

**IT 309 SOFTWARE ENGINEERING**

PROJECT DOCUMENTATION

Car Service App

Prepared by:

**Ishak Kazić**

**Bilal Duraković**

Proposed to:

**Nermina Durmić, Assist. Prof. Dr.**

**Aldin Kovačević, Teaching Assistant**

20.6.2023.

Content

[1. Introduction 3](#_Toc137670460)

[1.1. About the Project 3](#_Toc137670461)

[1.2. Project Functionalities and Screenshots 3](#_Toc137670462)

[2. Project Structure 4](#_Toc137670463)

[2.1. Technologies 4](#_Toc137670464)

[2.2. Database Entities 4](#_Toc137670465)

[2.3. Design Patterns 4](#_Toc137670466)

[2.4. Tests 4](#_Toc137670467)

[3. Conclusion 5](#_Toc137670468)

# 1. Introduction

## About the Project

The project/application we have been working on is a Car Service Management System. It is designed to streamline the process of managing car service requests, scheduling appointments, tracking vehicle maintenance history. The application provides an intuitive user interface for both service providers and customers to interact with the system efficiently. The project is deployed and can be accessed at the following link: https://car-service-ibu.netlify.app.

## 1.2. Project Functionalities and Screenshots

The car service website offers a range of functionalities to enhance the user experience and facilitate efficient car maintenance and parts shopping. The website provides user authentication, allowing users to create accounts, log in, and log out securely. Once logged in, users gain access to personalized features and information. A responsive navigation bar is available throughout the website, ensuring easy and intuitive navigation. Users can effortlessly browse different sections such as the dashboard, orders, shop, schedule, and cart. To facilitate the search for specific car parts, a search functionality is implemented. Users can enter keywords or specific part names to quickly find the desired items, saving time and effort. The home page displays recently added items, allowing users to stay updated with the latest additions to the inventory. They can view the product images, titles, types, and prices, helping them make informed decisions. The website includes a schedule management feature that enables users to book and manage appointments for car servicing or repairs. This functionality ensures a streamlined process for both customers and service providers. In the shop section, users can explore a wide range of car parts and accessories. Detailed information, including product descriptions, specifications, and pricing, is provided to assist users in making purchase decisions. Users can add items to their shopping cart and proceed to checkout for a seamless shopping experience. Special deals and discounts are showcased to attract users' attention and provide them with opportunities for savings. These deals may include discounted prices, bundle offers, or limited-time promotions. Finally, the website's footer contains useful links, such as sections for home, features, pricing, FAQs, and about. Additionally, a newsletter subscription form is available, allowing users to receive a monthly digest of new updates and exciting offers.

Overall, the website combines user-friendly functionalities, efficient search capabilities, convenient scheduling, and a seamless shopping experience to cater to the needs of car owners and enthusiasts.

# 2. Project Structure

## 2.1. Technologies

The car service website was developed using FlightPHP as the backend framework. We utilized MySQL as the database for efficient data storage. On the frontend, we used React.js and deployed the website using Netlify. Furthermore, we incorporated JWT (JSON Web Tokens) for secure authentication and authorization. To simplify the deployment process, we employed Docker, ensuring easy setup and scalability. These technologies, frameworks, and tools enabled us to create a robust, scalable, and secure car service website with efficient data management.

Afterwards, specify which *coding standard* you used and in which part of your project (was it on the backend, frontend, both, etc.). If you are unclear about coding standards, refer to Week 2 and Week 3 on LMS.

## 2.2. Database Entities

Entities in the CarService Database:

* Appointments: Represents appointments made by customers for vehicle services.
  + Fields: id, customer\_id, vehicle\_id, appointment\_date, service\_type\_id, notes
* Customers: Stores information about customers.
  + Fields: id, FirstName, LastName, Email, password, Phone, Address
* Orders: Contains details of customer orders for parts.
  + Fields: id, part\_id, customer\_id, quantity, order\_date
* Parts: Represents individual parts available for purchase.
  + Fields: id, name, description, price, photo\_link
* Services: Stores information about vehicle services performed.
  + Fields: id, VehicleID, ServiceDate, serviceTypeId
* ServiceTypes: Represents different types of vehicle services offered.
  + Fields: id, serviceName, serviceDescription, serviceCost
* Vehicles: Stores information about customer vehicles.
  + Fields: id, CustomerId, Make, Model, Year, VIN, LicensePlate

## 2.3. Design Patterns

Singleton Pattern:

* Implemented in various modules and components throughout the project, such as database connections and configuration objects. It ensured that there was only one instance of these classes in the application.

Observer Pattern:

* Applied in the frontend components, specifically in the React framework. It facilitated data communication between parent and child components and handled state changes in the child components.

Facade Pattern:

* Used in the backend component of our project. We created a facade.php file that served as a simplified interface for accessing routes and services. It effectively hid the implementation details and provided a cleaner entry point in index.php.

List the *design patterns* that you used in the project, and where they are in your source code (which files or exact line numbers). For example:

* builder pattern: used in the backend, in the file *rest/v1/OrderBuilder.php*

If you have a lot of design patterns, you can also add a separate subsection for type. Moreover, for each pattern, *briefly explain why* you chose to use that pattern and how it helped you out in that case. For example:

* “Since our orders can contain a multitude of parameters, many of them being optional, putting them all as constructor parameters would make our code messy. Therefore, we decided to use the builder pattern instead to create order objects. This makes it possible to build the order step-by-step, and only using the parameters that are necessary for that specific case…”

If you are not clear on what you need write in this section, refer to Week 5 and Week 6 on LMS, where we covered creational, structural and behavioral patterns, and visit [Refactoring.Guru](https://refactoring.guru/design-patterns) for detailed explanations on design patterns and their use cases.

## 2.4. Tests

Describe which kinds of tests you wrote for your application (e.g. unit tests, Selenium tests, etc.) and where they are located inside the project.

# 3. Conclusion

Provide some closing statements or your final thoughts about the project you implemented. Are you satisfied with the overall implementation you managed to do? Are there any things you think you could improve on in the future? Was there something that was particularly difficult or challenging to implement?

You can note whatever concluding thoughts you have about the project here, but please *do try* to write something, and not leave this section blank.