UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Web Applications A.Y. 2023-2024

Homework 1 – Server-side Design and Development

Master Degree in Computer Engineering
Master Degree in Cybersecurity
Master Degree in ICT for Internet and Multimedia

Deadline: 29 April, 2024

Group Acronym	WACAR	
Last Name	**First Name**	**Badge Number**
Di Martino	Ludovico	2104292
Galli	Filippo	2120826
Leonardi	Alessandro	2112308
Rigobello	Manuel	2103374
Scapinello	Michele	2087617

1 Objectives

The goal of this project is to develop WaCar, a web application designed to facilitate the booking process for car racing sessions across different tracks in Italy. WaCar targets both racing enthusiasts and casual users looking for a exciting track experience. Its primary objective is to provide users with a smooth experience in choosing their preferred car, track, number of laps, and booking time slots, At the same time, it provides a comprehensive view of all available cars and circuits, accompanied by their detailed descriptions. The platform also offers administrators the ability to efficiently manage tracks, cars and pricing.

2 Main Functionalities

WaCar is an intuitive online platform catering to two primary groups: users and administrators. Upon entering on WaCar's homepage, users are greeted with an array of features and options designed to streamline their experience. Firstly, users can see a comprehensive list of available cars and circuits, each accompanied by detailed descriptions to help them in their selection process. At the same time they are also presented with the option to either login or register on the platform, providing essential details like name, surname, address, email, and password in order to access to the platform's full set of features.

Once registered, users can seamlessly proceed to place orders through a dedicated subsection of the application. Here, users can select their desired car from various categories, ranging from supercars to off-road vehicles, with each category offering a diverse range of brands and models. Following car selection, users then choose a compatible circuit and specify the number of laps for their session.

The next step involves selecting a suitable time slot from the available options. Once a preferred time slot is chosen, users are presented with a preview of their order, affording them the opportunity to modify the booking as desired. Additionally, users have the capability to manage their orders, including accessing a detailed record of past orders for reference.

Furthermore, users can mark favorite combinations of circuits and cars for an easier access in the future, ensuring their preferred selections to be accessible during the next visits to the platform. On the administrative side, administrators oversee the management of tracks and cars, with responsibilities including adding, modifying and making them available or not as needed. Administrators are also able to adjust pricing for laps and to associate specific cars with particular circuits, determining which cars can operate on which circuits.

3 Presentation Logic Layer

As previously mentioned, WaCar web application is subdivided into two main areas: the admin area and the registered user area. The areas are accessible through the login page, where, by inserting a valid combination of email and password, people can access as an admin or a user, depending on the role associated with the email in the database. Below are listed the pages that are going to be developed in WaCar for the users.

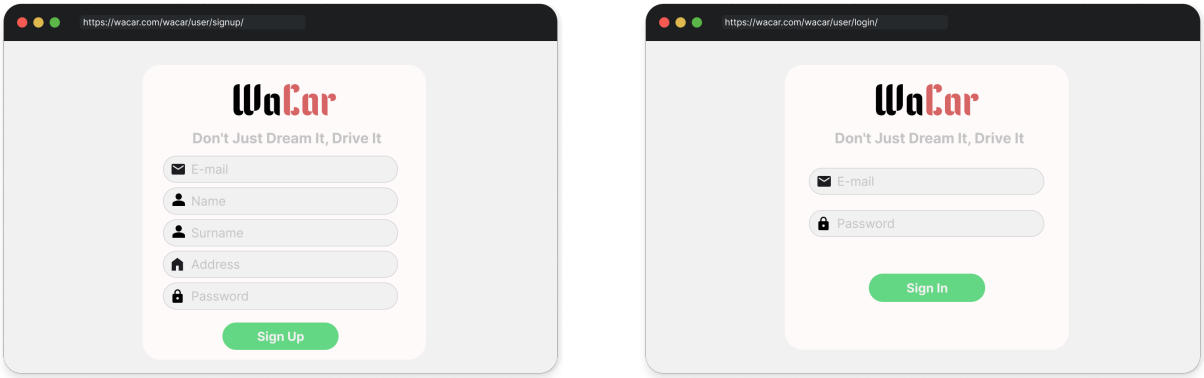
- Homepage: the initial page presented when connecting to the website. On this page login is not needed and not registered user can see the list or cars and circuit or eventually sign up to the website via the corrispondent page;
- Login: the page is used by registered admin/users to login
- Sign up: the page is used by new users to sign up to WaCar website;
- Car List: the page where the available brand and model of cars are listed;
- Circuit List: the page to see the circuits where one can race;
- Order List: the page is available only to logged users and allows them to check the order history and manage them;
- Create new order: the page is available to logged users and allows them to create a new order;

- List Favorite: the page is available only to logged users and allows them to check their favorite order list;
- User Page: the page is available to logged users and allows them to see their informations;

As it concerns the admin, the following pages have been developed:

- Insert Car: in this page are present all the needed fields to create the car: model, brand, description, max speed, horsepower, acceleration, availability, image and type.
- Insert Circuit: in this page are present all the needed fields to create the circuit: name, address, description, length, number of corners, lap price, availability, image and type.
- Insert Car Type: the page can be used to create a new car type.
- Insert Circuit Type: the page can be used to create a new circuit type.
- Car-Circuit Suitability: the page can be used to create or delete a matching between a car or a circuit.
- Edit Car: the page used to edit an existing car. Here you can for instance modify the description of the car.
- Edit Circuit: the page used to edit an existing circuit. Here you can for instance modify the address information of the circuit.

3.1 Sign Up and Login page



(a) Sign Up page.

(b) Login page.

In the `Sign Up` page, Fig. 1a, users input their information to create a new account and access to new functionalities. The required information for the registration are email, password, name, surname, and address. The password must meet certain criteria including a minimum of 8 characters, at least one number, and at least one uppercase letter. Upon clicking "Sign up" the user's credentials are added to the database and users are redirected to the homepage. If the provided credentials do not meet the required criteria an error message is displayed to the user. The login process mirrors the sign up process, but only requires the user (or the admin) to input their email and password and, upon clicking "Login", a search is conducted in the database Fig. 1b. If the provided email exists and the user's password matches the encrypted password stored in the database, the login operation is executed, otherwise the user (admin) is informed that email or password are incorrect. During the login operation, it is checked whether the account belongs to an admin or a regular user, and consequently, different pages will be

served depending on the role. It's worth mentioning that the sign-up page is exclusively for regular users, whereas administrators are added to the system by a superuser who directly manages the database. This distinction is made because the creation of administrator accounts is infrequent and typically occurs during the system's deployment phase.

3.2 Homepage

(a) User Homepage.

(b) Admin Homepage.

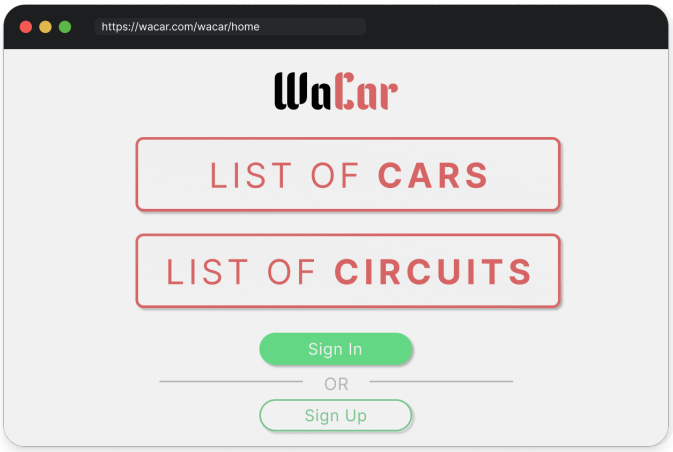


Figure 3: Homepage without logging.

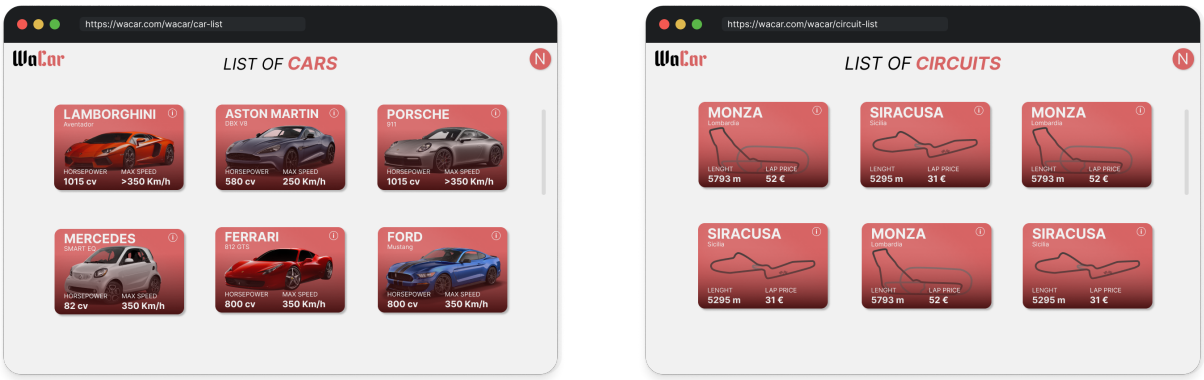
The homepage serves as the initial landing point for users upon entering WaCar. Its content dynamically adapts to the session state: when user is not logged in, Fig. 3, they are able to access two basic options that are `List of Cars` and `List of Circuits`. Additionally, `Login` and `Sign Up` buttons are provided.

If the user is logged in, in addition to the previously mentioned functionalities, `List of Orders` option and `Favourite` option are added, Fig. 2a. The latter allows the user to manage its favourite objects (car, circuit and

even pending orders added to favourite) while through `List of Orders` the user is able to view his orders history and manage the booked ones.

If an admin is logged in different functionalities are added to its home page as shown in Fig. 2b. The new features are `Add New Car`, `Add New Circuit` and `Map Car Type to Circuit Type` and allows the admin to manage such components according to the possible modification needed.

3.3 Car list and Circuit list



(a) Car list page.

(b) Circuit list page.

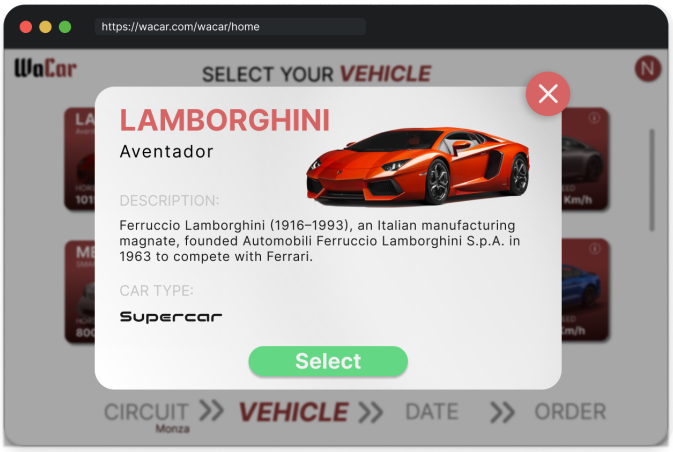


Figure 5: Car information pop up.

Car list, Fig. 4a, and Circuit list, Fig. 4b, pages are two basic pages that allows the user to visualize the possible cars and circuits they can select for their experience. Upon clicking over a car a series of detailed information of the latter are visualized in order to allow the user to better understand the car characteristics as shown in Fig. 5. Analogously, an identical functionality is implemented also in the Circuit list page.

3.4 Create order and Order recap page

The create order page displayed in Fig. 6 is a one page version for the process of creating an order and a different version may be implemented for the final version of WaCar. The process of creating an order consists in:

- selecting a car between the available ones;
- selecting an available circuit in which the selected car is suitable to race;
- selecting an available date and time for the user experience;
- selecting the number of laps that the user is intended to race;

After the latter steps the page will display the price for the experience selected by the user. Finally, if the user decides to not process the order immediately, he/she can save the latter in the favourites and recover it later. A possible idea to implement could be a mechanism that gives to the user the possibility to print the reservation.

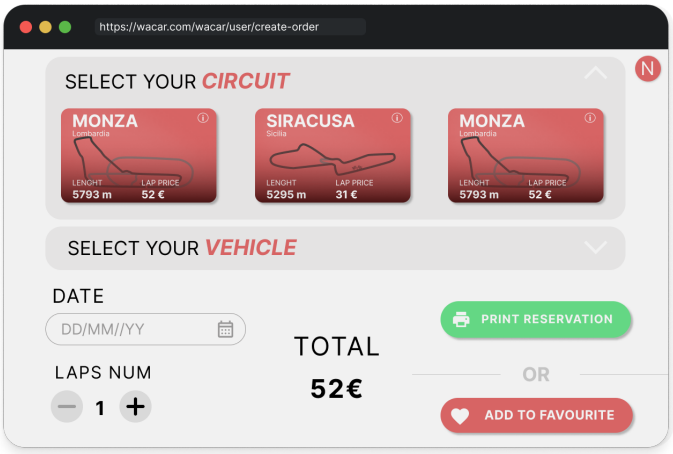


Figure 6: Order all in one page

4 Data Logic Layer

4.1 Entity-Relationship Schema

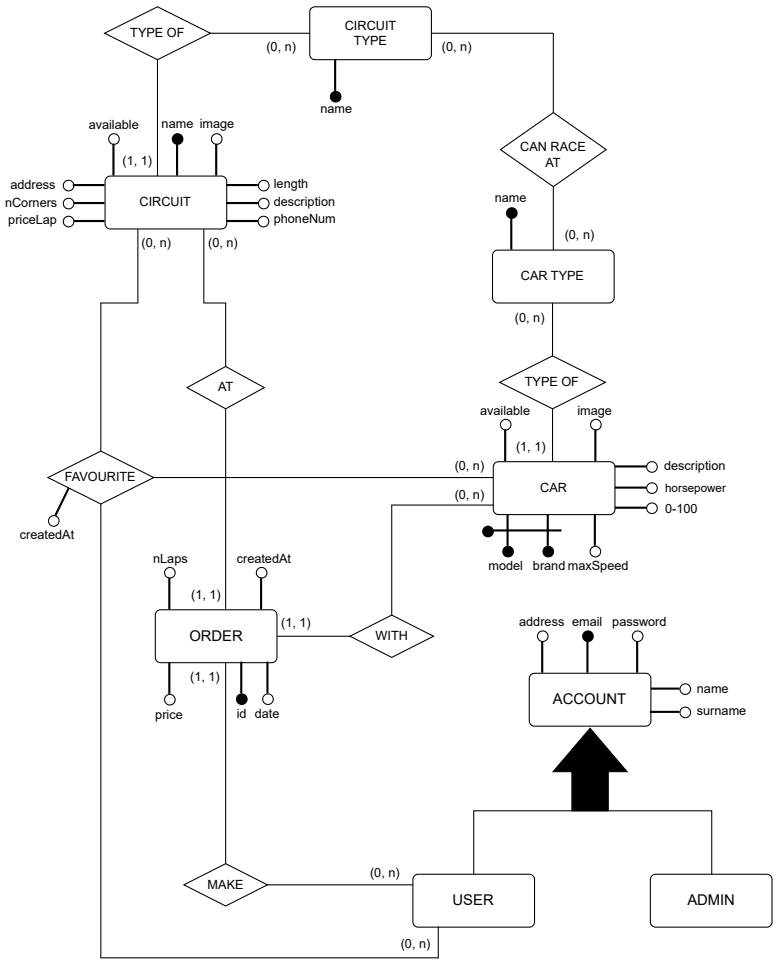


Figure 7: ER schema of WACAR

The ER schema contains the following entities:

- **account**:the account contains both the account for the users of the web app and the admin that manages the contents of the tables. It has an email (type TEXT) and a password (type TEXT) that it uses for signing in. Then, it has the name (type CHARACTER VARYING), the surname (type CHARACTER VARYING) and the address (type TEXT). Finally, it has the attribute type that is an enumerator and it can assume value "USER" or "ADMIN" which specifies the type of the account. An account have a 0-N relationship with entity *order*, so that it can make more than one order. Finally, an account can also add a pair *car-circuit* in a bookmarks list represented by the relationship *favourite*.

- **car**: it represents the entity for the car users car drive. Its primary key is composed by the brand and the model of a car (both type CHARACTER VARYING), some data are about the performance of the car such as the horsepower (type INTEGER), the 0-100 acceleration (type NUMERIC) and the maxSpeed (type INTEGER).

For more general information there is the description (type TEXT). Finally, it has a foreign key for the type of the car (e.g. 'supercar'), an image (type BYTEA) and an attribute available (type BOOLEAN) that defines if the car is available for new orders.

- **circuit**: this entity represents the circuit in which a user can race. It is identified by its name (type CHARACTER VARYING). Then, it contains the address (type TEXT), the price for doing a lap (type INTEGER) and some characteristics such as the length (type INTEGER), the number of corners (type INTEGER) and a general description (type TEXT) with also an image (type BYTEA). Finally, it has a foreign key for the type of circuit (e.g 'tarmac') and, as in car entity, it has the attribute available (type BOOLEAN).

- **circuitType** and **carType**: the first entity contains the name (type CHARACTER VARYING) of the type of a circuit, while the second entity contains the name (type CHARACTER VARYING) of the type of a car. Those entities are related to each other by the 0-N *can race at* relationship. This relationship saves the suitability between a type of circuit and a type of car: for example, if a pair circuitType-carType is composed by 'tarmac'-'supercar', then this means that a supercar can race on all the circuits of that type. Otherwise, if for example the pair 'rallycross'-'supercar' is not present, then the supercar cannot race on this type of circuit.

- **order**: its primary key is an id (type INTEGER). It contains the email of the *account* that has made the order and the date (type DATE) in which the user will have the experience. It also contains lapsNumber (type INTEGER), that is the number of laps for this booked session, and the price (type INTEGER) of the order. Then, to specify in which circuit and with which car the user wants to prenote, the order has the reference to the entity *circuit* and to the entity *car*.

4.2 Other Information

An account that has type "USER" can perform INSERT, UPDATE AND DELETE of an order or a favourite item via the interface. It can also perform the UPDATE on the account table to change the values of its attributes.

An account of type "ADMIN" can do the INSERT and the UPDATE on circuit and car tables. Additionally, it can do the same operation in carType and circuitType. Finally, it can INSERT and DELETE a new mapping between a carType and a circuitType.

The insertion operation in order table is protected by a function that is triggered before the insert command. The function checks if the circuit and the car associated with the new order are in a consistent status with respect to the suitability between a type of car and a type of circuit. This is performed by checking whether a row exists in carCircuitSuitability table which refers to the selected types. If the function returns a positive result, then the insertion is performed, otherwise an error will be thrown. In this function it is also checked if the price corresponds to the product of the number of laps and the price for a single lap.

There are some other controls in car and circuits tables to prevent that the user inserts some non valid values (e.g a negative horsepower in car or a negative length in circuit). It also checks if the order price corresponds to the product between the cost of a single lap and the number of laps.

An user can make multiple orders on the same date, but the function checks if all those orders contains the same circuit. This is designed to avoid possible logistical problems in the situation where the user books two circuits that are far apart.

5 Business Logic Layer

5.1 Class Diagram

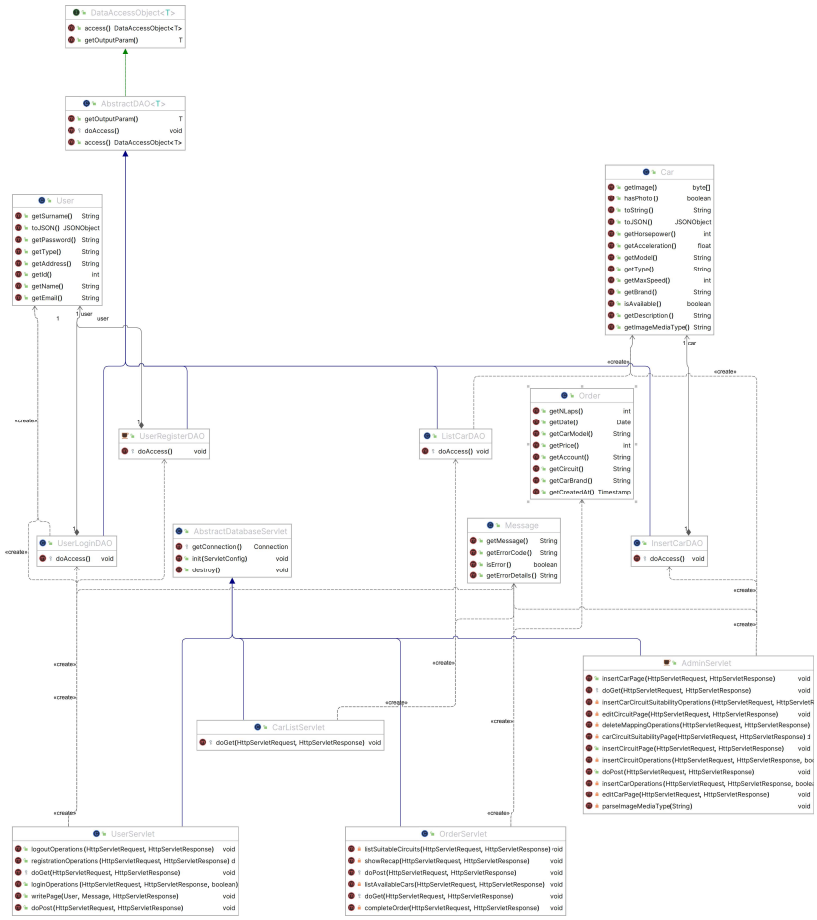


Figure 8: Class diagram of WACAR

The class diagram includes some of the classes utilized in WaCar. Each servlet extends the `AbstractDatabaseServlet`, necessary for acquiring the database connection. Four servlets are delineated:

UserServlet: Handles user operations such as login, registration, and logout. It implements the `doGet` and

doPost methods to manage HTTP requests. Not all URIs are freely accessible: while login and registration are, accessing user information requires authentication; otherwise, users are redirected to the login page using the corresponding filter (this is also valid for OrderServlet and AdminServlet).

- doGet: Manages GET requests related to user authentication. Supported operations include login, registration, and logout. Authenticated users are redirected to the user page; otherwise, appropriate error messages are returned. For example, when the URI ends with "login/", the current session is invalidated and the request is forwarded to the login page.
- doPost: Manages POST requests related to user authentication. Supports login and registration operations, handling client-sent data appropriately. For instance, when the URI ends with "login/", it triggers the login operation by calling the loginOperations method.

Three of the operations handled by the UserServlet are: user login, retrieval of user information, and user registration. The UserLoginDAO class checks the presence of a user in the database and creates a new user resource. The GetUserByEmailDAO class retrieves information about a user given their email. The UserRegisterDAO class is responsible for registering a new user in the system. It interacts with the database to insert the user's information, including email, password, name, surname, address, and assigns the user type as 'USER'. Note that the registration of an admin user is handled directly in the database for stricter control over the creation of admin accounts.

OrderServlet: Manages operations related to orders, such as viewing available cars, suitable circuits, completing orders, and displaying order summaries. It implements the doGet and doPost methods to manage HTTP requests.

CarListServlet: Handles requests related to retrieving the list of cars from the database and displaying them on a web page. It extends AbstractDatabaseServlet and implements the doGet method. The servlet retrieves the list of cars using ListCarDAO and forwards the request to the JSP page for rendering. ListCarDAO manages the data access for retrieving the list of cars from the database selecting all cars and constructs Car objects from the result set.

AdminServlet: Manages admin operations, such as inserting cars and circuits, edit them, inserting new car and circuit types and managing the mapping between type of cars and type of circuits. It implements the doGet and doPost methods to manage HTTP requests.

Each DAO class extends the AbstractDAO class, which manages interactions with the database, and defines the doAccess() method. The Message resource plays a crucial role in conveying messages to the end user, such as errors encountered while handling resources. All resources (message, user, order) are Java classes, each equipped with its constructors and methods for accessing the resource.

5.2 Sequence Diagram

The Sequence Diagram section list some of the main functionalities of our Web Application. One of the main operation is the Insertion of a new User in the database and its sequence diagram is shown in Fig. 9. For this operation the new user intended to register to WaCar access the Sign Up page and after compiling the form with the required information, sends a POST operation to pass the data needed to create a User to the web server which instantiate a new UserServlet. After the instantiation phase the UserServlet calls its doPost method and a new instance of the User object is created with the POST data. The UserServlet gets the connection in order to access the database and through UserRegisterDAO class performs the insertion of the newly created User in the database. The DAO class executes the INSERT INTO statement and in case of errors, such as weak password, user already present in the database or errors while accessing the database, generates an error message to inform the user of the latter. If the operation is done correctly UserServlet forwards the control to the JSP page which generates the HTML document returned to the newly created User.

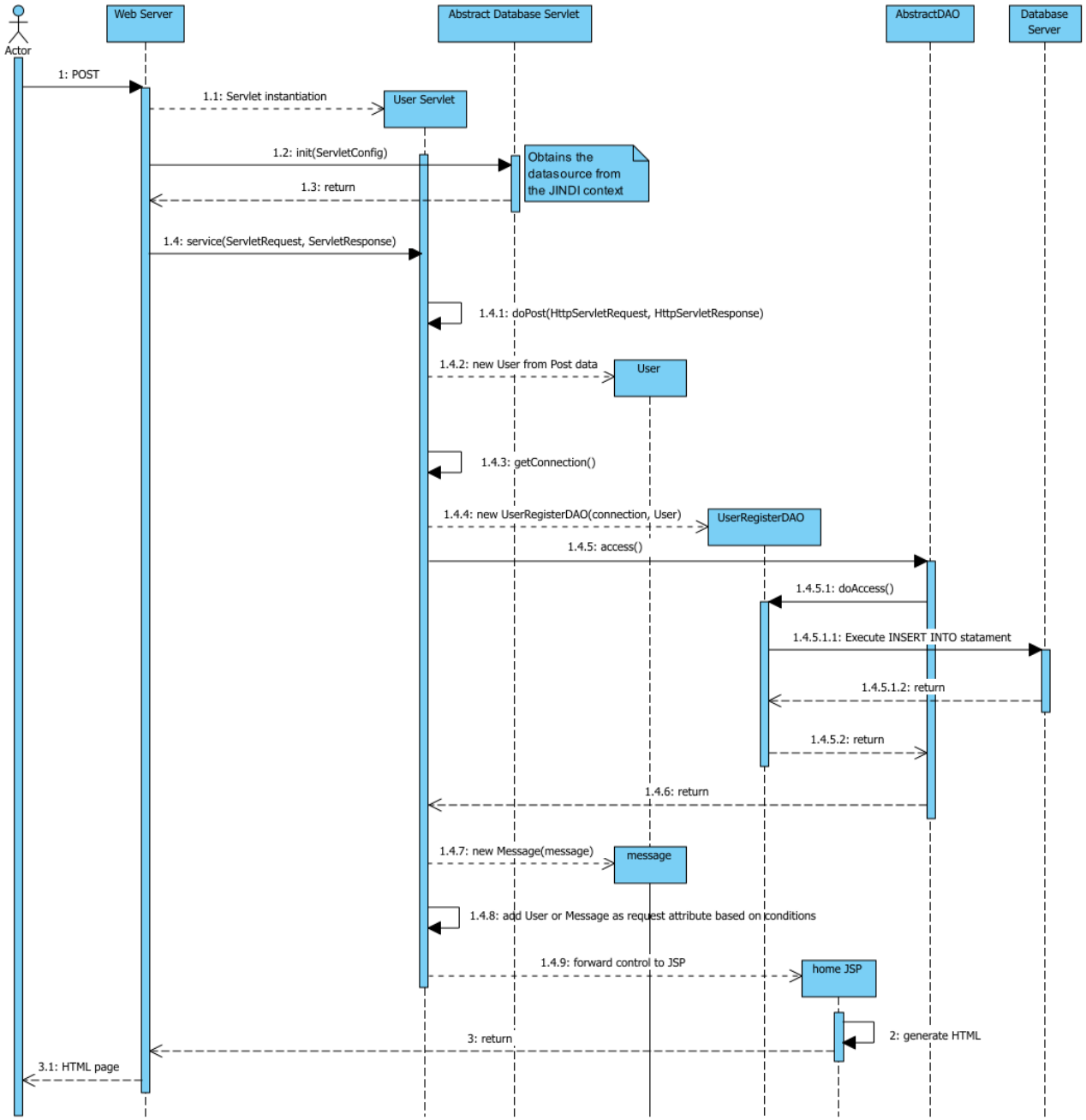


Figure 9: Insert User Sequence Diagram.

One of the basic functionalities of WaCar web application is to give the possibility to not registered user to see which brand and model of cars are available. The page List Circuit is also capable of dynamically adapt based on the fact of the type of user logged such that if the latter is an `ADMIN` it gives the possibility to modify the information about the Car or eventually delete it. The sequence diagram in shown in Fig. 10 exploits the steps done via the REST implementation of the circuit list functionality.

A logged `USER` has the possibility to review the history of its orders and also manage the one pending in order to eventually modify them. To do so, the user first needs to log in with its credentials and then access the page `List Orders`. Upon accessing such page, a `GET` request is sent to the Web Server to instantiate

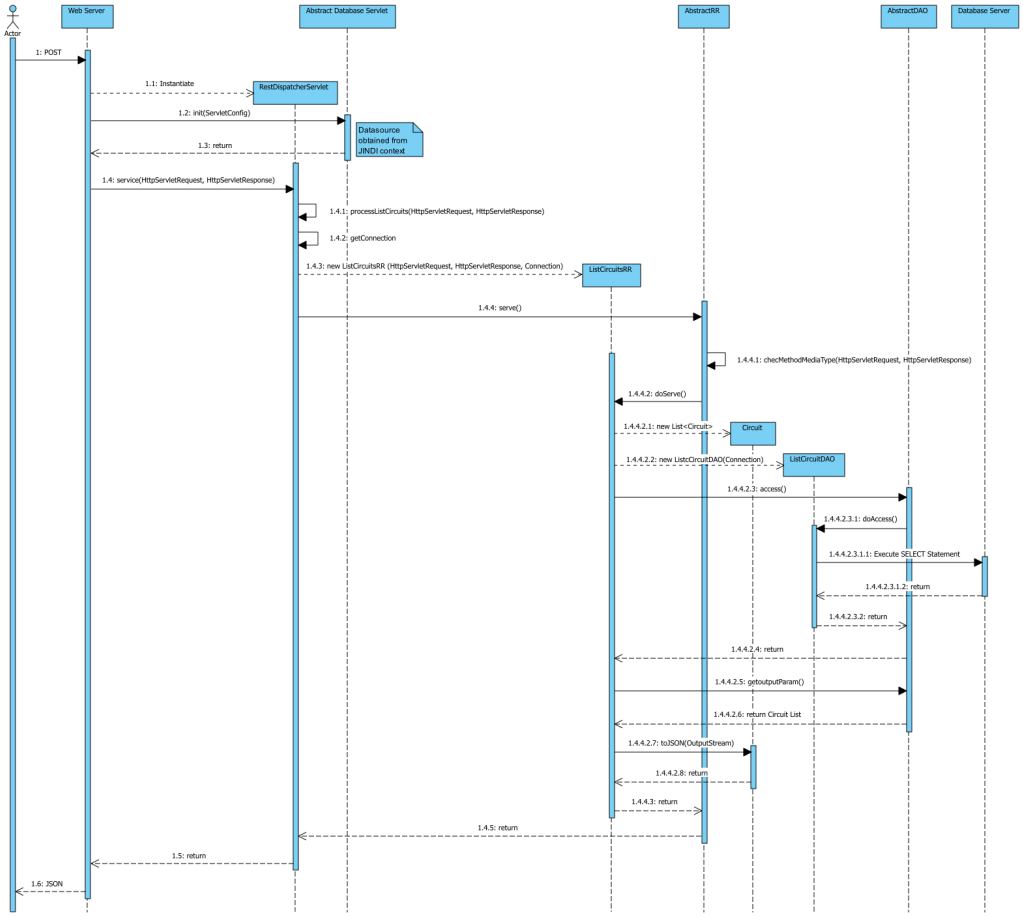


Figure 10: List Circuit REST Sequence Diagram.

the `ListOrdersByEmailServlet` which then calls its `doGet()` method and gets the connection to access the database. Afterwards, through the `ListOrdersByEmailDAO` class the appropriate `SELECT` statement is executed to retrieve the data from the database which in this case are the list of orders related to the specific user sending the request, searched in the database according to the email of the user itself. In case of errors a `Message` is instantiated to inform the user, otherwise the results retrieved by the query operation are processed and returned. The `ListOrdersByEmailServlet` forwards the control to the `list-orders` JSP page which generates the HTML document to display the information to the User.

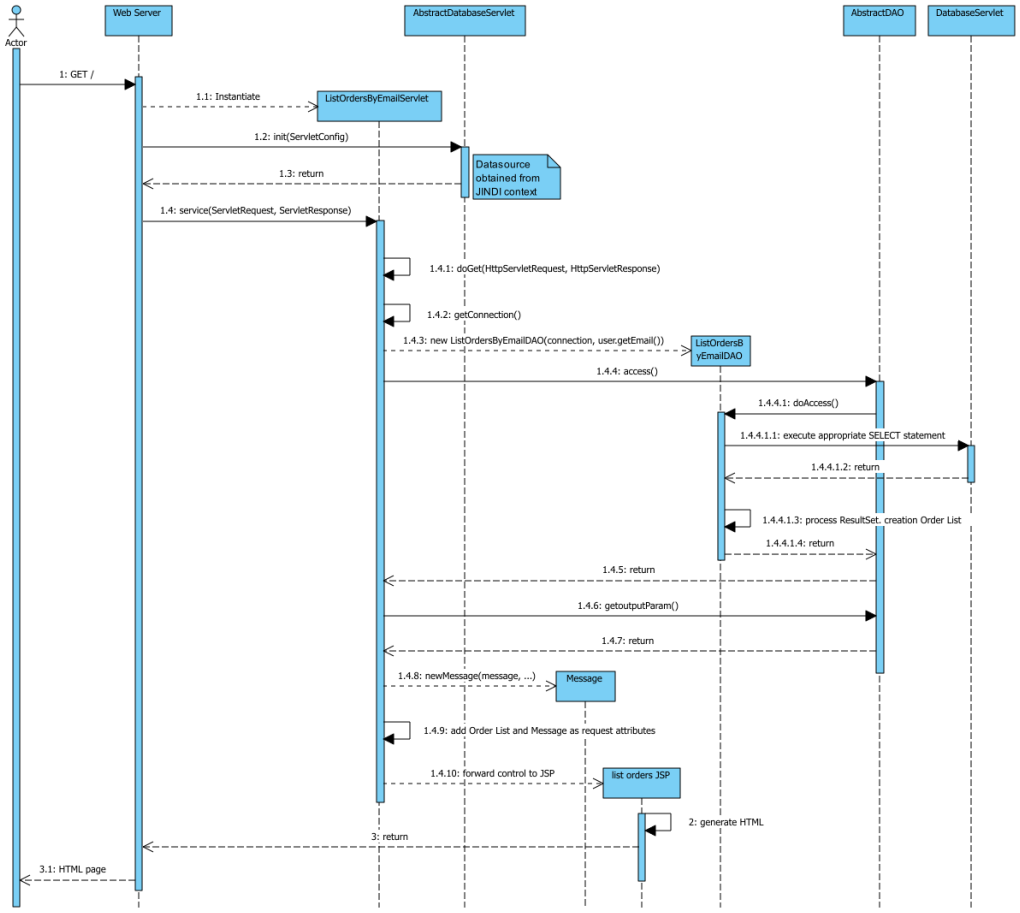


Figure 11: List Order by Email Sequence Diagram.

5.3 REST API Summary

Part of the URIs are filtered through different filters. U indicates that only users can access to the endpoint, A that only administrators can request the specified URI to the server.

URI	Method	Description	Filter
/rest/circuit	GET	Allows to list all the circuits	
/rest/cars	GET	Allows to list all the cars	
/rest/user/order/{orderId}	GET	Allows to retrieve the order with the specified id	U

Table 2: REST API Table

5.4 REST Error Codes

The provided table presents a comprehensive overview of error codes, their corresponding HTTP status codes, and descriptive explanations commonly encountered within an application or system. Error codes play a crucial role in facilitating effective communication between systems, enabling developers and administrators to quickly identify and troubleshoot issues that may arise during system operation. Each entry in the table outlines a unique error scenario, helping streamline the debugging and resolution process.

Error Code	HTTP Status Code	Description
E1A1	400 (Bad Request)	Password not compliant.
E1A2	400 (Bad Request)	Email not compliant.
E1A3	400 (Bad Request)	User does not exists.
E1A5	400 (Bad Request)	Email missing.
E1A6	400 (Bad Request)	Password missing.
E1A8	409 (Conflict)	Email already used.
E4A1	400 (Bad Request)	Output media type not specified.
E4A2	406 (Not Acceptable)	Unsupported output media type
E4A3	400 (Bad Request)	Input media type not specified.
E4A4	415 (Unsupported Media Type)	Unsupported input media type.
E4A5	405 (Method Not Allowed)	Unsupported operation.
E4A6	404 (Not Found)	Unknown resource requested.
E4A7	400 (Bad Request)	Wrong URI format.
E4A8	400 (Bad Request)	Wrong resource provided.
E4A9	400 (Bad Request)	Wrongly formatted resource provided.
E4B1	400 (Bad Request)	One or more input fields are empty or missing.
E4B2	400 (Bad Request)	One or more input fields are wrong.
E4B3	401 (Unauthorized)	Authorization needed.
E5A1	500 (Internal Server Error)	Unexpected error while processing a resource.
E5A2	409 (Conflict)	Resource already exists.
E5A3	404 (Not Found)	Resource not found.
E5A4	409 (Conflict)	Cannot modify a resource because other resources depend on it.
E5A5	500 (Internal Server Error)	Cannot create resource.
E5A6	500 (Internal Server Error)	Cannot delete resource.

Table 3: REST Error Codes

5.5 REST API Details

In this section we report all the resources available via the REST API. In particular, we implemented via REST API, all the parts of the web application that allows the final user to look at the cars and the circuits and to make an order (the REST resources have been tested via the curl method).

List circuits

The following endpoint allows to list all the circuits (and returns a JSON). For this first part of the project, to see the JSON output it is necessary to use the command "*curl http://localhost:8081/wacar/rest/circuit*" from the prompt, because currently the "application/json" media type is not supported. In this way, you can still check the first version of the page implemented with a servlet, while in the next part of the project we will complete the REST version.

- URL: `/rest/circuit`

- Method: `GET`
- URL parameters: `None`
- Data parameters: `None`
- Success Response:
  Code: 200
  Content: The JSON corresponding to the list of circuits (the tested URI is: rest/circuit):


```
{
 "resource-list": [
   {
  "circuit": {
    "name": "Autodromo Nazionale Monza",
    "type": "straights",
    "length": 5793,
    "corners number": 11,
    "address": "Viale di Vedano, 5, 20900 Monza MB",
    "description": "Ci sono molte rette",
    "lap price": 20,
    "available": true,
    "imageMediaType": "jpg"
  }
   },
   {
  "circuit": {
    "name": "Autodromo Vallelunga",
    "type": "tourism",
    "length": 4085,
    "corners number": 15,
    "address": "Via della Mola Maggiorana, 4, 00063 Campagnano di Roma RM",
    "description": "Alcune descrizioni.",
    "lap price": 12,
    "available": true,
    "imageMediaType": "jpg"
  }
   },
   ...
]
 }
```


- Error Response:
  Code: 500 (internal code: E5A1)
  Content: The message containing the error (server error when retrieving the circuits)


```
 {
```

```
    "message": {
    "message": ""Unexpected error while processing a resource."",
    "error-code": "E5A1"
}}
```

- Error Response:
  Code: 500 (internal code: E5A1)
  Content: The message containing the error (server error when retrieving the circuits)

```
{
    "message": {
    "message": ""Unexpected error while processing a resource."",
    "error-code": "E5A1"
}}
```

List cars

The following endpoint allows to list all the cars (and returns a JSON). As for the list car REST api, to see the JSON output it is necessary to use the command "*curl http://localhost:8081/wacar/rest/cars*"

- URL: `/rest/cars`
- Method: `GET`
- URL parameters: `None`
- Data parameters: `None`
- Success Response:
  Code: 200
  Content: The JSON corresponding to the list of cars (the tested URI is: rest/cars):

```
{
"resource-list": [
  {
    "circuit": {
      "name": "Autodromo Nazionale Monza",
      "type": "straights",
      "length": 5793,
      "corners number": 11,
      "address": "Viale di Vedano, 5, 20900 Monza MB",
      "description": "Ci sono molte rette",
      "lap price": 20,
      "available": true,
      "imageMediaType": "jpg"
    }
  },
  {
```

```
      "circuit": {
        "name": "Autodromo Vallelunga",
        "type": "tourism",
        "length": 4085,
        "corners number": 15,
        "address": "Via della Mola Maggiorana, 4, 00063 Campagnano di Roma RM",
        "description": "Alcune descrizioni.",
        "lap price": 12,
        "available": true,
        "imageMediaType": "jpg"
      }
    },
    ...
  ]
}
```

- Error Response:
  Code: 500 (internal code: E5A1)
  Content: The message containing the error (server error when retrieving the cars)

```
{
    "message": {
    "message": ""Unexpected error while processing a resource."",
    "error-code": "E5A1"
}}
```

- Error Response:
  Code: 500 (internal code: E5A1)
  Content: The message containing the error (server error when retrieving the cars)

```
{
    "message": {
    "message": ""Unexpected error while processing a resource."",
    "error-code": "E5A1"
}}
```

Read order

The following endpoint allows to retrieve a specific order given the id (and returns a JSON). To get the JSON of the order, you have to send an authorization token BASIC in the format user-email:password in Base64.

- URL: `/rest/user/order/{orderId}`
- Method: `GET`

- URL parameters:
  {orderId} = the identifier number of the order to be retrieved.
- Data parameters: `None`
- Success Response:
  Code: 200
  Content: The JSON corresponding to the order (the tested URI is: rest/user/order/1):


```
{
 "order":{
   "id":1,
   "account":"default@example.com",
   "date":"2024-05-10",
   "carBrand":"Pagani",
   "carModel": "Zonda R",
   "circuit": "Autodromo Enzo e Dino Ferrari",
   "nLaps":3,
   "price":90,
   "createdAt":"2024-04-25 07:50:06.857"
  }
}
```


- Error Response:
  Code: 400 (internal code: E4A7)
  Content: The message containing the error (wrong URI format)


```
{
 "message": {
  "message": "Wrong format for URI /order/{orderId}: no {orderId} specified.",
  "error-code": "E4A7",
  "error-detail": "exception content"
}}
```


- Error Response:
  Code: 401 (internal code: E4B3)
  Content: No authorization header sent by the client


```
{
 "message": {
  "message": "No authorization header sent by the client.",
  "error-code": "E4B3"
}}
```


- Error Response:
  Code: 401 (internal code: E4B3)

Content: You are not authorized to perform this operation

```
{
 "message": {
  "message": "You are not authorized to perform this operation",
  "error-code": "E4B3"
}}
```

- Error Response:
  Code: 500 (internal code: E5A1)
  Content: The message containing the error (server error when retrieving the order)

```
{
 "message": {
  "message": "Cannot retrieve the order: unexpected error.",
  "error-code": "E5A1",
  "error-detail": "exception content"
}}
```

6 Group Members Contribution

Di Martino Ludovico In this part of the project, I drew the mockup of the web application using Figma. Then I set up the *init_db.sql* (the sql file that populates the database) in order to make it run after the *wacar.sql* (the file used to create all the database tables). To do this I modified the *docker-compose.yml* and I created the *container_health_check.sh* script. Furthermore, I implemented the Admin servlet with all its functionalities: insert car, insert circuit, insert car type, insert circuit type, edit car, edit circuit, insert car-circuit suitability and delete car-circuit suitability. To do so, I also coded the useful DAOs, resource objects and JSP pages for the required functionalities. Finally, I contributed in the creation of the javadoc for the code that I've written.

Galli Filippo In this project my primary focus was on developing the homepage management and user operations. This involved implementing components such as the UserServlet and the HomeServlet, as well as crafting corresponding JSP pages (such as login.jsp, signup.jsp, userPage.jsp and home.jsp), implementing the UserRegisterDAO, the UserLoginDAO, and the GetUserByEmailDAO. Additionally, I developed the LoginFilter and the HomeFilter. I have also partially contributed in the creation of the JavaDoc, in the writing of the report and in the shaping of the Entity-Relationship schema and the error code infrastructure.

Leonardi Alessandro In this part of the project I contributed to the definition and the development of the Favourite resource, the visualization of the list of favourites and the possibility of deliting this favourites. I started by defining "Favourite.java" class, then "ListFavouriteDAO.java" and "DeleteFavouriteDAO.java", then I have continued by implementing the corresponding servlet "ListFavouriteServlet.java" and "DeleteFavouriteServlet.java". In conclusion I have represented the favourites with list-favourites.jsp where I have merged the visualization and the possibility to delete each Favourite.

Rigobello Manuel In this project, I contributed to the creation and development of the Entity-Relationship schema, the relational schema, to the population of the tables and to the triggers. Then, I started to develop the order page, which includes the jsp page (for the second homework I will improve the flow of creating an order by

including the client-side part), the ListCarByAvailabilityDAO, the ListCircuitByCarType, the InsertOrderDAO for inserting the new order into the table, the recap page and the OrderServlet that handles the different url. Next, I developed some rest api. In particular, I started with the dispatcher for retrieving a specific order given its id, and then I focused on the protected api to distinguish between user and admin api. Regarding the documentation, I partially contributed in the creation of the JavaDoc, while in the report I described the Data Logic Layer (section 4) and the Read Order REST api.

Scapinello Michele In this project, I contributed to improving the ER schema by raising questions about its structure and connecting it to the functionalities that our application needed to include. I developed DAO objects such as ListCarDAO, ListCircuitDAO, and ListOrdersByEmailDAO, along with their corresponding servlets ListCarServlet, ListCircuitServlet, ListOrdersByEmailServlet, and LoadCircuitImageServlet. Additionally, I created the associated JSP files, such as list-car.jsp, list-circuit.jsp, and list-orders.jsp. Following the introduction of the REST paradigm, I also developed the ListCarsRR and ListCircuitsRR resources and handled their invocation within the RestDispatcherServlet. I partially contributed, along with other team members, to implementing JavaDoc where necessary and addressing any general issues. In the report, I contributed to the Presentation Logic Layer section and the Business Logic Layer section.