**CS673 Software Engineering**

**Team Rocket - Project RCM: Rental Car Management**

**Software Design Document**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Alisa Belousova | Configuration Lead | *Alisa Belousova* | 9/9/2023 |
| Nick Cruz | Team Leader/ Requirements Lead | *Nickolas Cruz* | 9/9/2023 |
| Devon Duddley | QA Lead and Design and Implementation Lead | *Devon Dudley* | 9/9/2023 |
| Kris | Security Lead | *Cangqing (Kris) Wang* | 9/9/2023 |
| Chenghao Ye | Design and Implementation Lead | *Chenghao Ye* | 9/10/2023 |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| 1 | Devon Dudley | 9/24/2023 | Filled out doc |
| 2 | [Alisa Belousova](mailto:alisa007@bu.edu) | 9/26/2023 | Deleted mvc pattern, updated UI design |
| 3 | [Alisa Belousova](mailto:alisa007@bu.edu) | 10/08/2023 | Add security design, database design patterns, glossary sections |
| 4 | [Alisa Belousova](mailto:alisa007@bu.edu) | 10/09/2023 | Add class and db diagrams, software architecture section, update references |
| 5 | [Alisa Belousova](mailto:alisa007@bu.edu) | 10/16/2023 | Update business logic, update security design |

[Introduction](#_87t9hln2vjz0)

[Software Architecture](#_3ipvmjgn6clp)

[Frontend components](#_9uxgr4rk079m)

[Class Diagram](#_ky60nv8suxxm)

[UI Design](#_7ucksmkf6rzx)

[Home Screen](#_u5aevacz3bk)

[Login/Signin Screen](#_y04bo2tc3zmz)

[Console - Manage Cars](#_gztxs14kdp06)

[Console - Add Car/Edit Car](#_8y7v4v63r4gw)

[Console - Car Information](#_9assbk459nwx)

[Console - Maintenance](#_z2tda7c4hg1s)

[Console - Edit Maintenance](#_g3o5blub7frj)

[Database Design](#_tcmuor4nl1kz)

[Schema Design and Collections](#_lpaexh50apcy)

[Security Design](#_x18fj36s1121)

[Password Hashing](#_m2y7678tnmzh)

[Session Management](#_75avsog0z4a7)

[Data Validation](#_vsq1ilbwtjml)

[Data Encryption](#_1sezppl8zye9)

[Business Logic and/or Key Algorithms](#_mtfbusfb0eq3)

[Design Patterns](#_9zvwkmc4luo5)

[Chain of Responsibility Pattern](#_t24mac64n0vb)

[References](#_50ojo9i46ytq)

[Glossary](#_8n34lvocupub)

# Introduction

This document provides an in-depth design overview of our car tracking ERP system tailored for individuals renting out their cars. The design goals of our software system are to provide a user-friendly interface, facilitate easy car and maintenance management, ensure the security of user information, and optimize the retrieval and update operations for swift user interactions.

# Software Architecture

Our team has opted for an approach that seamlessly integrates with Vercel's platform, which inherently employs a serverless architecture. At its core, this architecture allows us to build and run applications without the need to manage the underlying infrastructure actively. ERP application is composed of functions that are triggered by various events, including HTTP requests, which are executed in stateless compute containers that are ephemeral, making our solution both scalable and efficient.

Furthermore, with a global Content Delivery Network (CDN), assets and content are delivered expediently to users around the world, thereby ensuring low latency and high performance for all users, irrespective of geography. With Vercel’s enhanced performance metrics and high-speed networks, we are confident that our applications will consistently deliver optimal experiences to our end-users.

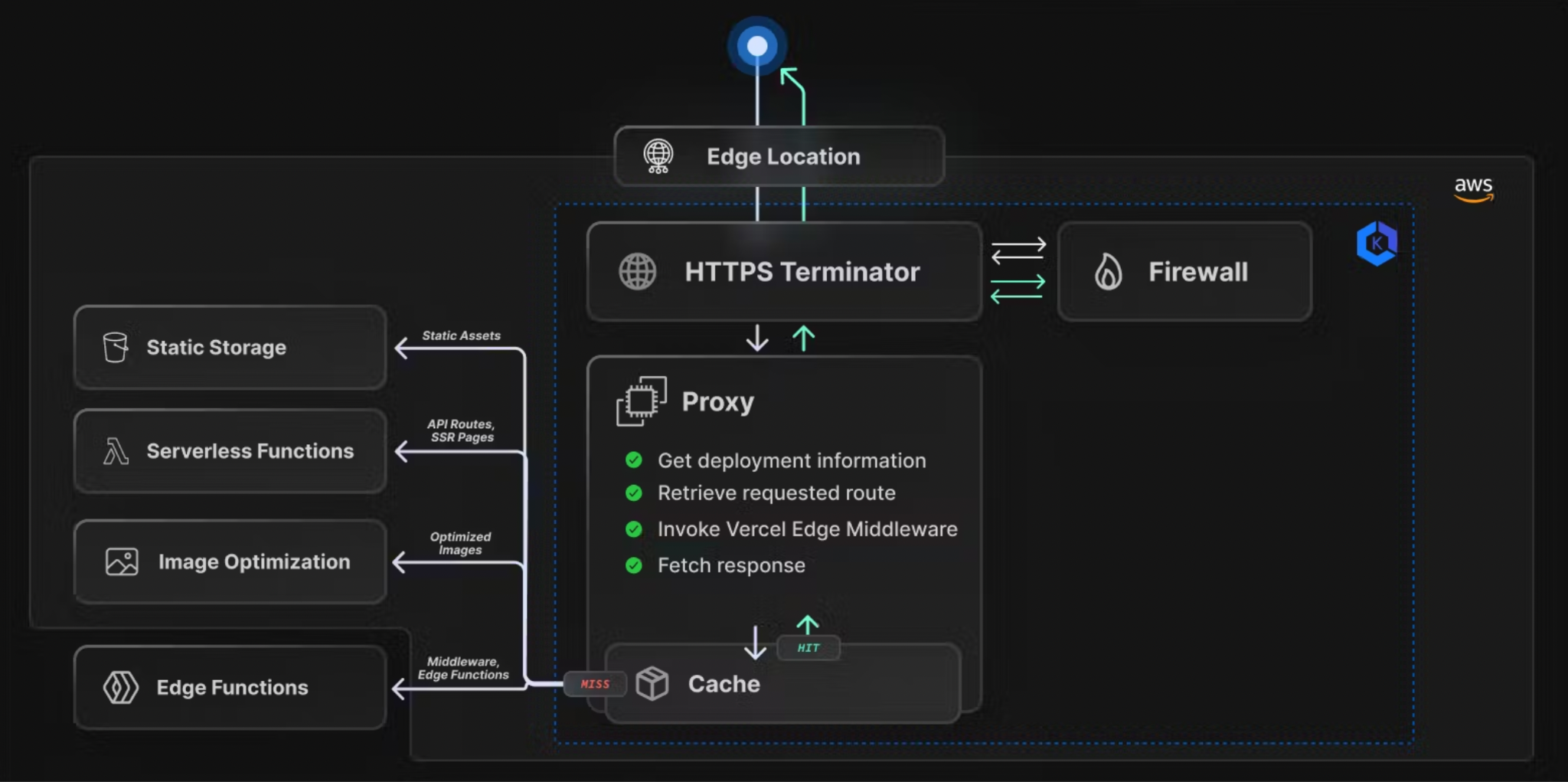
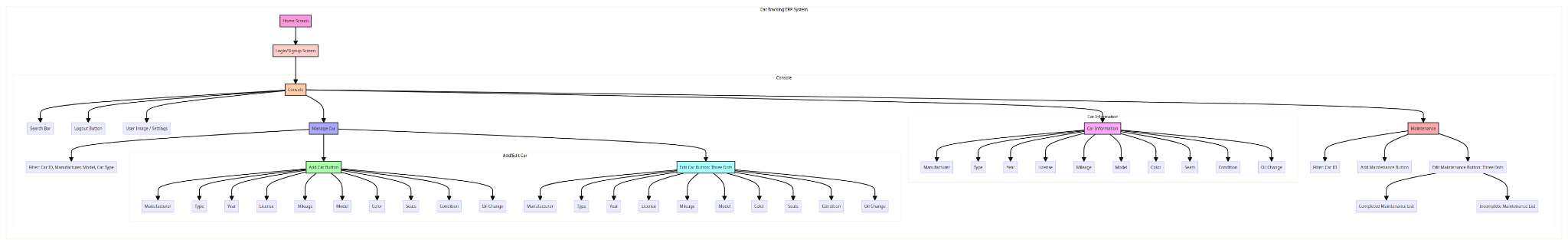


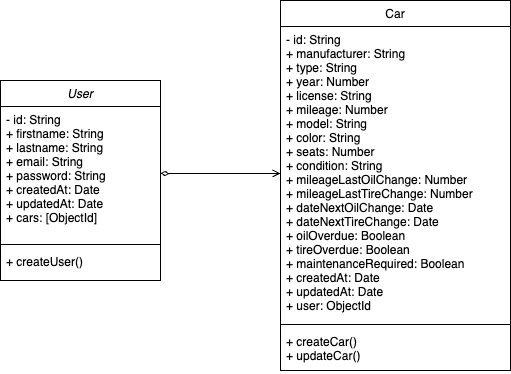
Figure 1: Flow of the request from client to requested resource. Retrieved from <https://vercel.com/blog/behind-the-scenes-of-vercels-infrastructure>

Adopting a serverless architecture with Vercel has not only streamlined our development and deployment workflows but has also ensured that our applications are performant, resilient, and ready to scale at a moment’s notice. This architectural decision underscores our commitment to delivering quality, performance, and value, all while ensuring that our team can remain focused on building great application.

## Frontend components



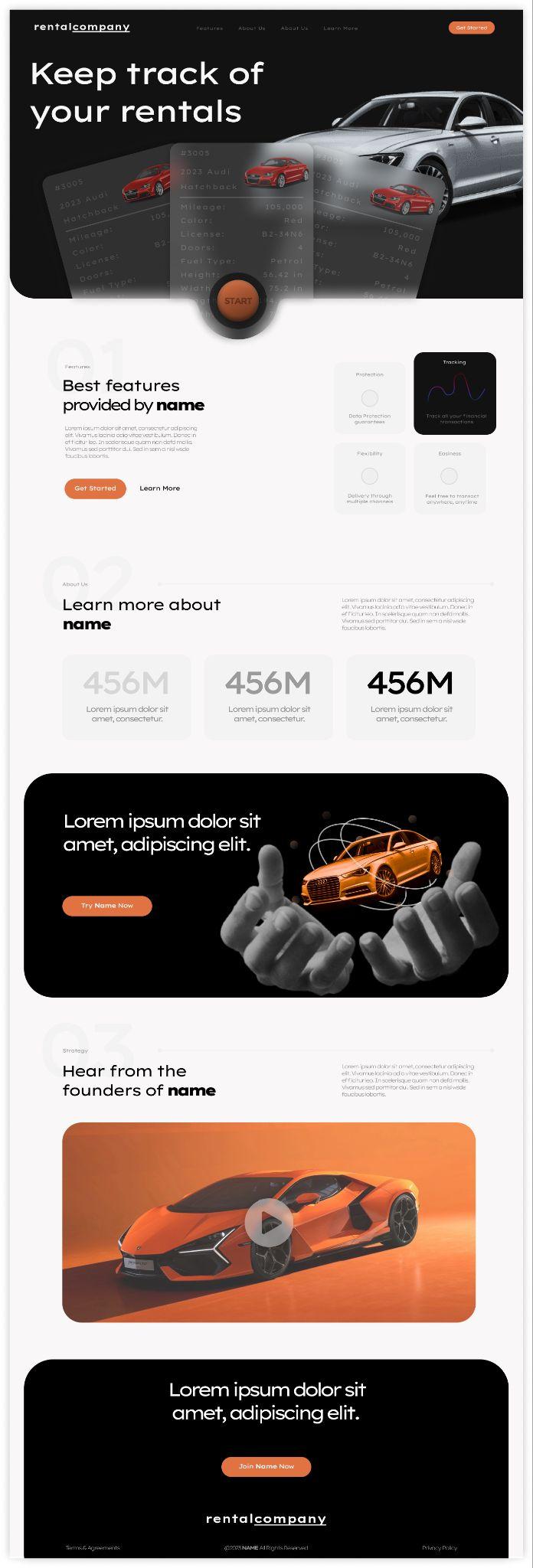
# Class Diagram



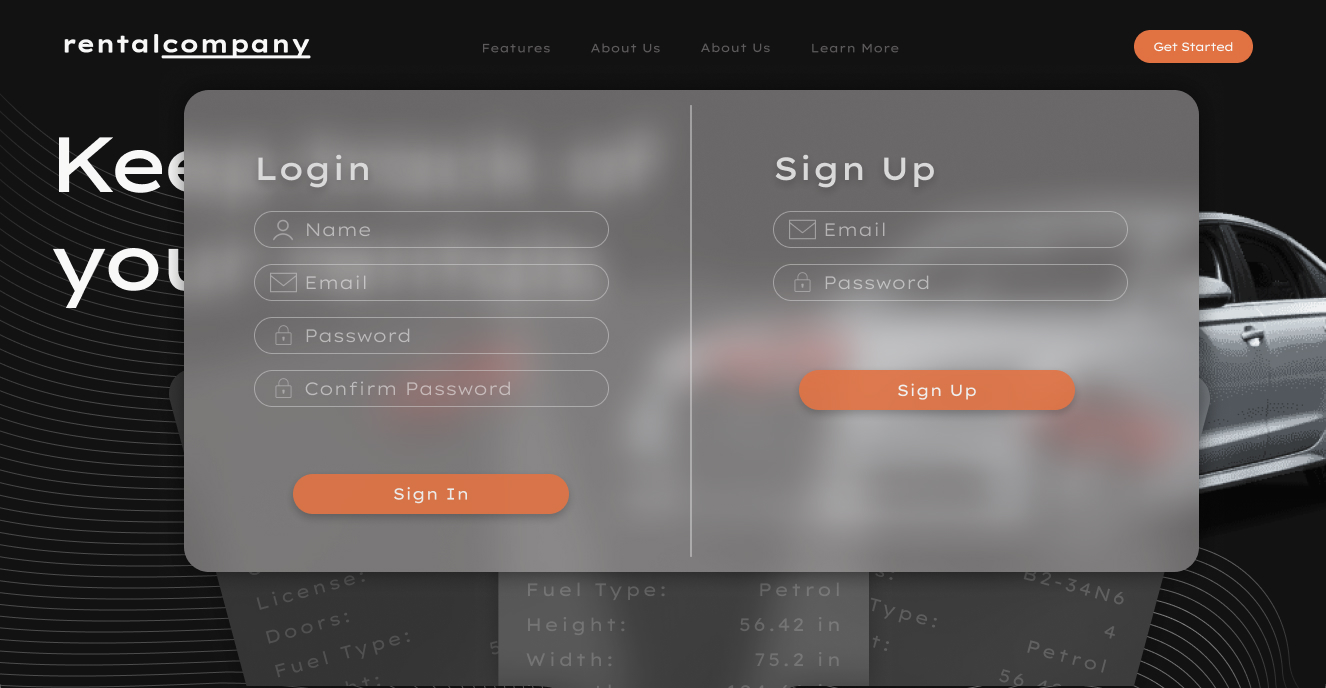
# UI Design

<https://www.figma.com/file/7Kn5zEgFuIhzLyTLVHKGPS/cs673-Group-Project?type=design&node-id=0-1&mode=design>

## Home Screen

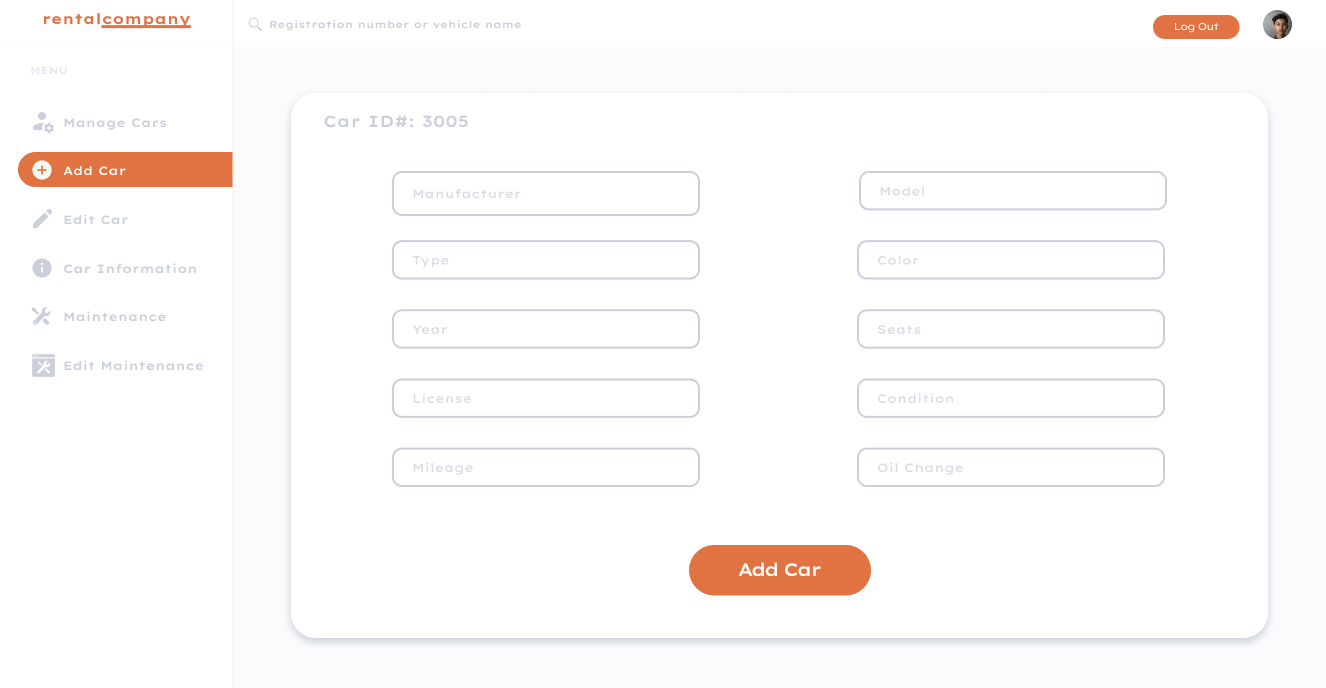


## Login/Signin Screen

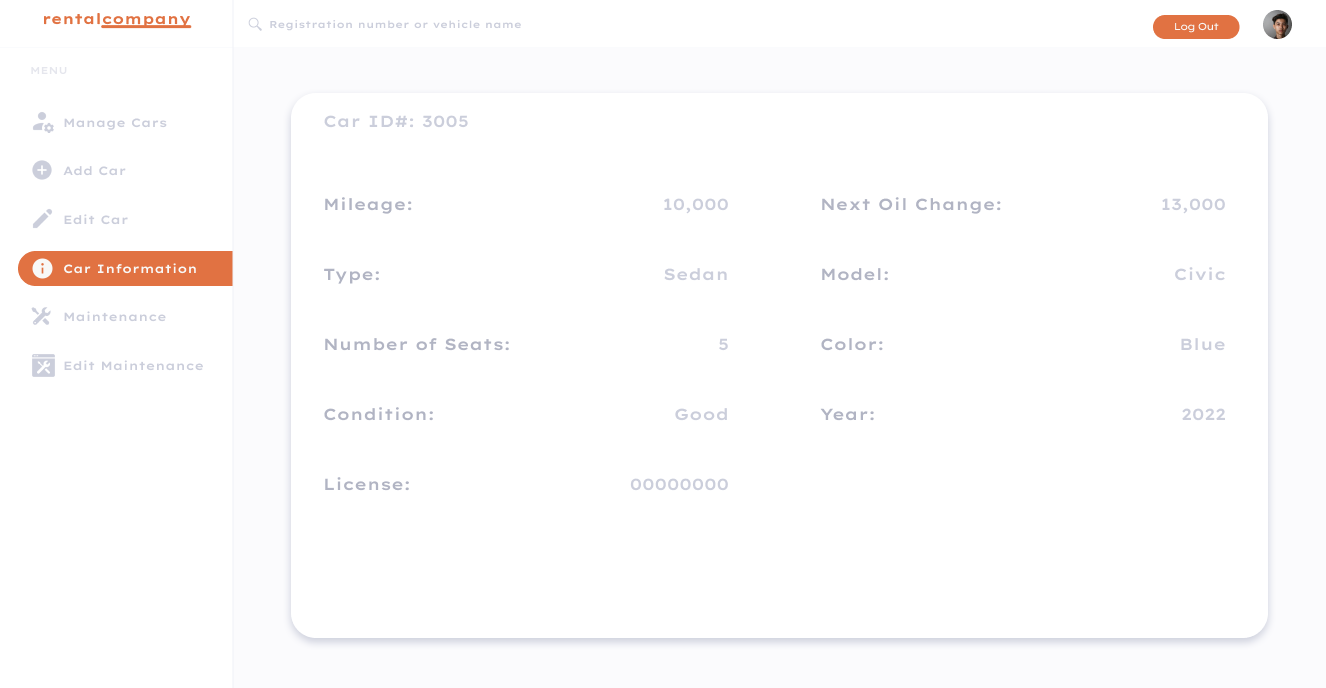


## Console - Manage Cars

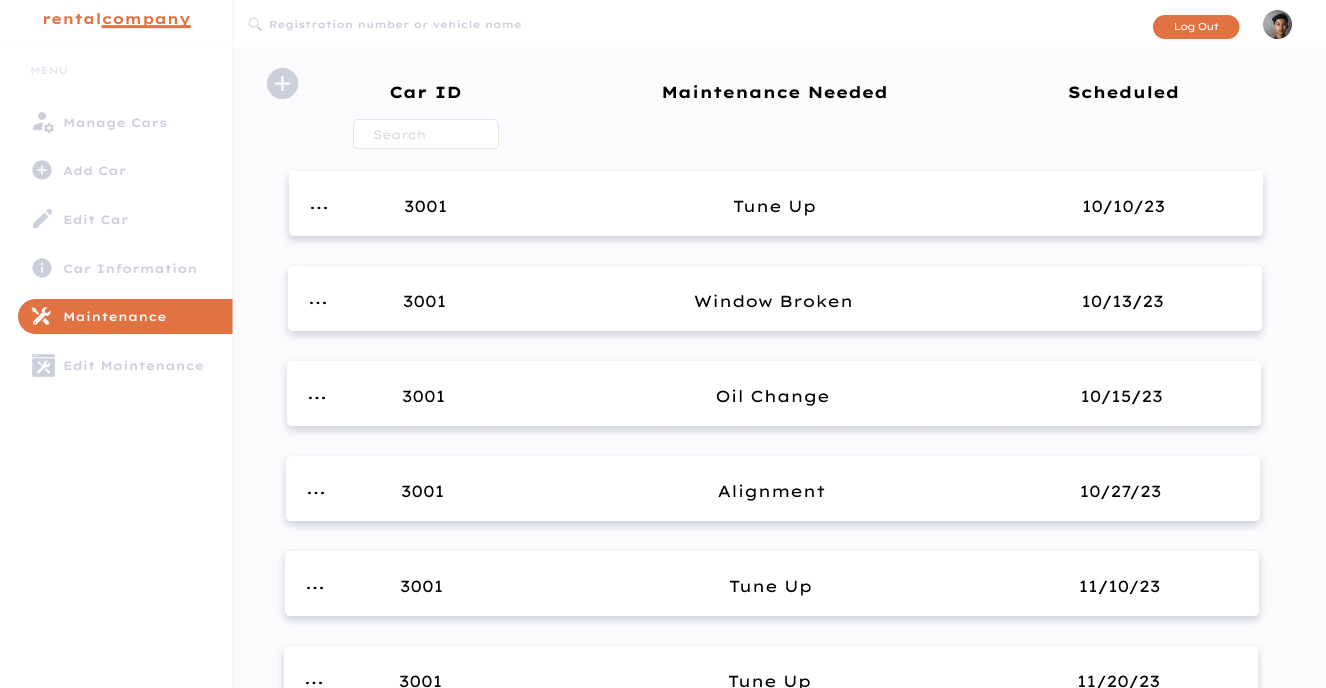
## Console - Add Car/Edit Car



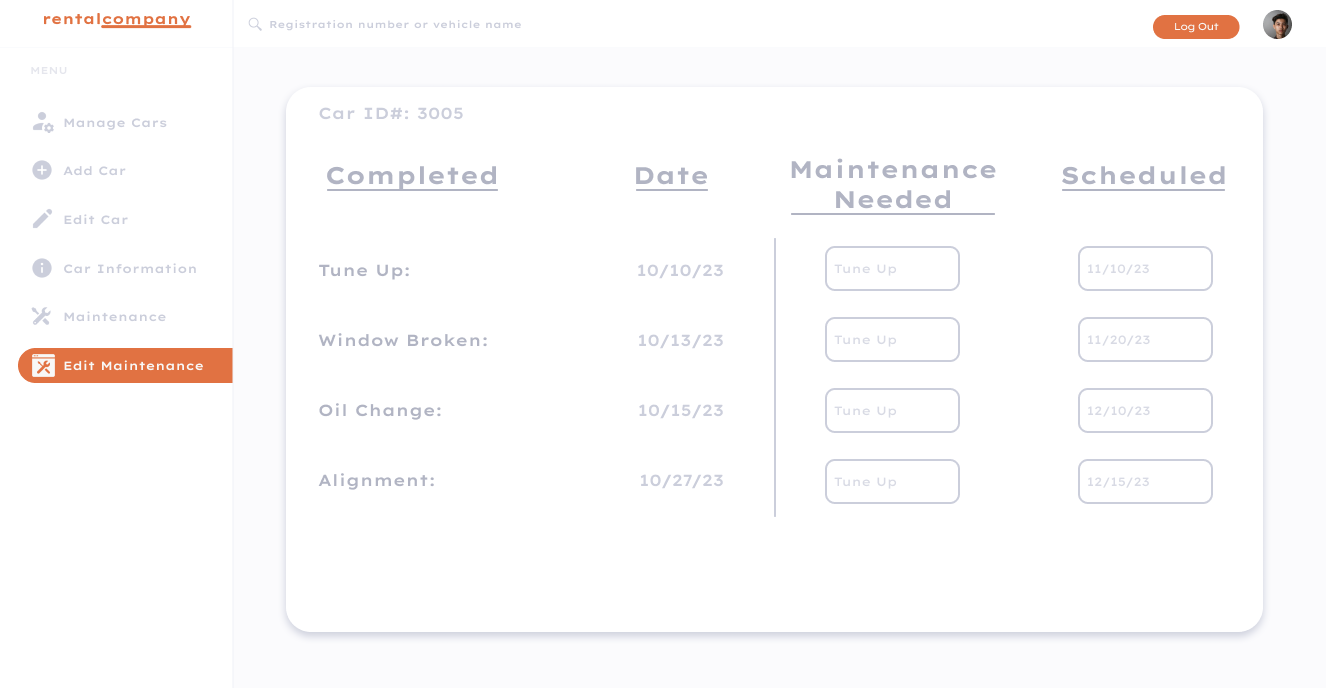
## Console - Car Information



## Console - Maintenance



## Console - Edit Maintenance



# Database Design

For the ERP system, we've employed MongoDB as our database of choice, given its agility and the schema-less nature, which provides the adaptability to manage data in a manner that aligns seamlessly with our varied and evolving use cases. While MongoDB brings a plethora of advantages to our system:

* Agility: MongoDB's NoSQL structure allows for rapid development cycles, effortlessly accommodating evolving data requirements.
* Schema-less Nature: Provides the flexibility to deal with data in a form that can be remodeled and manipulated as requirements evolve without a stringent predefined schema.

We've chosen to host our MongoDB instance on MongoDB Atlas, a fully-managed cloud database service that furnishes us with advantages critical to the seamless operation and sustainability of our ERP system:

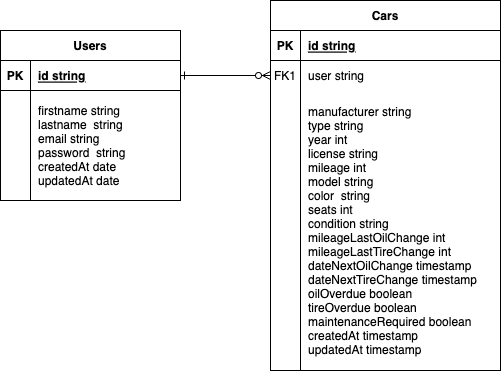
* Scalability: With MongoDB Atlas, our database can elastically scale in terms of storage and compute power, ensuring optimal performance even as our data and user load exponentially grow.
* Data Recovery: Leveraging automated backups and robust disaster recovery capabilities, MongoDB Atlas safeguards our data against unforeseen mishaps and ensures business continuity by minimizing downtime.
* Global Distribution: Through its global clusters feature, MongoDB Atlas allows us to distribute our data across regions, bringing it closer to our users and ensuring low-latency, high-throughput performance.
* Security: Implementing a myriad of security features such as VPC peering, IP whitelisting, and automated patching, MongoDB Atlas fortifies our data against unauthorized access and vulnerabilities.

Our commitment to an agile and scalable database design ensures that as the business processes and data requirements evolve, our database can adapt and scale in tandem, continuing to provide a robust foundation for our ERP system.

## Schema Design and Collections

Navigating deeper into our database structure, let’s see schema design and the foundational collections that facilitate our ERP’s core functionalities:

* User Collection: Managing user data, including credentials and personal information. The schema is designed to facilitate quick retrieval of user information and secure authentication, ensuring that user interactions with the system are secure and personalized.
* Car Collection: Holding pertinent details about cars. Designed to efficiently facilitate inventory management and enhance the user purchasing journey through quick, accurate data retrieval.



# Security Design

Our application underscores various key aspects of security design, each tailored to safeguard data, protect user privacy, and thwart potential malicious activities.

## Password Hashing

Password hashing is a quintessential security practice to ensure that user credentials are not stored in a vulnerable and decipherable format. Utilizing robust cryptographic algorithms, such as bcrypt, user passwords are hashed before being stored in the database. This mechanism not only protects the user’s original password from being exposed in case of a data breach but also verifies user credentials during login without ever revealing the actual password, fortifying user authentication processes and safeguarding user credentials against potential security threats.

## Session Management

Our application incorporates a robust session management system, meticulously designed to maintain a secure connection between the user and the server post-login. Utilizing JWT, the user’s authentication and authorization data are securely transmitted during interactions with the application.

## Data Validation

Data validation is an imperative defense against a myriad of security threats, such as SQL injections, XSS attacks, and more. Employing rigorous input validation mechanisms across all user interaction points, our application scrutinizes all user inputs through whitelist validation, type checks, and sanitation processes. By permitting only valid data to interact with the system, data validation serves as a frontline defense against malicious attempts to manipulate or corrupt database operations, thereby ensuring the integrity and reliability of the application's operations.

## Data Encryption

As a mechanism to fortify data security and protect user data at rest, our application employs comprehensive data encryption strategies. Leveraging advanced encryption algorithms, all sensitive user data stored within MongoDB is encrypted, effectively rendering the data useless in the hands of unauthorized entities. This encryption not only applies to user passwords but extends to any data deemed sensitive or personal, thereby establishing a robust security blanket across all stored data. This ensures that even in the eventuality of a data breach, the acquired data remains inaccessible and safeguarded against malicious exploitation.

# Business Logic and/or Key Algorithms

Within our car ERP system, we've designed a set of algorithms and business logic to ensure efficient data retrieval, filtering, and user-car management.

**Filter Algorithm:** This algorithm helps filter cars based on specific attributes such as Car license or Manufacturer, enabling more accessible access to entries.

**Car Maintenance Scheduler:** Schedules and tracks car maintenance.

Additional business logic, including user authentication, car addition, editing, and deletion, are also in place to ensure a seamless experience.

# Design Patterns

## Chain of Responsibility Pattern

In our application, the **Chain of Responsibility Pattern** has been implemented within the authentication middleware, facilitating a clean, organized, and scalable approach to user authentication. When an HTTP request aimed at secure endpoints hits our server, it traverses through a chain of middleware functions, each with its own distinct responsibility in the authentication process. The first middleware might check the presence of authentication token, passing the request along the chain upon validation, or rejecting it if the check fails. The subsequent middleware could decode and verify the token, retrieving the user information, and appending it to the request object for downstream use. Another in the line might ensure the user has the requisite permissions to access the desired resources. Each middleware function, serving as a link in the chain, takes a decision: either halt the process and respond with an error or pass the request further down the chain. This architectural approach enables each piece of the authentication and authorization logic to exist in isolated, testable, and maintainable units, simplifying debugging, extension, and management of the authentication flow, all while ensuring a stringent and secure user validation process.

# References

1. Vercel. (2023). Flow of the request from client to requested resource [Image]. Retrieved from <https://vercel.com/blog/behind-the-scenes-of-vercels-infrastructure>

# Glossary

1. RCM: Rental Car Management
2. Repo: Repository
3. PR: Pull Request
4. IDE: Integrated Development Environment
5. CI: Continuous Integration
6. CD: Continuous Deployment
7. JWT: JSON Web Tokens