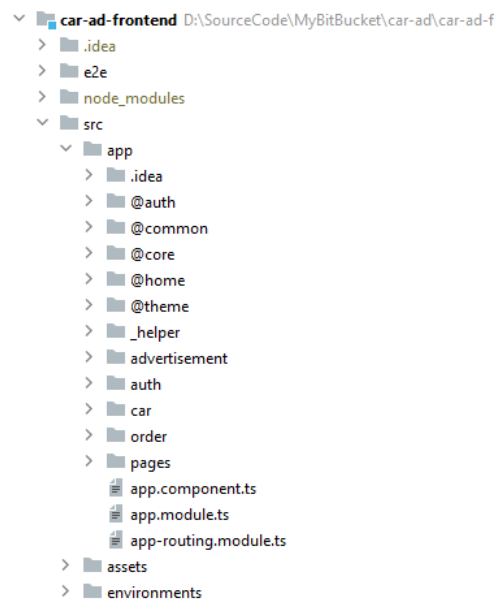
retrieve data, it interacts seamlessly with the Mnesia database via the jinterface, ensuring that data is accurate and up-to-date.

the application's architecture enables smooth communication between the front end and the backend using HTTP requests. The backend, powered by Spring Boot, handles user authentication, authorization, and acts as an interface to the Erlang Mnesia database. The jinterface serves as a crucial link between the Spring Boot backend and the Erlang node, allowing efficient and consistent data exchange, which ensures a seamless and reliable user experience.

6. Project Structure:

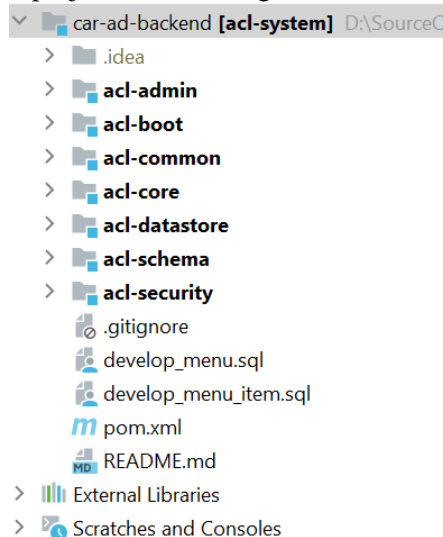
6.1 Front end project:

The front end is an angular project. All the modules have been defined in the app folder. The routing has been defined for each module.



6.2 Back end project:

The backend is a maven multi module project. All the modules have been defined in the parent pom.xml. The project is built using maven and war is generated for deployment.



7. Erlang project:

The Erlang part is one of the pillars of this project.

Integrating a Java API with an Erlang core file involves using a mechanism to enable communication and interoperability between the two languages. Typically, this is achieved through an interface that allows Java code to call functions from the Erlang core file and vice versa. There are several approaches to achieve this that we used Jinterface.

7.1 operations in the Erlang Module

7.1.1. Database Initialization:

- We've implemented the **initDB/0** function to initialize the database. It starts by creating the necessary schema using **mnesia:create_schema([node()])**. Then, we ensure Mnesia is up and running with **mnesia:start()**.
- Next, we make sure the "advertisement" and "car" tables exist by calling the **createTableAdvertisement/0** and **createTableCar/0** functions, respectively.

7.1.2. Table Creation:

- The **createTableAdvertisement/0** function is responsible for creating the "advertisement" table. We've added a try-catch block to check if the table already exists. If it does, we simply print a message indicating the table's existence. If not, we create the table with specific attributes using **mnesia:create_table/2**.
- Similar to **createTableAdvertisement/0**, the **createTableCar/0** function creates the "car" table if it doesn't exist.

7.1.3. Advertisement Functions:

- For storing advertisements, we have **storeAdvertisement/8** function. This function takes various advertisement details as input and writes a new advertisement record to the "advertisement" table within a transaction.
- To retrieve advertisements based on their ID, we use the **getAdvertisements/1** function. It queries the "advertisement" table for matching advertisements with the provided ID and returns the relevant data.
- For fetching all advertisements, we've implemented the **getAllAdvertisements/0** function. It performs a query to retrieve all advertisement records from the "advertisement" table and returns the data in a list.
- Deleting an advertisement by its ID is done through the **deleteAdvertisement/1** function. It finds the advertisement records with the provided ID and then deletes each of them from the "advertisement" table within a transaction.

7.1.4. Car Functions:

- To store car information, we use the **storeCar/13** function. It takes car details as input and writes a new car record to the "car" table within a transaction.
- For fetching all car entries, we've created the **getAllCars/0** function. It queries the "car" table for all records and returns the data in a list.

7.1.5. Additional Details:

- Throughout the code, we've utilized the QLC (Query List Comprehension) library to perform efficient queries on the Mnesia tables.
- Also, we've defined two record structures, **advertisement** and **car**, to represent the data for advertisements and cars, respectively. This approach helps to organize and access data consistently within the module.

As a team, we've successfully developed a module that provides a simple and effective interface for managing advertisements and car data in the Mnesia database using Erlang.

OtpSelf:

Represents the local Erlang node (self) and allows the creation of a connection to other nodes in the cluster. It is used to interact with the local node's functionalities.

OtpPeer:

Represents a peer Erlang node, i.e., another node to which the local node can connect. It is used to specify the target node for communication.

OtpConnection:

Represents a connection to a remote Erlang node. It provides methods for communication between the local and remote nodes, such as sending and receiving data.

OtpErlangObject:

Represents an Erlang data object that can be sent and received between nodes. It serves as a generic data container for various types of Erlang data.

OtpErlangList:

Represents an Erlang list data structure. It can contain multiple elements of various types and is often used for grouping and organizing data.

OtpErlangAtom:

Represents an Erlang atom data type, which is used for representing constants. Atoms are often used to represent symbols and unique identifiers.

OtpErlangTuple:

Represents an Erlang tuple data structure, used for grouping multiple elements into a fixed-size container. Tuples can hold different data types.

getConnection():

A method that establishes a connection to a remote Erlang node. It is used to create and initialize an OtpConnection object.

sendRPC():

A method that sends a Remote Procedure Call (RPC) to a remote Erlang node over the established connection. It allows the local node to invoke functions on the remote node.

receiveRPC():

A method that receives the response from a Remote Procedure Call (RPC) sent to a remote Erlang node. It waits for and retrieves the result of the RPC call.

convertToAdvertisementDtoList():

A utility method that converts an Erlang list (received from an RPC call) to a list of AdvertisementDto objects. It extracts data from the Erlang list and organizes it into Java objects.

convertToCarDtoList(): A utility method that converts an Erlang list (received from an RPC call) to a list of CarDto objects. Similar to the previous method, it converts Erlang data to Java objects

7.2 Core Erlang

```
-module(database_logic).
```

Defines the Erlang module named "database_logic," which contains the implementation of various functions related to database operations.

```
-include_lib("stdlib/include/qlc.hrl").
```

Includes the QLC (Query List Comprehensions) header file from the standard Erlang library. QLC is used for querying Mnesia database tables.

```
%% API
-export([initDB/0,
        storeAdvertisement/8,
        getAdvertisements/1,
        deleteAdvertisement/1,
        getAllAdvertisements/0,
        storeCar/13,
        getAllCars/0]).
```

Specifies the functions that are exported and can be accessed externally by other modules.

```
-record(advertisement,
{id,title,description,price,region,manufacturer,productionYear,username}).
```

Defines an Erlang record named "advertisement" with fields: id, title, description, price, region, manufacturer, productionYear, and username. Records are used to define structured data types.

```
-record(car, {
  id,           % ObjectId("6051ed390a85c466ff1347f4")
  region,       % "auburn"
  price,        % "35990"
  year,         % "2010"
  manufacturer, % "chevrolet"
  model,        % "corvette grand sport"
  fuel,         % "gas"
  transmission, % "other"
  type,         % "other"
  paint_color,  % null (can be represented as 'undefined' in Erlang)
  description,  % "Carvana is the safer way to buy a car..."
  posting_date, % "2020-12-02T08:11:30-0600"
  phone        % "4057759884"
}).
```

Defines another Erlang record named "car" with fields: id, region, price, year, manufacturer, model, fuel, transmission, type, paint_color, description, posting_date, and phone.

```
initDB()->
  mnesia:create_schema([node()]),
  mnesia:start(),
  createTableAdvertisement(),
  createTableCar().
```

Initializes the Mnesia database by creating and starting a schema and creating tables for advertisements and cars.

```
createTableAdvertisement()->
  try
    mnesia:table_info(type,advertisement),
    io:format("Table 'advertisement' already exists. ~n",[])
  catch
    exit:_->
      mnesia:create_table(advertisement,[{attributes, record_info(fields,
advertisement)},
      {type,bag},
      {disc_copies,[node()]}]),
      io:format("Table 'advertisement' Created. ~n",[])
  end.
```

Creates the "advertisement" table in the Mnesia database if it does not already exist.

```
createTableCar() ->
  try
    mnesia:table_info(type,car),
    io:format("Table 'car' already exists. ~n",[])
  catch
    exit:_->
      mnesia:create_table(car,[{attributes, record_info(fields, car)},
      {type,bag},
      {disc_copies,[node()]}]),
      io:format("Table 'car' Created. ~n",[])
  end.
```

Creates the "car" table in the Mnesia database if it does not already exist.

```
storeAdvertisement(Id, Title, Description, Price, Region, Manufacturer,
ProductionYear, Username) ->
  io:format("Creating Advertisement. ~n",[]),
  AF = fun() ->
    mnesia:write(#advertisement{
      id = Id,
      title = Title,
      description = Description,
      price = Price,
      region = Region,
      manufacturer = Manufacturer,
      productionYear = ProductionYear,
      username = Username
    })
  end,
  mnesia:transaction(AF).
```

Stores an advertisement in the "advertisement" table by writing data into the Mnesia database.

```
getAdvertisements(Id) ->
  AF = fun()->
    Query = qlc:q([X || X <- mnesia:table(advertisement),
```

```

    X#advertisement.id == Id]),
Results = qlc:e(Query),
lists:map(fun(Item)-> {Item#advertisement.id,
                        Item#advertisement.title,
                        Item#advertisement.description,
                        Item#advertisement.price,
                        Item#advertisement.region,
                        Item#advertisement.manufacturer,
                        Item#advertisement.productionYear,
                        Item#advertisement.username} end, Results)

end,
{atomic,Advertisements} = mnesia:transaction(AF),
Advertisements.

```

Retrieves advertisements from the "advertisement" table based on the given Id.

```

getAllAdvertisements() ->
AF = fun()->
    Query = qlc:q([X || X <- mnesia:table(advertisement)]),
    Results = qlc:e(Query),
    lists:map(fun(Item)-> {Item#advertisement.id,
                            Item#advertisement.title,
                            Item#advertisement.description,
                            Item#advertisement.price,
                            Item#advertisement.region,
                            Item#advertisement.manufacturer,
                            Item#advertisement.productionYear,
                            Item#advertisement.username} end, Results)
    end,
{atomic,Advertisements} = mnesia:transaction(AF),
Advertisements.

```

Retrieves all advertisements from the "advertisement" table.

```

deleteAdvertisement(Id) ->
AF = fun()->
    Query = qlc:q([X || X <- mnesia:table(advertisement),
                    X#advertisement.id == Id]),
    Results = qlc:e(Query),

    F = fun() ->
        lists:foreach(fun(Result) ->
                        mnesia:delete_object(Result)
                        end, Results)
        end,
    mnesia:transaction(F)

    end,
mnesia:transaction(AF).

```

Deletes an advertisement from the "advertisement" table based on the given Id.

```

storeCar(Id, Region, Price, Year, Manufacturer, Model, Fuel, Transmission,
Type,
    PaintColor, Description, PostingDate, Phone) ->
AF = fun() ->
    mnesia:write(#car{
        id=Id,
        region=Region,
        price=Price,
        year=Year,

```

```

        manufacturer=Manufacturer,
        model=Model,
        fuel=Fuel,
        transmission=Transmission,
        type=Type,
        paint_color=PaintColor,
        description=Description,
        posting_date=PostingDate,
        phone=Phone
    })
    end,
    mnesia:transaction(AF).

```

Stores car data in the "car" table by writing data into the Mnesia database.

```

getAllCars() ->
AF = fun() ->
    Query = qlc:q([X || X <- mnesia:table(car)]),
    Results = qlc:e(Query),
    lists:map(fun(Item) -> {Item#car.id,
        Item#car.region,
        Item#car.price,
        Item#car.year,
        Item#car.manufacturer,
        Item#car.model,
        Item#car.fuel,
        Item#car.transmission,
        Item#car.type,
        Item#car.paint_color,
        Item#car.description,
        Item#car.posting_date,
        Item#car.phone} end, Results)
    end,
    {atomic, Cars} = mnesia:transaction(AF),
    Cars.

```

Retrieves all car data from the "car" table.

8. Deployment and Configurations

The application components to be deployed are:

- Tomcat application server
- Erlang Server and the Mnesia DB
- MySQL DB server
- The Apache Web Server

The components and sub components are distributed and replicated among the available machines. The current configuration uses 3 remote Ubuntu machines, available at the following addresses:

- container1: 10.2.1.29
- container2: 10.2.1.30
- container4: 10.2.1.54

The main components of the project are deployed as the configuration schema depicted in table [1](#).

Component	IP
Tomcat application server	10.2.1.29
Erlang Server	10.2.1.54
MySQL DB Server	10.2.1.29
The Apache Web Server	10.2.1.30

Table 1: Components and their deployment containers.

9. Conclusion:

The distributed system provides a seamless and secure user experience, allowing users to manage advertisements, cars, and orders. The modular architecture, JWT-based authentication, and role-based authorization ensure controlled access and efficient handling of data across the system's different components. The combination of Angular, Spring Boot, MySQL, and Erlang Mnesia provides a powerful and robust foundation for the successful operation of the distributed application.

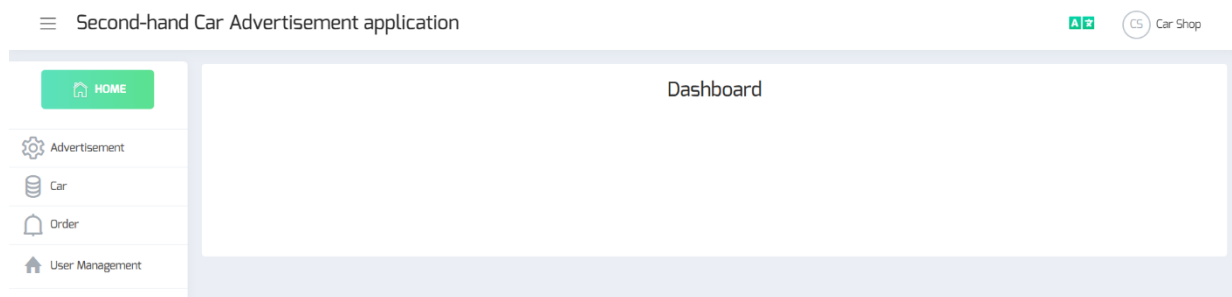
10. User Manual

In this subsection we will present the overview of how our system works

Login

[Forgot Password?](#)

To have access to the application , use this username [admin@shop.com](#) and this password: s3cret you will go directory to the admin panel as follow:



At the left you can see the menu has access the different modules such as Advertisement car and orders and User Management

at the advertisement menu, you can see the *active advertisements* and from the left, you can choose the *create* a new advertisement or *list them* and by *pace Order* you will have accesses to all the listed advertisements to do the place ordering and accept the order.

Second-hand Car Advertisement application

CS Car Shop



















HOME

Create

List

Place Order

List of Advertisement

Actions	ID	Title	Description	Price	Region	Manufacturer	Production Year
	ID	Title	Description	Price	Region	Manufacturer	Production Year
 	\1c01bce36-9a06-4687-9066-477279d193be\	testad4edined	testdes4ed	\12345\	region4	manu4	\2023\
 	\133a3896-ab56-4228-b15c-dfed9bdeca0f\	testad111edited	testad11description	\2222\	testregion111	manu111	\2023\
 	\d464fec0-5f8e-46b1-90f-efa01d8e1823\	\Ad-23\ updated	\ad-23-description\	\1111.0\	\test region\	\Test manu\	\2121\
 	\92e22063-fccc-4934-9d0d-b6082cbd15e2\	\for sale\	\good car\	\2500\	pisa	\BMW\	\2012\
 	\959b2b25-36a0-416f-bba5-43ce3635188b\	testad3	testdes3ed	\1234\	regin3	manu3	\2023\
 	\7e66f292-0d13-4a4e-9e5e-746e13a612dc\	'test ad 15 edited'	\test ad 15 des\	\1111\	\test region 15\	\Test manu 15\	\2023\
 	\8f410ee7-5484-47a9-a0f4-2460a883d6bc\	\ad-test2\ updated	\ad-test2-des\	\2222.0\	\test region\	\Test manu\	\2121\
 	\d6dcf687-d11d-45ee-81a7-7107ee2e31ac\	\good car\ updated	\really nice\	\2500.0\	pisa	fiat	\2005\
 	\7a5540a8-ac0f-47c8-92b6-afa07942a09d\	badcar	relaynice	\50000\	pisa	fiat	\2006\

From the crate menu you will see some fields which have to be completed to create a new advertisement and this data will save on the Mnesia Database.

Second-hand Car Advertisement application

HOME

Create

List

Advertisement

Title *

Enter Title

Description *

Enter Description

Price *

Enter Price

Region *

Enter Region

Manufacturer *

Enter Manufacturer

Production Year *

Enter Production Year

User *

Select user

SAVE



















And from the car menu, illustrated the list of the cars, in our case, we put around **100 records** in the Mnesia DB for just testing purposes and this data is Universal Data

Second-hand Car Advertisement application

HOME

List

List of Car

Actions	ID	Region	Price	Year	Manufacturer	Model	Fuel
	ID	Region	Price	Year	Manufacturer	Model	Fuel
 	'6051ed390a85c466f1134807'	auburn	'47000'	'2020'	jeep	gladiator	gas
 	'6051ed390a85c466f1134847'	auburn	'13450'	'2013'	toyota	camry	gas
 	'6051ed390a85c466f1134806'	auburn	'32990'	'2019'	ford	'1150 supercrew cab xlt'	gas
 	'6051ed390a85c466f1134846'	auburn	'8500'	'2012'	chevrolet	'equinox lt'	gas
 	'6051ed390a85c466f113481f'	auburn	'36990'	'2010'	chevrolet	'corvette grand sport'	gas
 	'6051ed390a85c466f1134805'	auburn	'2650'	'1996'	toyota	'1100 4x4'	gas
 	'6051ed390a85c466f1134845'	auburn	'29990'	'2014'	chevrolet	'silverado 1500 crew'	gas
 	'6051ed390a85c466f113481e'	auburn	'0'	'2014'	ram	other	diesel
 	'6051ed390a85c466f1134804'	auburn	'33990'	'2012'	chevrolet	'corvette grand sport'	gas



















Here as you can see we have some actions on the car's menu, edit and Delete order action, from the edit admin can edit the advertisement and or Delete it.

Second-hand Car Advertisement application

HOME

List

List of Car

Actions	ID	Region	Price	Year	Manufacturer	Model	Fuel
	ID	Region	Price	Year	Manufacturer	Model	Fuel
 	'6051ed390a85c466ff134807'	auburn	'47000'	'2020'	jeep	gladiator	gas
 	'6051ed390a85c466ff134847'	auburn	'13450'	'2013'	toyota	camry	gas
 	'6051ed390a85c466ff134806'	auburn	'32990'	'2019'	ford	'f150 supercrew cab xlt	gas
 	'6051ed390a85c466ff134846'	auburn	'8500'	'2012'	chevrolet	'equinox lt'	gas
 	'6051ed390a85c466ff13481f'	auburn	'36990'	'2010'	chevrolet	'corvette grand sport'	gas
 	'6051ed390a85c466ff134805'	auburn	'2650'	'1996'	toyota	't100 4x4'	gas
 	'6051ed390a85c466ff134845'	auburn	'29990'	'2014'	chevrolet	'silverado 1500 crew'	gas
 	'6051ed390a85c466ff13481e'	auburn	'0'	'2014'	ram	other	diesel
 	'6051ed390a85c466ff134804'	auburn	'33990'	'2012'	chevrolet	'corvette grand sport'	gas

from the order action in the advertisemets menu , if you chose some advert as place an order , from the main menu , in *Order* section you will see the placed orders here

Second-hand Car Advertisement application

HOME

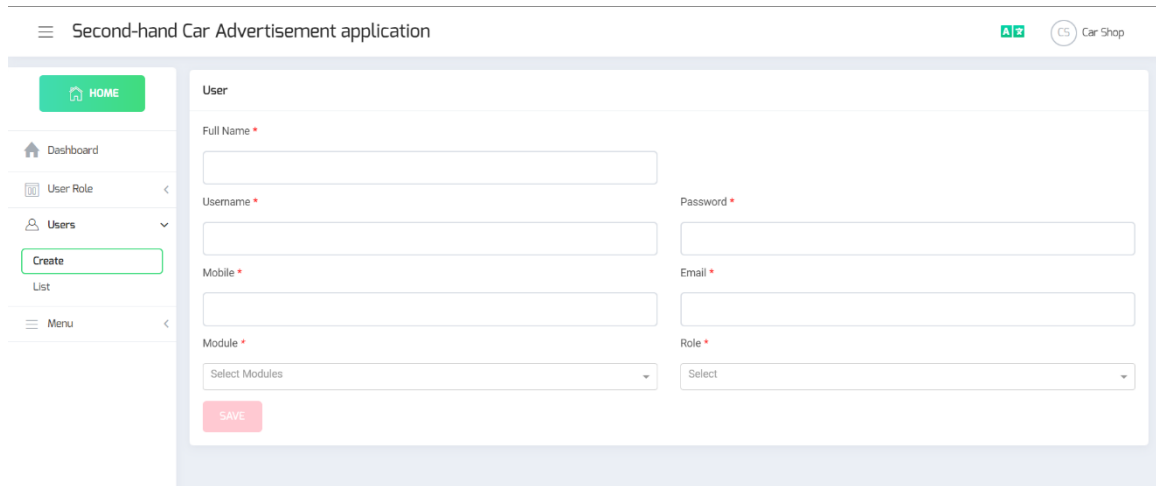
List

List of Orders

ID	Ad. Title	Order Date	User	Status
ID	Ad. Title	Order Date	User	Status
1	'ad-test3'	23-Jul-2023	admin	PENDING
2	'Test ad 5'	23-Jul-2023	admin	PENDING
3	'ad-test3'	23-Jul-2023	admin	PENDING
4	'ad-test2'	23-Jul-2023	camillo	PENDING
5	'good car'	24-Jul-2023	admin	PENDING

user management menu

And from the user management menu, we can create a new user also we are able to choose, which module and what role this new user would have.



Second-hand Car Advertisement application

HOME

Dashboard

User Role

Users

Create

List

Menu

User

Full Name *

Username *

Password *

Mobile *

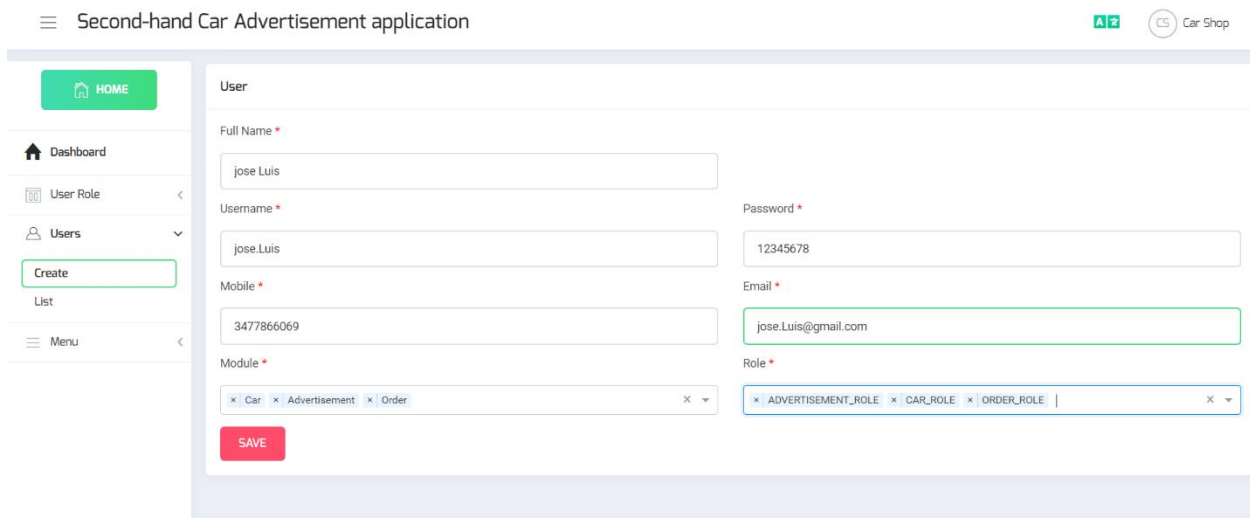
Email *

Module *

Role *

SAVE

For example:



Second-hand Car Advertisement application

HOME

Dashboard

User Role

Users

Create

List

Menu

User

Full Name *

Username *

Password *

Mobile *

Email *

Module *

Role *

SAVE

and with this user, you can log in to the system and do some actions with the application.

And finally, you can see the users from the list:

Second-hand Car Advertisement application

CS Car Shop

HOME

Dashboard

User Role

Users

Create

List

Menu

List of Users

Username	Full Name	Mobile	Email	Roles	Modules
Username	Full Name	Mobile	Email	Roles	Modules
admin	Car Shop	12345678	admin@shop.com	ADMIN	Advertisement, Car, Order, User Management
camillo	Camillo Cavalli	01811483577	camillo@gmail.com	ADVERTISEMENT_ROLE, CAR_ROLE, ORDER_ROLE	Advertisement, Car, Order
jose.Luis	Jose Luis	3477866069	jose.Luis@gmail.com	ADVERTISEMENT_ROLE, CAR_ROLE, ORDER_ROLE	Advertisement, Car, Order
k.mehravar	kamran Mehravar	3477866069	k.mehravar@studenti.unipi.it	ADVERTISEMENT_ROLE, CAR_ROLE, ORDER_ROLE	Advertisement, Car, Order
mozahid	Mozahidul Islam	01811483577	mozahidone@gmail.com	ADVERTISEMENT_ROLE, CAR_ROLE, ORDER_ROLE	Advertisement, Car, Order
s.mahmud	Sultan mahmud	3348957796	s.mahmid@studenti.unipi.it	ADVERTISEMENT_ROLE, CAR_ROLE, ORDER_ROLE	Advertisement, Car, Order

we have to note that, the user information will be stored in SQL database.

11.Functionality Test and System Engineering

11.1 Tomcat status

First of all we have to check the status of Tomcat server to be sure it is running , for this , we have to logging in to the proper server and for our case the Host IP address is **10.2.1.29** and we have to run this command on the terminal to insure the Tomcat status is on running stage.

command: `# ps fax | grep tomcat`

```
root@Distributed2022:~/servers/apache-tomcat-8.5.91/bin# ps fax | grep tomcat
15007 pts/0    S+   0:00 | \_ grep --color=auto tomcat
3317 ?        Sl   212:40 /root/.sdkman/candidates/java/current/bin/java -Djava.util.logging.config.file=/root/servers/apache-tomcat-8.5.91/conf/logging.properties -Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager -Djdk.tls.ephemeralDHKeySize=2048 -Djava.protocol.handler.pkgs=org.apache.catalina.webresources -Dorg.apache.catalina.security.SecurityListener.Umask=0027 -Dignore.endorsed.dirs=-classpath /root/servers/apache-tomcat-8.5.91/bin/bootstrap.jar:/root/servers/apache-tomcat-8.5.91/bin/tomcat-juli.jar -Dcatalina.base=/root/servers/apache-tomcat-8.5.91 -Dcatalina.home=/root/servers/apache-tomcat-8.5.91 -Djava.io.tmpdir=/root/servers/apache-tomcat-8.5.91/temp org.apache.catalina.startup.Bootstrap start
```

11.2 Apache Web server

Notice that our project is separated into the back-end and Front-end infrastructure As we mentioned before in section 2, we have to be sure that our front-end server is also running, to achieve this manner, we have to prompt this command in the proper server which is **10.2.1.30** .

```
root@Distributed2022:~# systemctl status apache2
● apache2.service - The Apache HTTP Server
   Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2023-08-02 10:10:41 UTC; 1 month 8 days ago
     Docs: https://httpd.apache.org/docs/2.4/
   Process: 1775 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)
    Main PID: 1791 (apache2)
      Tasks: 55 (limit: 2388)
     Memory: 57.7M
        CPU: 16min 23.150s
    CGroup: /system.slice/apache2.service
            └─1791 /usr/sbin/apache2 -k start
              └─1792 /usr/sbin/apache2 -k start
                └─1793 /usr/sbin/apache2 -k start
```

11.3 Erlang System Management

To be sure that, our Erlang server and Mnesia DB is in running state and that our data can be retrieved by Mnesia DB after logging into the application via the browser we have to do this command into the Erlang Server with this IP address: 10.2.1.54

```
root@Distributed2022:~# cd ..
root@Distributed2022:/# cd opt/erlang/
root@Distributed2022:/opt/erlang# erl -name mnesia@localhost -setcookie mnesiacookie -dir .
Erlang/OTP 24 [erts-12.2.1] [source] [64-bit] [smp:1:1] [ds:1:1:10] [async-threads:1] [jit]

Eshell V12.2.1 (abort with ^G)
(mnesia@localhost)1> mnesia:start().
ok
```

And into the another command window we have to prompt this command too:

```
root@Distributed2022:~# cd ..
root@Distributed2022:/# cd opt/erlang/
root@Distributed2022:/opt/erlang# java -jar caradapi.jar
```

After running the last command into the server the result would be like this picture and we can use the Erlang functionality as we expected.

```

  ____ _
 / ___ \
/ /   _ \
/ /___) \
/_____/

:: Spring Boot ::                (v2.1.2.RELEASE)

2023-09-10 19:30:18.031 INFO 35639 --- [          main] com.carad.api.ApiApplication
: Starting ApiApplication v2.1.2.RELEASE on Distributed2022 with PID 35639 (/opt/erlang/caradapi.jar
r started by root in /opt/erlang)
2023-09-10 19:30:18.045 INFO 35639 --- [          main] com.carad.api.ApiApplication
: No active profile set, falling back to default profiles: default
2023-09-10 19:30:21.553 INFO 35639 --- [          main] o.s.b.w.embedded.tomcat.TomcatWebServer
: Tomcat initialized with port(s): 8081 (http)
2023-09-10 19:30:21.660 INFO 35639 --- [          main] o.apache.catalina.core.StandardService
: Starting service [Tomcat]
2023-09-10 19:30:21.669 INFO 35639 --- [          main] org.apache.catalina.core.StandardEngine
: Starting Servlet engine: [Apache Tomcat/9.0.14]
2023-09-10 19:30:21.693 INFO 35639 --- [          main] o.a.catalina.core.AprLifecycleListener
```

11.4 Erlang functionality test

In this part we are going to test our Erlang functionality to be insure that our program is working as we desire. In the user manual part, we have seen that, we can retrieve our data from Mnesia DB which can be illustrated in the application via Browser, this picture proves that, whenever we send a request from the application to Erlang, for example, the advertisement action from the menu in the application which is the area we can see all of our advertisements that we created and doing edit/delete on the one of the advertisements, erlang server prompts the exact function in which have to do for that request.

```
2023-09-10 22:38:55.988 INFO 62698 --- [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2023-09-10 22:38:56.003 INFO 62698 --- [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 14 ms
2023-09-10 22:38:56.058 INFO 62698 --- [nio-8081-exec-1] com.carad.api.ctrl.RestApiController : Returning All Avertisements >>>>>>
{atomic,ok}
2023-09-10 22:40:24.707 INFO 62698 --- [nio-8081-exec-4] com.carad.api.ctrl.RestApiController : Returning All Avertisements >>>>>>
2023-09-10 22:40:33.539 INFO 62698 --- [nio-8081-exec-5] com.carad.api.service.ErlangService : Service:Deleting Advertisement with Id:9ae37802-7e72-4404-8364-cc75fef428b5
{atomic,ok}
2023-09-10 22:40:33.665 INFO 62698 --- [nio-8081-exec-6] com.carad.api.ctrl.RestApiController : Returning All Avertisements >>>>>>
```

11.5 My SQL functionality test

As we have mentioned before, our user data will be stored in the My SQL Database, To ensure that, our data is available and persists well, we can log in to the proper server which is **10.2.1.29**.

From the picture below we have selected the proper table of my SQL and you can see the user data persists correctly.

To see the data, after logging into the server you have to run this command in advance:

```
#mysql
```

```
#SHOW DATABASES;
```

```
#use car_advertisement
```

```
#SHOW TABLES;
```

```
#SELECT * FROM users;
```

```
mysql> SELECT * FROM users;
```

user_id	created_at	email	created_by	last_modified_by	last_modified_at	account_non_expired	mobile	password	account_non_locked	credentials_non_expired
admin	2023-07-23 12:36:03	admin@shop.com	SYSTEM	SYSTEM	2023-07-23 12:36:03	0x01	12345678	\$2a\$10\$ceqJKTxTYFmhHu6cTZL.4uqBb0KmjNGV20qGZFfi0BcxpHtomVye	0x01	0x01
camillo	2023-07-23 18:23:36	camillo@gmail.com	admin	admin	2023-07-23 18:23:36	0x01	01811483577	\$2a\$10\$RtbJCpkq18CK4TRLvA..je.ApMra/doktgeIezNBFseVLgyjo9iAe	0x01	0x01
jose.Luis	2023-07-24 08:32:47	jose.Luis@gmail.com	admin	admin	2023-07-24 08:32:47	0x01	3477866069	\$2a\$10\$pvigPQU2226qpl/fm2JKieigh9M.eLS.qIAI0Z3/jSWVB.RmpHaJK	0x01	0x01
k.mehravar	2023-07-23 22:37:31	k.mehravar@studenti.unipi.it	admin	admin	2023-07-23 22:37:31	0x01	3477866069	\$2a\$10\$vaY10oz34pSYNB/Z0P0SBUlvtHcD07zk7VlvZ5db8j51k48CKIu10	0x01	0x01
mozahid	2023-07-23 19:28:55	mozahidone@gmail.com	admin	admin	2023-07-23 19:28:55	0x01	01811483577	\$2a\$10\$umz6zc.NULY8hX0hzr8V1ucAIwu9q/nIT69EUS3LPKQM23SFuXui	0x01	0x01
s.mahmud	2023-07-23 22:39:18	s.mahmid@studenti.unipi.it	admin	admin	2023-07-23 22:39:18	0x01	3348957796	\$2a\$10\$GsvcnrocP1VD0W0.Eyt9NeoyfMv..Wu935Wv7otpgA0bcUKGaib.i	0x01	0x01

6 rows in set (0.00 sec)