



IT 309 SOFTWARE ENGINEERING

PROJECT DOCUMENTATION

IT car

Prepared by:
Delić Faris
Herenda Ajla

Proposed to:
Nermina Durmić, Assist. Prof. Dr.
Aldin Kovačević, Teaching Assistant

14th of June 2023

TABLE OF CONTENTS

<u>1. Introduction</u>	3
<u>1.1. About the Project</u>	3
<u>1.2. Project Functionalities and Screenshots</u>	4
<u>2. Project Structure</u>	10
<u>2.1. Technologies</u>	10
<u>2.2. Database Entities</u>	10
<u>2.3. Design Patterns</u>	10
<u>2.4. Tests</u>	11
<u>2.5. Releases</u>	15
<u>3. Conclusion</u>	16

1. Introduction

Evolving from the stage of proposing a project topic, to its development and now documentation writing, has not been a straight line. On that path we tried, failed and retried in hope of success, sounds like a learning curve, since it is.

In the Software Engineering course, we were not introduced how to develop a web application, since that was the responsibility of the Introduction to Web Programming course, rather what approaches can be taken in order to achieve our goal. Introduced to us were as well the terms of architectural and design patterns, and that in essence there is a reusable solution - pattern to our problem - our project.

The solution to our project is a layered architecture, whose need is recognised in the communication between the frontend and database, for which we use Data Access Objects, passed onto according services and furthermore routes, who are called on particular requests.

More than the above about our project and what we learnt in the Software Engineering course can be found in the sections to follow.

1.1. About the Project

Live at --> <https://car-dealership-itcars.onrender.com/index.html>

GitHub repo --> <https://github.com/ajlaherenda/Web-Programming-2023>

IT car is a car dealership web application, which allows users to browse through the business' database, which contains currently available and on sale vehicles, as well as vehicles that are soon going to be available. In case the user recognises the car of his/her dreams - his/her IT car, he/she can book a test drive by filling out a simple form, after which he will receive a confirmation email. In order to make the user's car journey even smoother, he/she can decide to search for particular car specifications through our search bar. We on the other hand, as admins of the system, can do all of the aforementioned as well as delete cars and their car ads, as well as change the ads' status, for example when an available car becomes reserved/goes on sale, or an incoming car becomes available.

Keywords:

- Web Application
- DAO CRUD Operations
- Single Page Application
- jQuery
- Car Dealership Web Application

1.2. Project Functionalities and Screenshots

Our project can go down 2 paths:

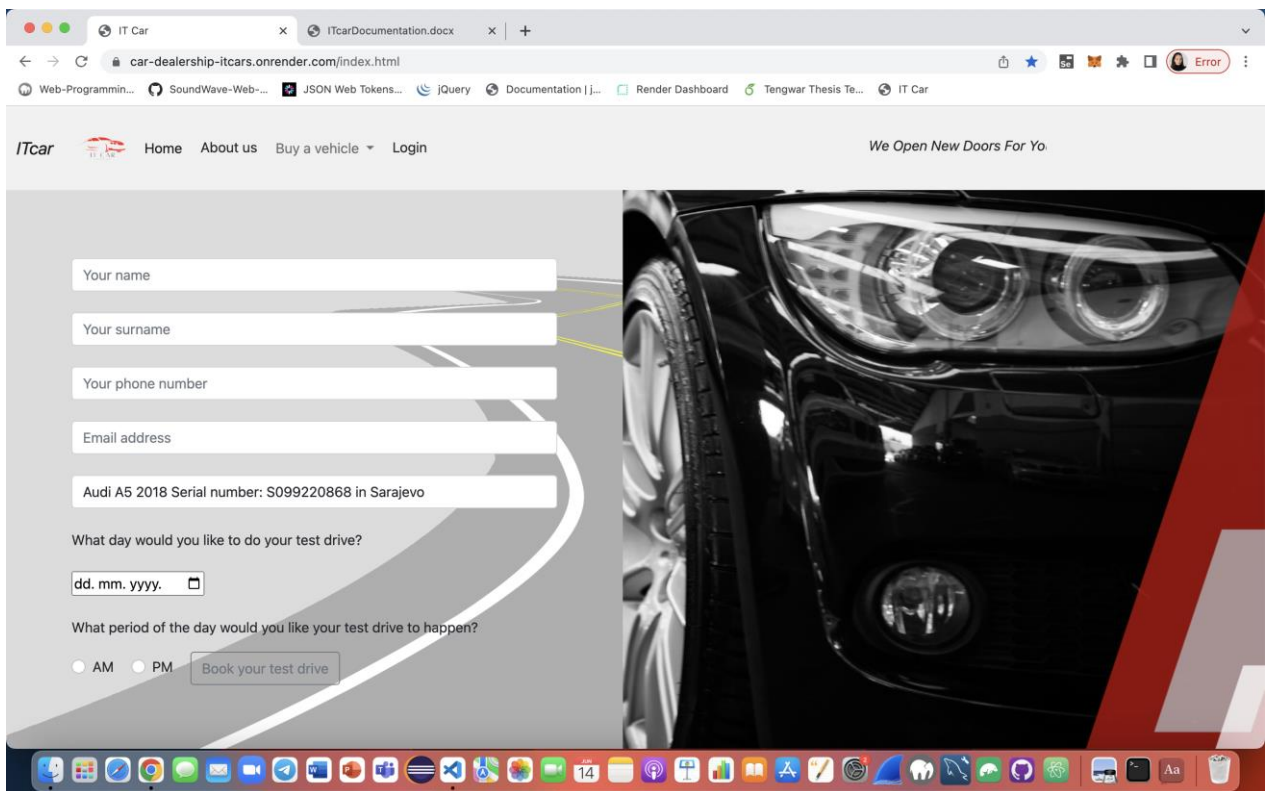
1st regular user browsing for cars

2nd admin user, managing the cars and car ads in the system \Leftrightarrow DB

The 1st user has following functionalities:

- Browsing cars on home page, based on their AVAILABILITY, if they are on SALE or if they are INCOMING
- They can search for cars based on particular search params entered in the search input field.
- The car of their choice can be highlighted in a modal, which contains more detailed car ad details.
- In case they find their IT car, they can book a test drive with it and receive upon success a confirmation email, with the test drive details.

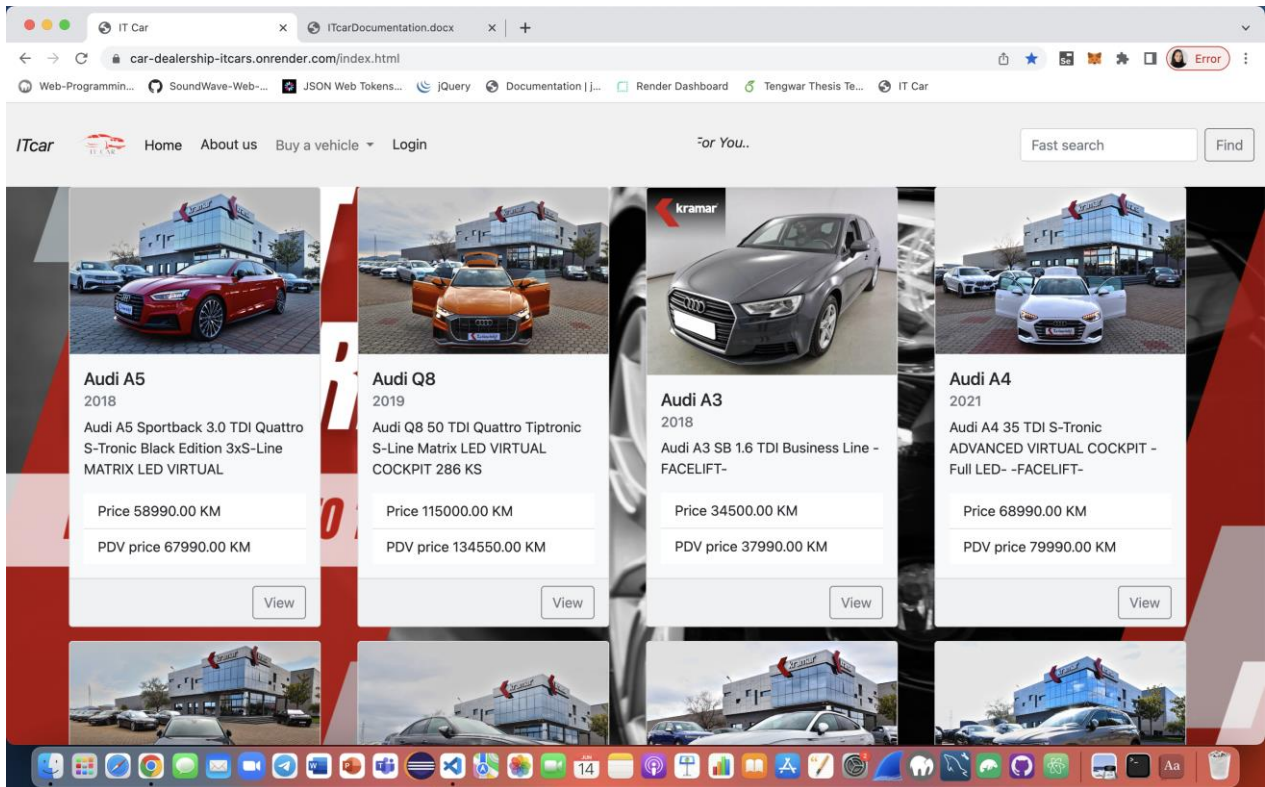
Booking a test drive



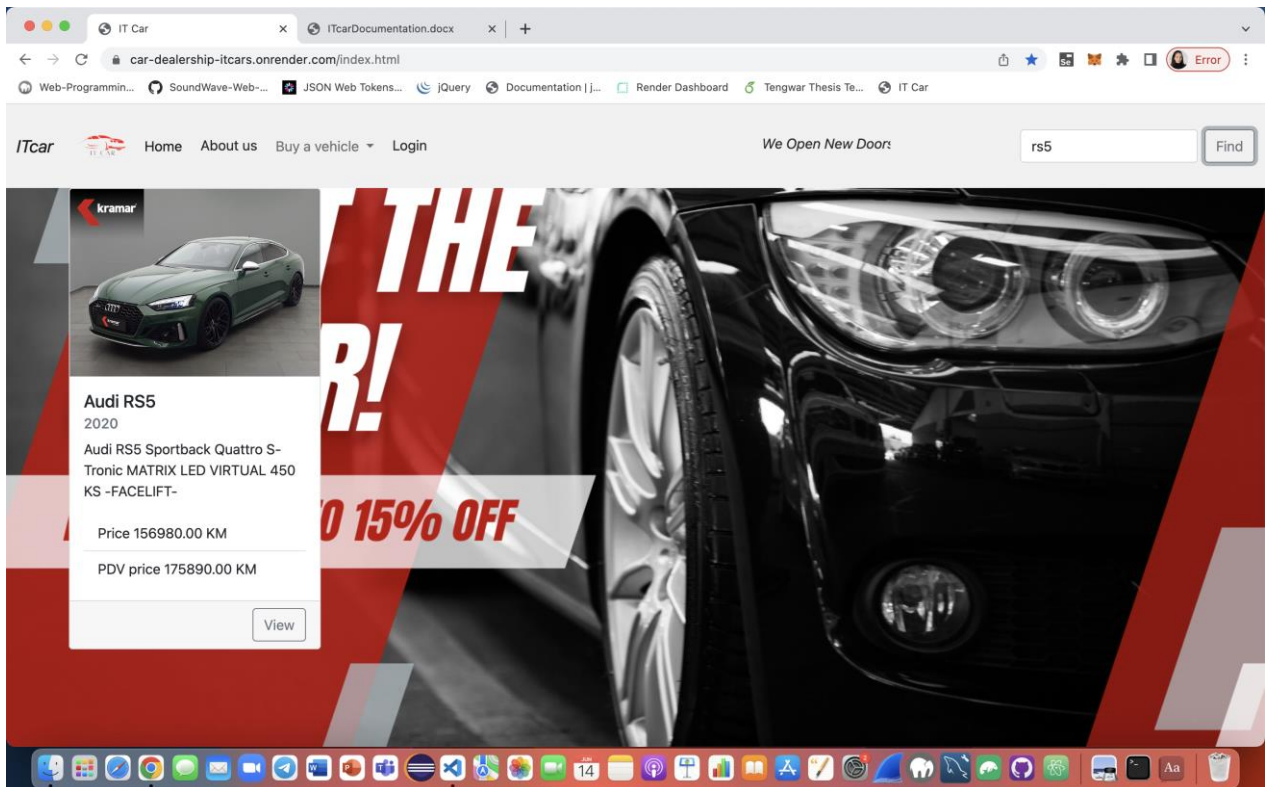
The screenshot shows a web browser window with the URL `car-dealership-itscars.onrender.com/index.html`. The website has a navigation bar with links: **ITcar**, Home, About us, Buy a vehicle, and Login. A tagline "We Open New Doors For Yo" is visible. The main content area features a booking form on the left and a large image of a car's front end on the right. The form includes the following fields and options:

- Text input: Your name
- Text input: Your surname
- Text input: Your phone number
- Text input: Email address
- Text input: Audi A5 2018 Serial number: S099220868 in Sarajevo
- Text input: What day would you like to do your test drive? (format: dd. mm. yyyy.)
- Text input: What period of the day would you like your test drive to happen? (options: AM, PM)
- Submit button: Book your test drive

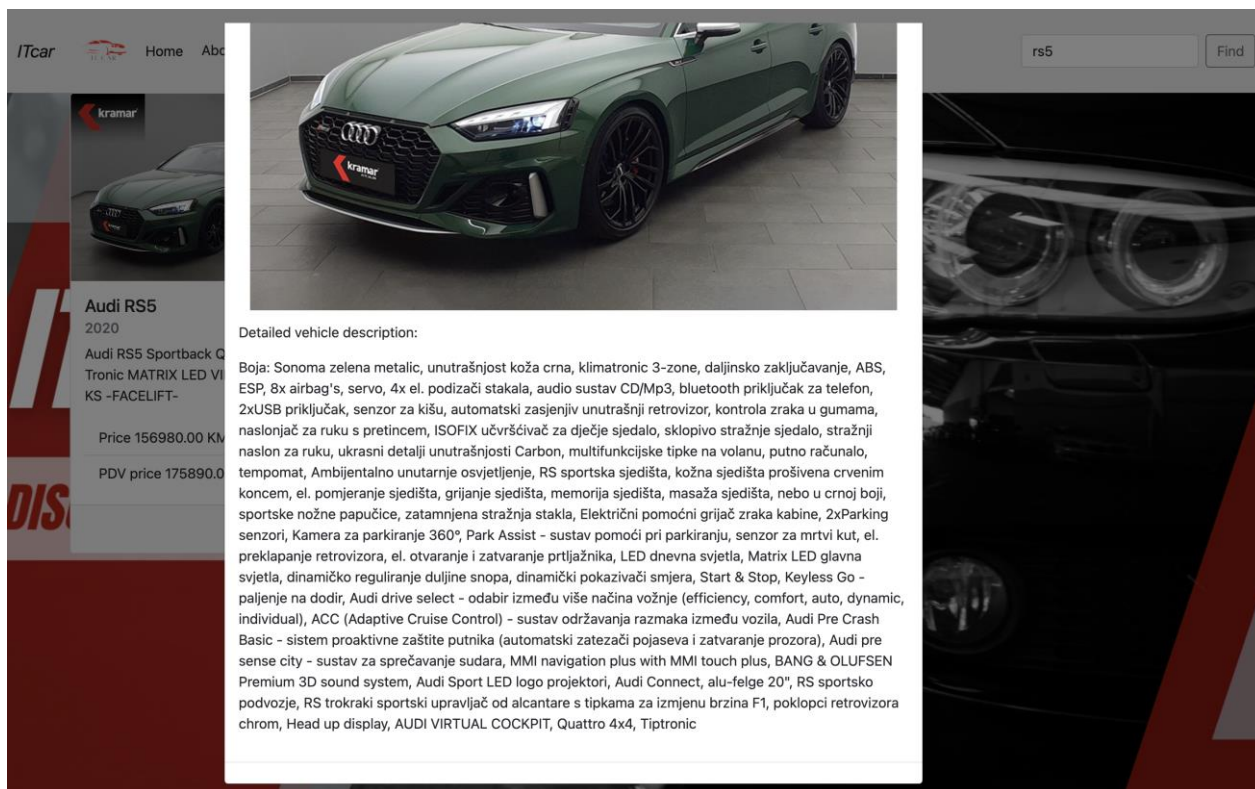
Browsing cars on home page



Searching for a rs5



Modal of the selected vehicle



Audi RS5
2020
Audi RS5 Sportback 2.0 TFSI quattro S tronic MATRIX LED V8 40 TFSI quattro
KS -FACELIFT-

Price 156980.00 KVN
PDV price 175890.00 KVN

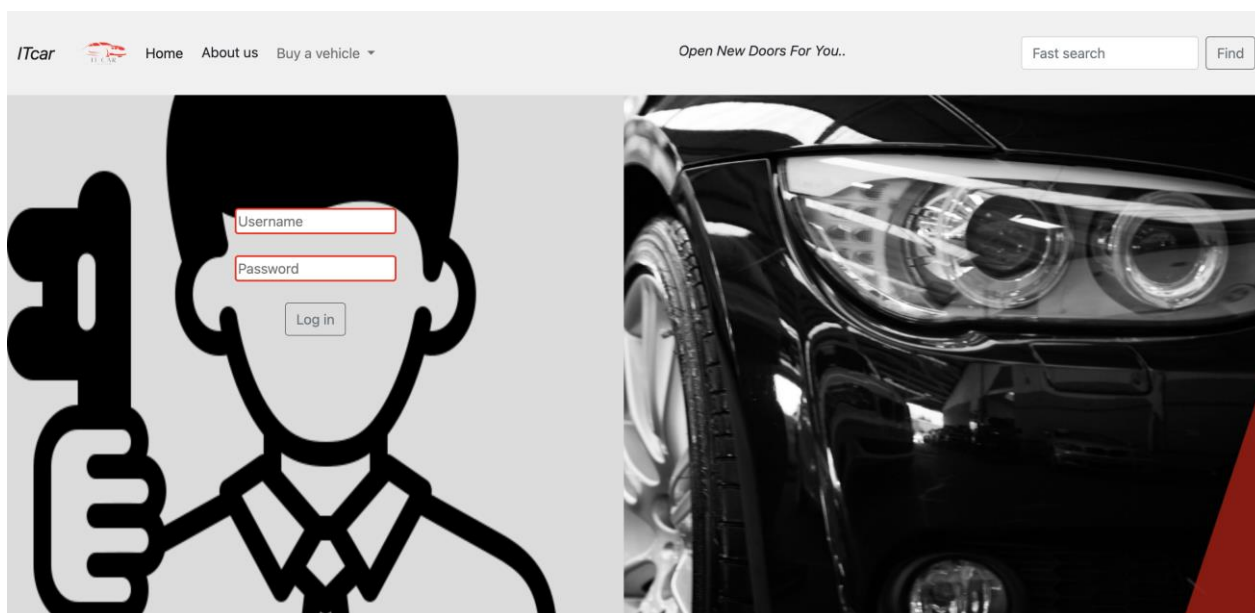
Detailed vehicle description:

Boja: Sonoma zelena metalic, unutrašnjost koža crna, klimatronic 3-zone, daljinsko zaključavanje, ABS, ESP, 8x airbag's, servo, 4x el. podizači stakala, audio sustav CD/Mp3, bluetooth priključak za telefon, 2xUSB priključak, senzor za kišu, automatski zasjenjiv unutrašnji retrovizor, kontrola zraka u gumama, naslonjač za ruku s pretincem, ISOFIX učvršćivač za dječje sjedalo, sklopivo stražnje sjedalo, stražnji naslon za ruku, ukrasni detalji unutrašnjosti Carbon, multifunkcijske tipke na volanu, putno računalo, tempomat, Ambijentalno unutarnje osvetljenje, RS sportska sjedišta, kožna sjedišta prošivena crvenim koncem, el. pomjeranje sjedišta, grijanje sjedišta, memorija sjedišta, masaža sjedišta, nebo u crnoj boji, sportske nožne papučice, zatamnjena stražnja stakla, Električni pomoćni grijač zraka kabine, 2xParking senzori, Kamera za parkiranje 360°, Park Assist - sustav pomoći pri parkiranju, senzor za mrtvi kut, el. preklapanje retrovizora, el. otvaranje i zatvaranje prtljagarnice, LED dnevna svjetla, Matrix LED glavna svjetla, dinamičko reguliranje duljine snopa, dinamički pokazivači smjera, Start & Stop, Keyless Go - paljenje na dodir, Audi drive select - odabir između više načina vožnje (efficiency, comfort, auto, dynamic, individual), ACC (Adaptive Cruise Control) - sustav održavanja razmaka između vozila, Audi Pre Crash Basic - sistem proaktivne zaštite putnika (automatski zatezači pojaseva i zatvaranje prozora), Audi pre sense city - sustav za sprečavanje sudara, MMI navigation plus with MMI touch plus, BANG & OLUFSEN Premium 3D sound system, Audi Sport LED logo projektori, Audi Connect, alu-felge 20", RS sportsko podvozje, RS trokraki sportski upravljač od alcantare s tipkama za izmjenu brzina F1, poklopci retrovizora chrom, Head up display, AUDI VIRTUAL COCKPIT, Quattro 4x4, Tiptronic

The 2nd user has all of the above functionalities and the below:

- With valid data he can log in
- As well as log out
- Update ad status
- Delete ad and car from DB

Admin login



ITcar Home About us Buy a vehicle ▾

Open New Doors For You..

Fast search Find

Username

Password

Log in

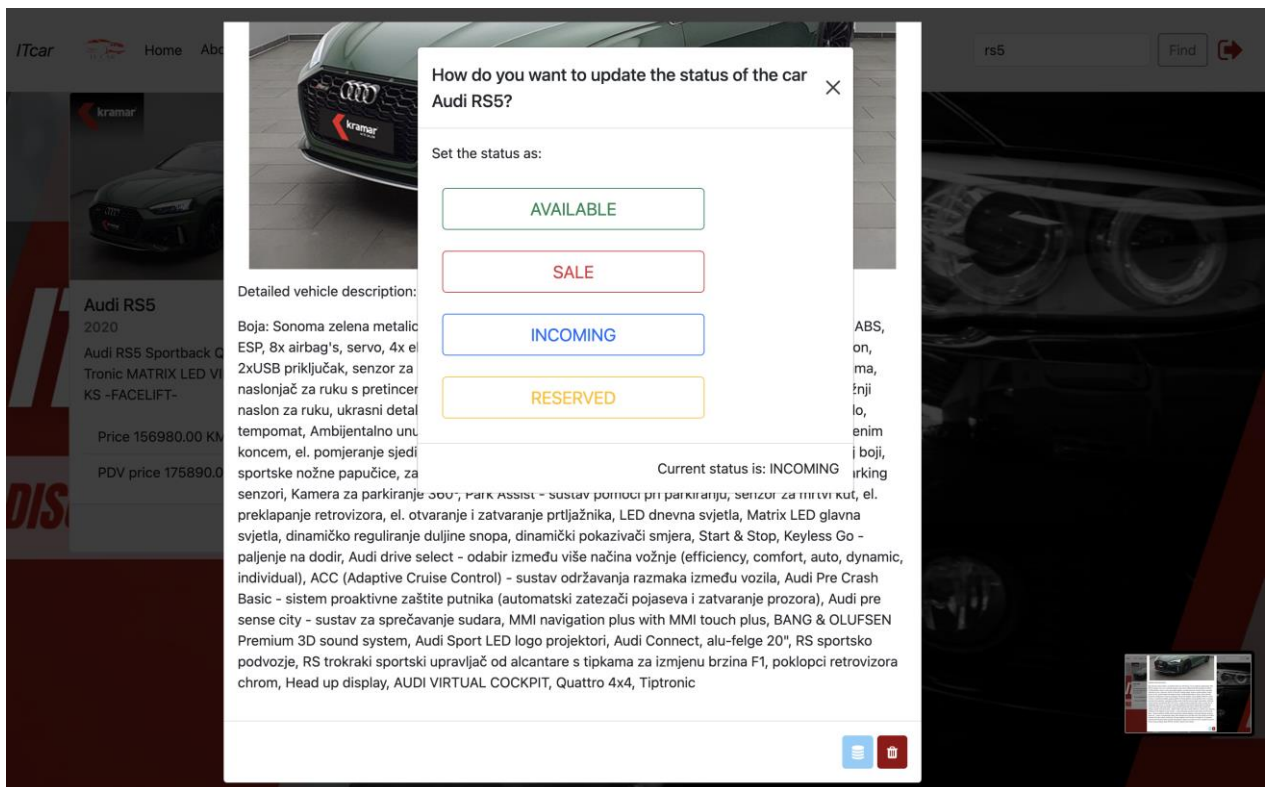
Admin Home page with Log out in upper right corner

The screenshot shows the ITCAR Admin Home page. At the top, there's a navigation bar with 'ITcar', 'Home', 'About us', and 'Buy a vehicle'. A search bar with 'Fast search' and a 'Find' button is on the right. The main content area displays a grid of car listings. Each listing includes a car image, the model name (e.g., Audi A5, Audi Q8, Audi A3, Audi A4, Audi A6, Audi A5, Volkswagen Tiguan), the year, a brief description of the model, the price, and the PDV price. A 'View' button is located at the bottom right of each listing card.

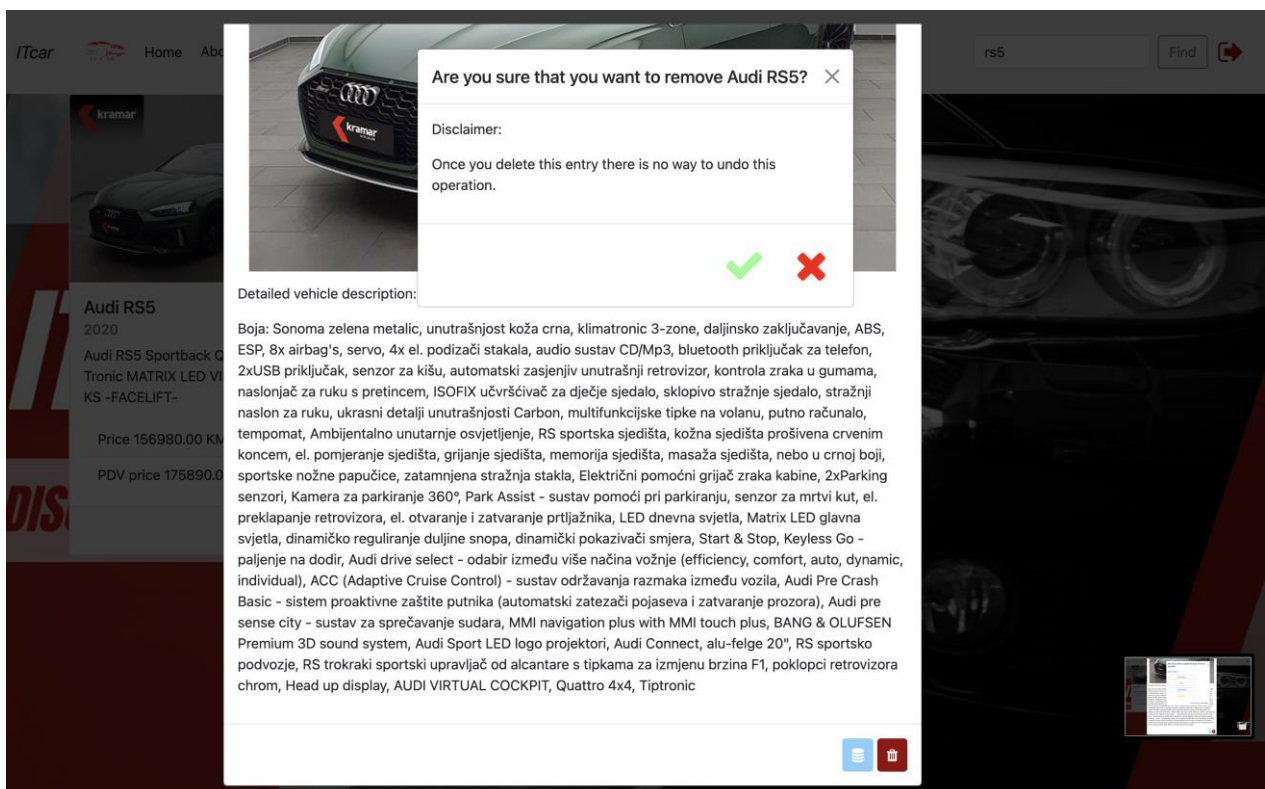
Selected car's modal with update and delete buttons

The screenshot shows the ITCAR Admin Home page with a detailed modal for the Audi RS5. The modal includes a large image of the car, a 'Detailed vehicle description:' section, and a list of features. The description mentions: 'Boja: Sonoma zelena metalic, unutrašnjost koža crna, klimatronic 3-zone, daljinsko zaključavanje, ABS, ESP, 8x airbag's, servo, 4x el. podizači stakala, audio sustav CD/Mp3, bluetooth priključak za telefon, 2xUSB priključak, senzor za kišu, automatski zasjenjiv unutrašnji retrovizor, kontrola zraka u gumama, naslonjač za ruku s pretincem, ISOFIX učvršćivač za dječje sjedalo, sklopivo stražnje sjedalo, stražnji naslon za ruku, ukrasni detalji unutrašnjosti Carbon, multifunkcijske tipke na volanu, putno računalo, tempomat, Ambijentalno unutarnje osvjetljenje, RS sportska sjedišta, kožna sjedišta prošivena crvenim koncem, el. pomjeranje sjedišta, grijanje sjedišta, memorija sjedišta, masaža sjedišta, nebo u crnoj boji, sportske nožne papučice, zatamnjena stražnja stakla, Električni pomoćni grijač zraka kabine, 2xParking senzori, Kamera za parkiranje 360°, Park Assist - sustav pomoći pri parkiranju, senzor za mrtvi kut, el. preklapanje retrovizora, el. otvaranje i zatvaranje prtljažnika, LED dnevna svjetla, Matrix LED glavna svjetla, dinamičko reguliranje duljine snopa, dinamički pokazivači smjera, Start & Stop, Keyless Go - paljenje na dodir, Audi drive select - odabir između više načina vožnje (efficiency, comfort, auto, dynamic, individual), ACC (Adaptive Cruise Control) - sustav održavanja razmaka između vozila, Audi Pre Crash Basic - sistem proaktivne zaštite putnika (automatski zatezači pojaseva i zatvaranje prozora), Audi pre sense city - sustav za sprečavanje sudara, MMI navigation plus with MMI touch plus, BANG & OLUFSEN Premium 3D sound system, Audi Sport LED logo projektori, Audi Connect, alu-felge 20", RS sportsko podvozje, RS trokraki sportski upravljač od alcantare s tipkama za izmjenu brzina F1, poklopci retrovizora chrom, Head up display, AUDI VIRTUAL COCKPIT, Quattro 4x4, Tiptronic'. At the bottom right of the modal, there are 'Update' and 'Delete' buttons.

On update modal



On delete modal



2. Project Structure

2.1. Technologies

Frontend → HTML 5 and JavaScript, written in the Airbnb style, as well as CSS for styling of our HTML, alongside inline/in document styles.

Backend → pure PHP 8.2.5, which obeys the PSR 12 coding standard.

Database → SQL DB created and managed through the MySQL Workbench, lastly hosted on the <https://db4free.net/> testing service.

*In order to follow the coding standards above, we used eslint (eslint-config-airbnb) and the PHP Sniffer developer tools.

For the deployment we used Render, due to which we had to include a Dockerfile, since Render does not provide support for PHP (in order for Render to read your environment variables written in PHP, they have to be accessed with `getenv('variableName')`).

2.2. Database Entities

- cars → contains general and most crucial information of a car that is offered at the dealership
- car_ads → contains details related to a particular car ad, related to a particular car from the cars table
- testdrives → is responsible for storing bookings of test drives
- locations → is used to store various dealership locations and their according IDs
- admins → is used to store and manually generate admin users and their usernames/passwords
- colors → entity containing the color names and their IDs

2.3. Design Patterns

The project makes use of 2 design patterns:

- **Factory method**, utilised in the creation of DAO classes, in order to prevent code repetition and increase operational efficiency, by utilising resources instantiated/defined once and used throughout the entire project. Found in the dao folder and recognised in the methods and behaviours, which classes (Ex. CarDao.class.php, CarAdsDao.class.php etc.) inherit from the BaseDao.class.php.

- **Chain of responsibility**, utilised in the frontend --> backend request and parameter forwarding, as a security layer achieved by data validations and routing middleware. Found in *routes of type POST* and *services* called by the aforementioned, as well as *DAOs*.

Ex. Routes/CarAdsRoutes.php/ line 42 and 77, which further goes to the service it calls, located in services/CarAdsServices.php/ line 26 and 37. In a similar manner this is implemented in the remaining POST methods which require input and data validation. The middleware on the other hand, which as well is a chain of responsibility can be located in the *rest/index.php line 26*. In our case the middleware is responsible for ensuring that authentication is required, when accessing particular routes related to the UPDATE and DELETE of data in the DB.

2.4. Tests

In terms of the testing, we have performed on our project, we decided to go down the JUnit path, since their implementation in PHP reminded us of the JUnit tests we have previously learned how to write in Java, as part of the Software Verification and Validation course.

The targets of our tests were the database connectivity, result sets and constraints, that were put on the business and database logic, which resulted in us writing 13 tests, found below.

Disclaimer + info:

- tests are located in the tests folder
- any method that is tested with the `assertCount()` and utilises DAO functions who return `reset($result)`, have been while testing modified to return `$result` only, due to the nature of theirs, which causes the result set not to be countable therefore.
- tests are run through the terminal `$./vendor/bin/phpunit --verbose tests`
- or `$./vendor/bin/phpunit --verbose tests/PhpFileName.php`

Each test contains a one line description regarding what it does, what the expected and actual value is.

CarAdsDaoTest.php



```

tests > CarAdsDaoTest.php
1  <?php
2
3  require_once __DIR__ . '/../rest/dao/BaseDao.class.php';
4  require_once __DIR__ . '/../rest/dao/CarAdsDao.class.php';
5
6  use PHPUnit\Framework\TestCase;
7
8  final class CarAdsDaoTest extends TestCase
9  {
10     // in the test below we check if the connection has been established and further check if that is the proper table
11     public function testDBConnection()
12     {
13         $carAdsDao = new CarAdsDao("car_ads");
14         $this->assertSame('car_ads', $carAdsDao->getTableName());
15     }
16 }
17

```

AdminDaoTest.php

tests > AdminDaoTest.php

```
1  <?php
2
3  require_once __DIR__ . '/../rest/dao/BaseDao.class.php';
4  require_once __DIR__ . '/../rest/dao/AdminDao.class.php';
5
6  use PHPUnit\Framework\TestCase;
7
8  final class AdminDaoTest extends TestCase
9  {
10     // in the test below we check if the connection has been established and further check if that is the proper table
11     public function testDBConnection()
12     {
13         $adminDao = new AdminDao("admins");
14         $this->assertSame('admins', $adminDao->getTableName());
15     }
16     // the test ensures that there is not admin in the database with the username "ime"
17     public function testGetAdminByUsername()
18     {
19         $adminDao = new AdminDao("admins");
20         $username = "ime";
21         $admin = $adminDao->getAdminByUsername($username);
22         $this->assertEmpty($admin);
23     }
24     // we check that the password for the admin with username "admin" is not "1234"
25     public function testGetAdminByUsernamePass()
26     {
27         $adminDao = new AdminDao("admins");
28         $admin = $adminDao->getAdminByUsername("admin");
29         $this->assertNotSame("1234", $admin['password']);
30     }
31 }
32
```

LocationDaoTest.php

tests > LocationDaoTest.php

```
1  <?php
2
3  require_once __DIR__ . '/../rest/dao/BaseDao.class.php';
4  require_once __DIR__ . '/../rest/dao/LocationDao.class.php';
5
6  use PHPUnit\Framework\TestCase;
7
8  final class LocationDaoTest extends TestCase
9  {
10     // in the test below we check if the connection has been established and further check if that is the proper table
11     public function testDBConnection()
12     {
13         $locationDao = new LocationDao("locations");
14         $this->assertSame('locations', $locationDao->getTableName());
15     }
16     // below we ensure that the name of the location at 71000 is actually Sarajevo
17     public function testGetById()
18     {
19         $locationDao = new LocationDao("locations");
20         $location = $locationDao->getById('71000');
21         $this->assertSame('Sarajevo', $location['location_name']);
22     }
23 }
```

CarDaoTest.php

README.md M CarDaoTest.php 2 ×

tests > CarDaoTest.php

```
1  <?php
2
3  require_once __DIR__ . '/../rest/dao/BaseDao.class.php';
4  require_once __DIR__ . '/../rest/dao/CarDao.class.php';
5
6  use PHPUnit\Framework\TestCase;
7
8  final class CarDaoTest extends TestCase
9  {
10     // in the test below we check if the connection has been established and further check if that is the proper table
11     public function testDBConnection()
12     {
13         $carDao = new CarDao("cars");
14         $this->assertSame('cars', $carDao->getTableName());
15     }
16     // we test if the number of vehicles matching the search parameters is 2
17     public function testSearchTool()
18     {
19         $carDao = new CarDao("cars");
20         $searchParam = "Audi A5";
21         $searchResult = $carDao->searchTool($searchParam);
22         $this->assertCount(2, $searchResult);
23     }
24     // tested is if the number of incoming cars is actually 2
25     public function testGetCarsBasedOnAdStatus()
26     {
27         $carDao = new CarDao("cars");
28         $cars = $carDao->getCarsBasedOnAdStatus('INCOMING');
29         $this->assertCount(2, $cars);
30     }
31     // we check if the id is indeed unique for vehicles
32     public function testGetCarsById()
33     {
34         $carDao = new CarDao("cars");
35         $car = $carDao->getCarsById('1');
36         $this->assertCount(1, $car);
37     }
38 }
```


TestDriveDaoTest.php

```
tests > TestDriveDaoTest.php
1  <?php
2
3  require_once __DIR__ . '/../rest/dao/BaseDao.class.php';
4  require_once __DIR__ . '/../rest/dao/TestDriveDao.class.php';
5
6  use PHPUnit\Framework\TestCase;
7
8  final class TestDriveDaoTest extends TestCase
9  {
10     // in the test below we check if the connection has been established and further check if that is the proper table
11     public function testDBConnection()
12     {
13         $testDriveDao = new TestDriveDao("testdrives");
14         $this->assertSame('testdrives', $testDriveDao->getTableName());
15     }
16     // we assert that there are no booked test drives with the given vehicle, date and time of test drive
17     public function testCheckAvailability1()
18     {
19         $scar = "Audi A4 2021 Serial number: S152763866";
20         $date = "11.6.2023.";
21         $time = "AM";
22         $testDriveDao = new TestDriveDao("testdrives");
23         $bookedTestDrive = $testDriveDao->checkAvailability($scar, $date, $time);
24         $this->assertEmpty($bookedTestDrive);
25     }
26     // we assert that there are is a booked test drive with the given vehicle, date and time of test drive
27     // additionally there can be only 1 for that vehicle, that date and period of day!
28     public function testCheckAvailability2()
29     {
30         $scar = "Mercedes GLC 400 D 2019 Serial number: S072220628";
31         $date = "23.6.2023.";
32         $time = "PM";
33         $testDriveDao = new TestDriveDao("testdrives");
34         $bookedTestDrive = $testDriveDao->checkAvailability($scar, $date, $time);
35         $this->assertCount(1, $bookedTestDrive);
36     }
37 }
38
```

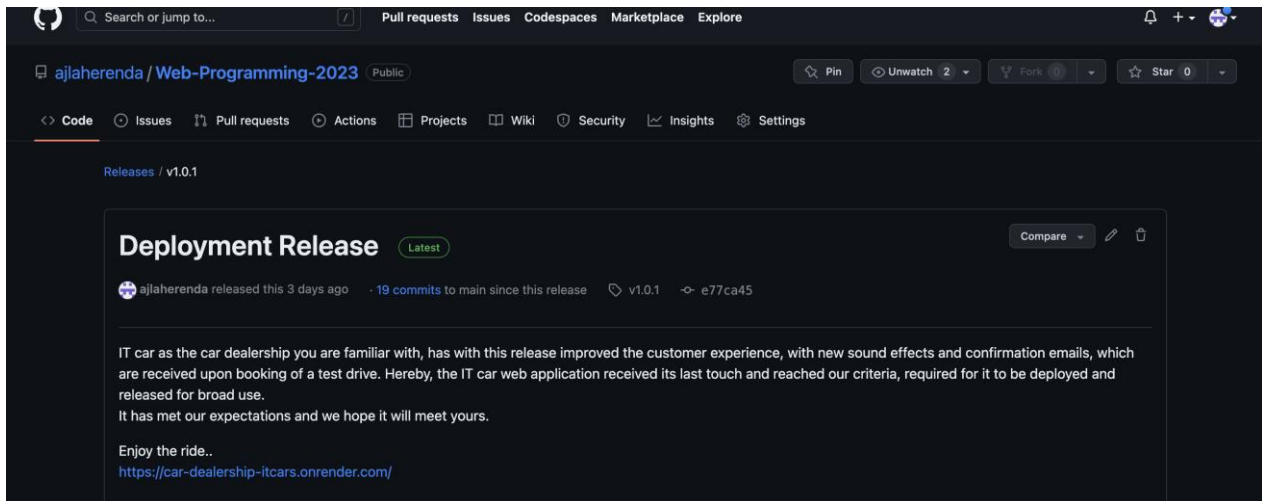
Testing results:

13/13 tests have passed, when included the modifications made to the getMethods() which return \$reset(\$result), because such return is not properly countable by assertCount(). If the aforementioned not replaced by just \$result, this would indicate that 11/13 tests would pass and 2 not passing would be the testCheckAvailability() in TestDriveDaoTest.php and testCarsGetById() in the CarDaoTest.php, due to their use of reset(), whose result set once again cannot be counted by assertCount().

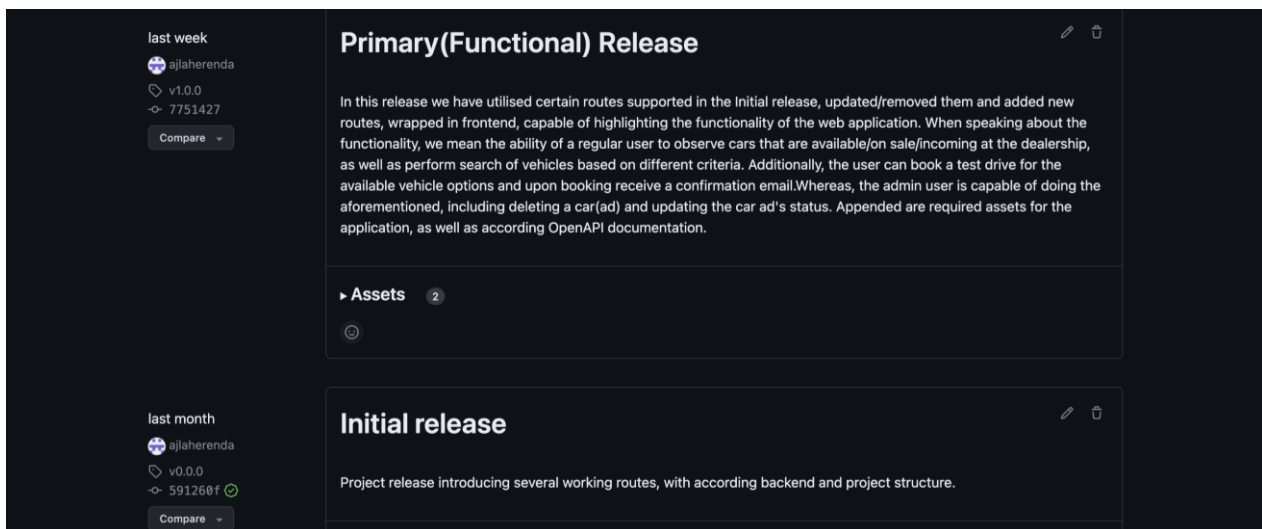
2.5. Releases

As part of this course, we had been taught about the concept of semantic versioning or the MAJOR. MINOR. PATCH. versioning, guided by whose concepts we published releases of our own project, listed underneath (3 of them in total).

Last release



First and second release



3. Conclusion

Being in a post project development phase, we consequently reach the end of the aforementioned project's documentation as well, which we hope provided required and additional information, and proves that our project has satisfied the requirements.

Recalling our statement that the learning process is not a straight line, neither was this course and the implementation of the design patterns. In particular the chain of responsibility is a behavioral design pattern, we did not cover in our lectures, but appeared on the website, which the Assistant himself used to advise us to look up for further details (link <https://refactoring.guru/design-patterns/chain-of-responsibility>). Aside from the aforementioned obstacle of implementing another design pattern, the learning curve of ours crossed the point where we revised the JUnit tests and saw them in action in PHP, as well as the point where we got to learn how to deploy our application to services like Render and distribute it to the public with according releases, which we learnt how to version.

The learning curve has still not reached its highest point on the y axis and never will, since we learn our whole life, but at the moment we are sufficiently distant from the coordinate beginning and at the end of this course and project, to state that this course, project and lab sessions, have jointly and undoubtedly contributed to the project development(both courses), since it offered us a new perspective of how things are built and what should be built, but not only with the aim to be functional, but to function efficiently and resolve a particular user problem or satisfy a user's need.

Thank you for being a part of our learning curve!