

Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH



DOKUMENTACJA TECHNICZNA PROJEKTU „CAR RENTAL”

Kompleksowego systemu webowego obsługującego
wypożyczalnię samochodów

Warszawa, czerwiec 2021 r.

Spis treści

1.	<i>Wstęp</i>	5
2.	<i>Podział ról</i>	5
3.	<i>Architektura aplikacji</i>	6
3.1.	<i>Frontend</i>	6
3.1.1.	HTML5	6
3.1.2.	SCSS	6
3.1.3.	JavaScript.....	6
3.1.4.	React-Bootstrap.....	7
3.2.	<i>Serwer</i>	7
3.3.	<i>Baza danych</i>	8
3.3.1.	Marki samochodów	8
3.3.2.	Modele samochodów	8
3.3.3.	Samochody	9
3.3.4.	Użytkownicy.....	9
3.3.5.	Transakcje	9
4.	<i>Funkcjonalność (frontend)</i>	10
4.1.	<i>Belka nawigacyjna</i>	10
4.2.	<i>Strona tytułowa</i>	11
4.3.	<i>Katalog samochodów</i>	12
4.3.1.	Lista samochodów	12
4.3.2	Podgląd samochodu.....	13
4.4.	<i>Historia Transakcji</i>	13
4.4.1.	Lista transakcji	13
4.4.2	Podgląd konkretnego wypożyczenia.....	14
4.5.	<i>Zarządzanie katalogiem</i>	14
4.5.1	Moduł Zarządzania samochodami	14
4.5.2	Moduł Zarządzania modelami.....	15
4.5.3	Moduł Zarządzania markami.....	15
4.6.	<i>Profil użytkownika</i>	16
4.8.1	Podgląd danych.....	16
4.8.2	Edycja danych	16
4.7.	<i>Rejestracja</i>	17
4.7.1	Rejestracja użytkownika	17
4.7.2	Rejestracja administratora.....	17
4.8.	<i>Logowanie</i>	18
5.	<i>API</i>	19
5.1.	<i>Autoryzacja i uwierzytelnienie</i>	19
5.1.1.	Rejestracja użytkownika	19

5.1.2.	Rejestracja administratora	20
5.1.3.	Logowanie	20
5.1.4.	Aktualizacja danych	21
5.1.5.	Pobranie danych.....	21
5.2.	Katalog Marek Samochodów.....	22
5.2.1.	Pobranie wszystkich marek	22
5.2.2.	Pobranie jednej marki	22
5.2.3.	Dodanie nowej marki	23
5.2.4.	Edycja marki	23
5.2.5.	Usunięcie marki	24
5.3.	Katalog Modeli Samochodów.....	25
5.3.1.	Pobranie wszystkich modeli	25
5.3.2.	Pobranie jednego modelu	26
5.3.3.	Dodanie nowego modelu	26
5.3.4.	Edycja modelu	27
5.3.5.	Usunięcie modelu.....	27
5.4.	Katalog Samochodów.....	28
5.4.1.	Pobranie wszystkich samochodów	28
5.4.2.	Pobranie jednego samochodu	29
5.4.3.	Dodanie nowego samochodu	30
5.4.4.	Edycja samochodu.....	30
5.4.5.	Usunięcie samochodu.....	31
5.5.	Moduł Transakcji	32
5.5.1.	Zainicjowanie transakcji	32
5.5.2.	Zamknięcie transakcji	32
5.5.3.	Pobranie wszystkich transakcji użytkownika	33
5.5.4.	Pobranie wszystkich transakcji w systemie	34
5.5.5.	Pobranie jednej transakcji	35
5.6.	Moduł obsługi plików JPG	36
5.6.1.	Pobranie pliku z zasobów serwera (np. plik JPG).....	36
5.6.2.	Upload plików JPG	36
6.	Informacje dodatkowe	37
6.1.	Domyślne konta użytkowników	37
6.1.1.	Administrator	37
6.1.2.	Klient.....	37
6.2.	Domyślny katalog samochodów	37
6.3.	Domyślna historia transakcji	37
7.	Instrukcja instalacji	38
7.1.	Lokalnie	38
7.1.1.	Baza danych.....	38
7.1.2.	Serwer	38

7.1.3. Klient.....	38
7.2. Kontener.....	39
8. Podsumowanie.....	40

1. Wstęp

„Car Rental”, to kompleksowy system informatyczny, obsługujący wypożyczalnię samochodów. Jest to oprogramowanie w architekturze Klient-Serwer. Klient jest aplikacją webową, dzięki czemu do działania usługi nie jest wymagana instalacja – potencjalny klient potrzebuje tylko przeglądarki internetowej, aby założyć konto w serwisie i wypożyczyć swoje pierwsze auto.

Zarówno serwer, jak i klient „Car Rental” jest stworzony w języku programowania JavaScript, co znaczco obniża koszty utrzymania dla potencjalnego nabywcy – wystarczy być biegłym tylko w jednej technologii.

Poniżej znajduje się pełna dokumentacja techniczna projektu opisująca szczegółowo architekturę serwisu, funkcjonalność klienta webowego, API serwera aplikacji, jak i bazę danych przygotowaną i ujętą na potrzeby usługi.

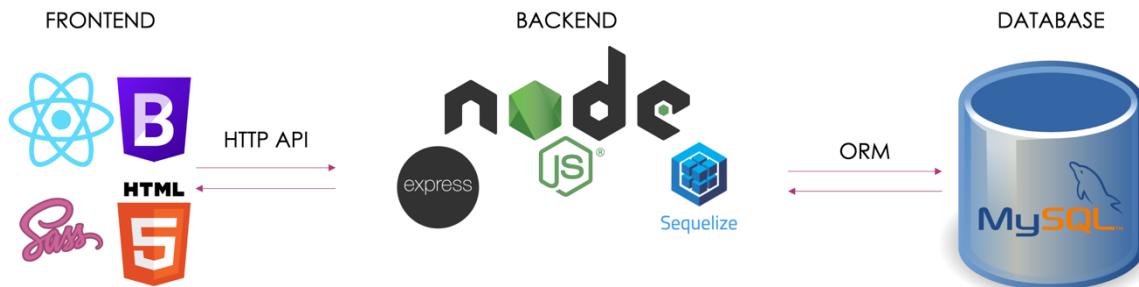
2. Podział ról

Zespół projektowy składał się z trzech osób. Zakres obowiązków był następujący:

- **Cały zespół** – zaprojektowanie funkcjonalności serwisu,
- **Maciej Krysiak** – zaprojektowanie architektury systemu, implementacja bazy danych, implementacja serwera aplikacji, przygotowanie RESTApi,
- **Łukasz Śleboda** – implementacja logiki biznesowej klienta aplikacji,
- **Michał Malinowski** – wygląd klienta aplikacji.

3. Architektura aplikacji

Aplikacja została podzielona na 3 główne moduły: Frontend (klienta), Backend (serwer) i bazę danych.



Rysunek 1 Architektura aplikacji

3.1. Frontend

Atrakcyjna i przejrzysta warstwa graficzna, stanowi w dzisiejszych czasach podstawę wszystkich poważnych aplikacji webowych. W związku z tym, twórcy aplikacji „car rental” zdecydowali się na użycie nowoczesnych i popularnych rozwiązań programistycznych.

3.1.1. HTML5

Jest wersją rozwijową języka HTML ([ang.](#) *HyperText Markup Language*) służącego do tworzenia dokumentów hipertekstowych. Powszechnie używany do tworzenia podstawowej funkcjonalności stron internetowych. Odpowiada głównie za strukturę, natomiast wygląd poszczególnych elementów modyfikowany jest za pomocą plików CSS (w projekcie zastąpionych plikami SCSS). W projekcie „Car rental” język HTML jest używany sporadycznie (<div>,
) ze względu na zastąpienie go językiem JSX (JavaScript XML).

3.1.2. SCSS

SCSS (inaczej SASS z ang. *syntactically awesome style sheets*) jest to preprocesor języka CSS pozwalający tworzyć arkusze stylów szybciej i dużo czytelniej. Plik zawierający kod SASS zapisywany jest z rozszerzeniem. sass bądź .scss i wzbogaca napisanie arkuszy stylów o dodatkowe funkcje takie jak zmienne, albo obliczenia. Dzięki temu pozwala skupić się na najważniejszych aspektach pracy, czyli tworzeniu. Wszystko kompilowane jest na lokalnym komputerze developera do postaci CSS'a gotowego do podpięcia na stronę.

3.1.3. JavaScript

JavaScript to język programowania, który umożliwia wdrożenie na stronie internetowej skomplikowanych elementów, dzięki którym strona ta może nie tylko wyświetlać statyczne informacje, ale również obsługiwać zmianę treści odpowiednio do sytuacji, wyświetlać interaktywne mapy i animacje grafiki 2D/3D, wyświetlać video itd. Jest to trzecia warstwa standardowych technologii internetowych, z których dwie (HTML i CSS). Dodatkowo w projekcie rozszerzono jego funkcjonalności za pomocą biblioteki React i framework'u

Bootstrap, a raczej ich połączonej formy – framework'u React-Bootstrap (w skrócie Reactstrap).

3.1.4. React-Bootstrap

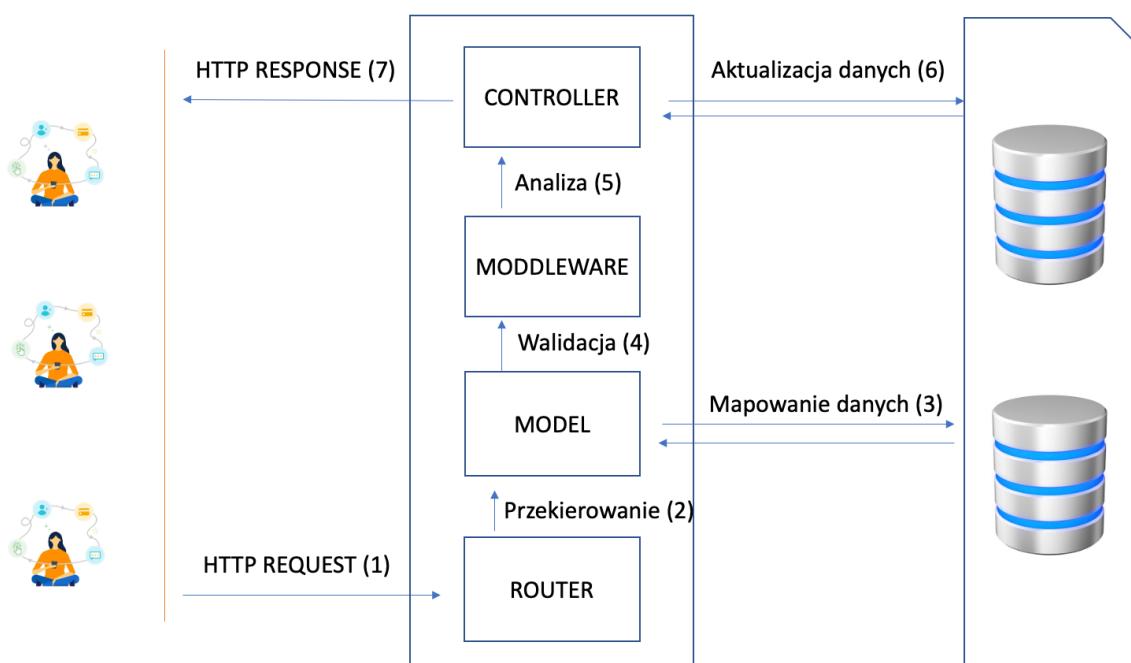
React-Bootstrap jest framework'iem zastępującym standardowy framework Bootstrap. Został opracowany specjalnie pod bibliotekę React, rozszerzając jej możliwości o te znane z oryginalnego Bootstrap'a. W projekcie zdecydowano się na zastosowanie powyższego framework'u ze względu na łatwość implementacji z API serwera, a także przyjemną dla oka warstwę wizualną.

3.2. Serwer

Serwer aplikacji został stworzony w technologii NodeJS, czyli w środowisku uruchomieniowym języka JavaScript. Działanie serwera HTTP zapewnia framework ExpressJS, a komunikację z bazą danych odsługuje system ORM Sequelize.

Projekt został podzielony na cztery główne moduły:

- **Routes** – moduł odpowiedzialny za routing RESTApi aplikacji, wszystkie linki URL, kierowanie ruchu przychodzącego do serwera przy pomocy frameworka ExpressJS,
- **Models** – moduł odpowiedzialny za komunikację bazy danych z serwerem NodeJS, klasy mapujące tabele MySQL na klasy JavaScript przy pomocy Sequelize ORM,
- **Middleware** – moduł odpowiedzialny za weryfikację uprawnień i walidację danych, na przykład sprawdzający poprawność tokenów użytkownika i duplikaty unikatowych danych (jak adres email użytkownika podczas rejestracji nowego klienta).
- **Controllers** – moduł odpowiedzialny za logikę serwera.



Rysunek 2 Analiza blokowa przetwarzania żądania HTTP w serwerze

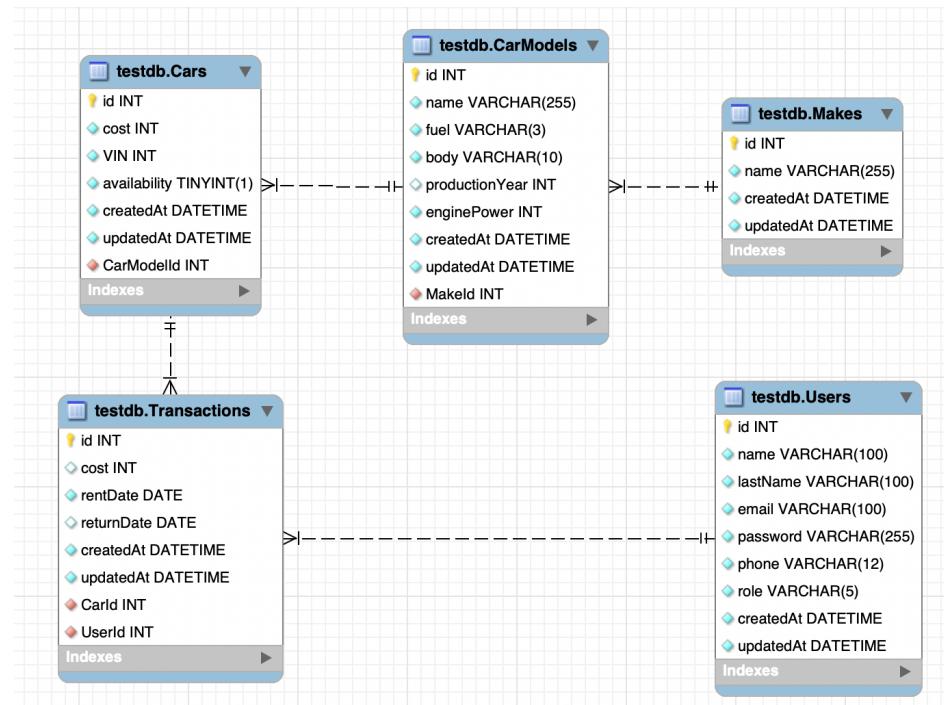
Dodatkowo:

- w folderze **/public/images** można znaleźć obrazki dostępne przez API,
- w pliku `.env` znajdują się wrażliwe dane konfigurujące dostęp do bazy danych.

3.3. Baza danych

W projekcie zdecydowano się na bazę MySQL z powodu jej popularności, multiplatformowości, darmowości, jak i pewnego doświadczenia w jej obsłudze, posiadany przez zespół programistyczny. Podczas produkcji oprogramowania używano lokalną, najnowszą wersję MySQL.

Sama baza składa się z 5 głównych tabel, w której skład wchodzi katalog pojazdów (3 tabele), użytkownicy systemu (1 tabela) i tabela transakcji.



Rysunek 3 Schemat bazy danych zastosowanej w projekcie

3.3.1. Marki samochodów

Tabela o nazwie „Makes” przechowuje katalog marek samochodów.

Atrybuty:

- id,
- nazwa.

3.3.2. Modele samochodów

Tabela o nazwie „CarModels” przechowuje katalog modeli samochodów.

Atrybuty:

- id,

- nazwa,
- paliwo,
- nadwozie,
- rok produkcji,
- moc silnika,
- id marki samochodu.

3.3.3. Samochody

Tabela o nazwie „Cars” przechowuje samochody dostępne do wypożyczenia w salonie.

Atrybuty:

- id,
- cena za dzień wypożyczenia,
- numer VIN,
- aktualna dostępność pojazdu,
- Id modelu samochodu.

3.3.4. Użytkownicy

Tabela o nazwie „Users” przechowuje dane użytkowników systemu.

Atrybuty:

- id,
- imię,
- nazwisko,
- email,
- hasło w postaci „posolonego” hasha,
- numer telefonu,
- rola (klient, admin itd.).

3.3.5. Transakcje

Tabela o nazwie „Transactions” przechowuje transakcje wypożyczenia pojazdu dokonane przez użytkowników.

Atrybuty:

- id,
- łączna cena wypożyczenia,
- data wypożyczenia,
- data oddania,
- Id pojazdu,
- Id użytkownika.

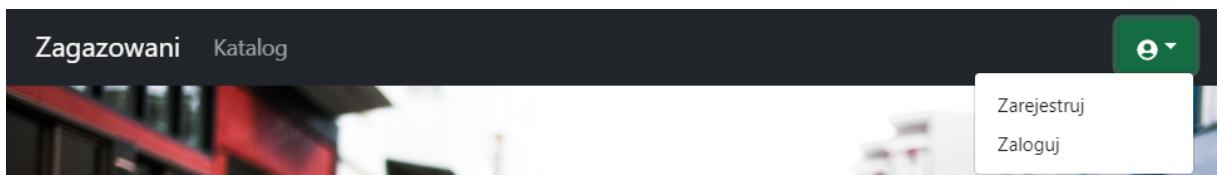
4. Funkcjonalność (frontend)

Aplikacja zapewnia pełną funkcjonalność, której wymaga prawdziwa wypożyczalnia samochodów. Poniżej zaprezentowane zostały jej moduły, oraz opis pełnej funkcjonalności.

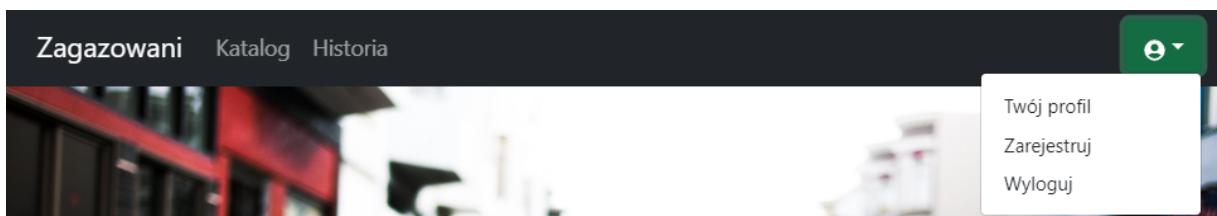
4.1. Belka nawigacyjna

Belka nawigacyjna umożliwia prostą i szybką nawigację po aplikacji. Pozwala ona za pomocą dostępnych przycisków dostać się do takich modułów jak:

- Zagazowani - Strona główna wypożyczalni,
- Katalog – Katalog samochodów w ofercie wypożyczalni,
- Historia – Historia wypożyczeń,
- Menu profilu:
 - Logowanie – Panel logowania do aplikacji,
 - Rejestracja - Panel zakładania nowego konta,
 - Twoje dane - Podgląd danych użytkownika



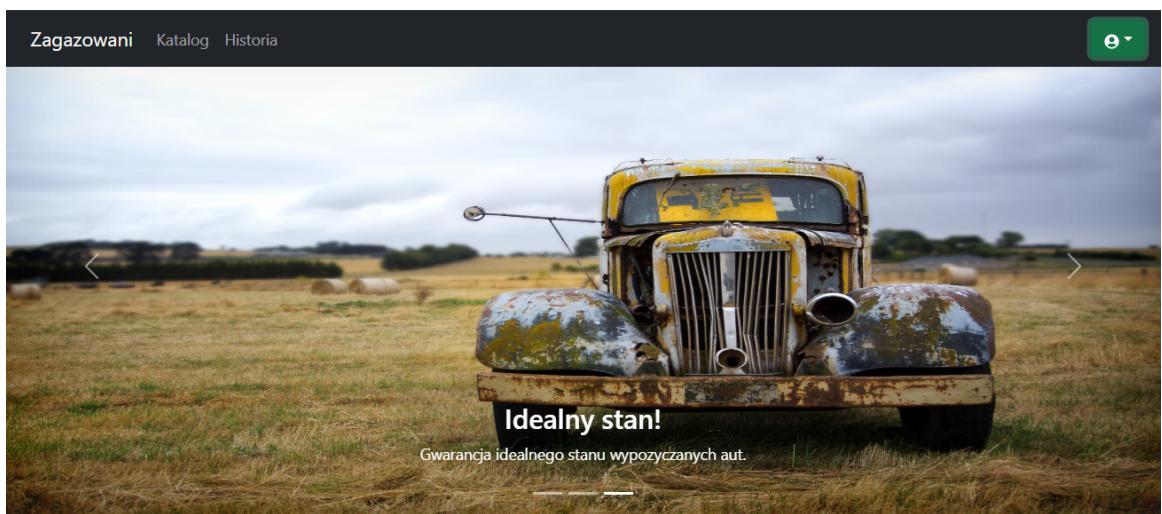
Rysunek 4 Belka nawigacyjna niezalogowanego użytkownika



Rysunek 5 Belka nawigacyjna zalogowanego użytkownika

4.2. Strona tytułowa

Strona tytułowa jest wizytówką i zaproszeniem do skorzystania z usług wypożyczalni. Prezentuje krótką historię naszej firmy, oferowane usługi oraz ciekawe grafiki.



Wymarzone auto na każdą kieszeń...

Wypożyczalnia zagazowani, oferuje wynajem dugo i krótko terminowy luksusowych samochodów osobowych. Jesteśmy firma z wieloletnim doświadczeniem, doceniona przez setki klientów z całej Europy. Kazdy znajdzie u nas auto dostosowane do jego indywidualnych potrzeb. Zapoznaj się z naszą ofertą i dołącz do zadowolonych klientów już dzisiaj!

Zagazowani © 2021 Zwierzaki
Powered by Zwierzaki

Rysunek 6 Strona powitalna

4.3. Katalog samochodów

Aplikacja pozwala zarówno zalogowanym, jak i niezalogowanym użytkownikom na zapoznanie się z gamą oferowanych samochodów.

4.3.1. Lista samochodów

Lista samochodów umożliwia zapoznanie się z pełną ofertą wypożyczalni. W widoku tabeli możliwy jest przegląd wszystkich oferowanych przez wypożyczalnię samochodów, które możemy dowolnie filtrować po ich dostępności. Tabela prezentuje podstawowe dane samochodu, takie jak: Marka, model, typ nadwozia, rok produkcji, moc, oraz najważniejszą rzeczą, jaką jest cena za dzień wynajmu. Przycisk podgląd przenosi nas do szczegółowego podglądu pojazdu.

The screenshot shows a mobile application interface for car rentals. At the top, there are navigation items: 'Zagazowani' and 'Katalog'. On the right, there is a green button with a dropdown icon. Below the header, a banner says 'Wybierz swój wymarzony samochód!' (Choose your dream car!). Underneath the banner, there is a section titled 'Status wypożyczenia' with three radio buttons: 'Dostępne' (selected), 'Niedostępne', and 'Wszystkie'. A table follows, displaying three rows of car data:

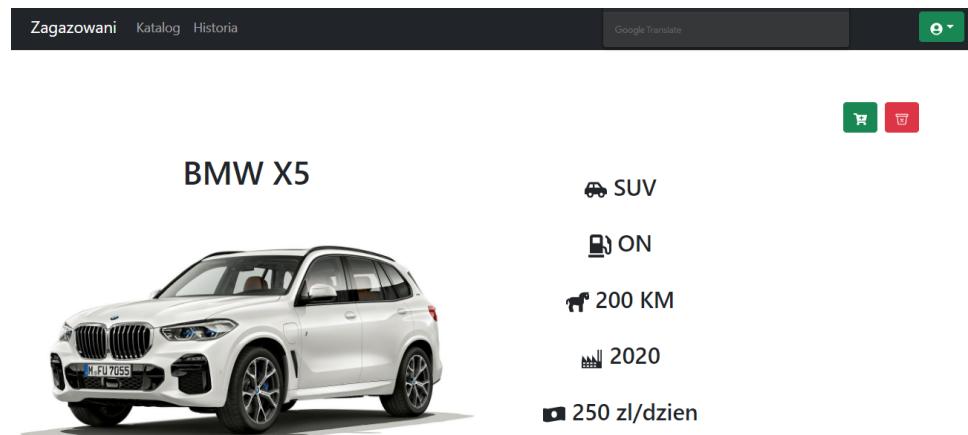
Id	Marka	Model	Typ	Rok	Moc	Cena	
1	BMW	X5	SUV	2020	200	250	Podgląd
2	AUDI	Q7	SUV	2019	220	275	Podgląd
3	PORSCHE	Cayenne GTS Coupe	SUV	2021	650	999	Podgląd

At the bottom of the screen, there is a footer with the text 'Zagazowani © 2021 Zwierzaki' and 'Powered by Zwierzaki'.

Rysunek 7 Lista samochodów do wynajęcia

4.3.2 Podgląd samochodu

Szczegółowy podgląd pojazdu pozwala użytkownikowi na zapoznanie się ze szczegółowymi danymi technicznymi oraz obejrzenie fotografii pojazdu. W przypadku zalogowania na konto standardowego użytkownika i sytuacji, w której wybrany samochód będzie w obecnej chwili wolny, dostępny będzie również przycisk „Wypożycz”, za pomocą którego zawierana jest umowa z wypożyczalnią. W przypadku zalogowania na konto administratora istnieje również możliwość wycofania nieużywanego obecnie pojazdu z oferty za pomocą przycisku “usuń samochód z oferty”.



4.4. Historia Transakcji

Każdy użytkownik aplikacji ma możliwość zapoznania się z historycznymi wynajmami dokonanymi w naszej wypożyczalni.

4.4.1. Lista transakcji

Lista transakcji pozwala standardowemu użytkownikowi na podgląd wszystkich zrealizowanych z wypożyczalnią transakcji. Zarówno historycznych, jak i transakcji, które wciąż są w toku. Perspektywa administratora pozwala dodatkowo na podgląd transakcji wszystkich klientów wypożyczalni. Przyciski podgląd w wierszach tabeli przenoszą użytkownika do szczegółowego podglądu transakcji.

Nr	Marka	Model	Silnik	Od	Do	Łączny koszt	Użytkownik	
1	BMW	X5	ON	2021-05-15	2021-05-18	1250	admin	Podgląd
2	AUDI	Q7	ON	2021-05-20	2021-05-25	2000	example@user.com	Podgląd

Rysunek 9 Lista transakcji – administrator

4.4.2 Podgląd konkretnego wypożyczenia

Szczegółowy podgląd transakcji pozwala na zapoznanie się z jej detalami. Widnieje tam między innymi informacja o dacie wypożyczenia, zwrotu oraz o całkowitym koszcie wynajmu. Perspektywa administratora pozwala dodatkowo na zmianę statusu samochodu na dostępny po odebraniu go od klienta.



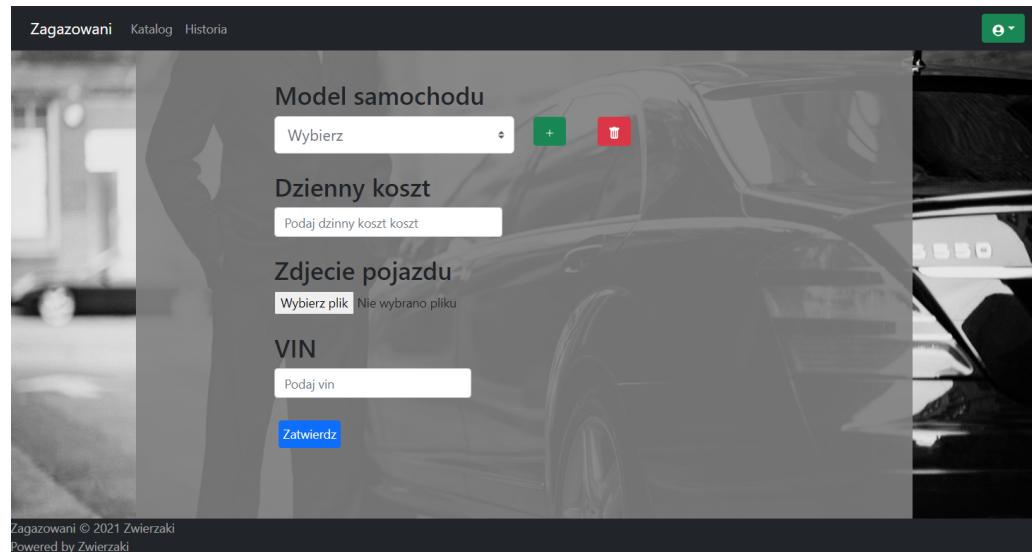
Rysunek 10 Podgląd wypożyczenia

4.5. Zarządzanie katalogiem

Administrator posiada możliwość zarządzania katalogiem dostępnych pojazdów, marek oraz modeli w wypożyczalni. Do tego celu wykorzystywane są trzy moduły do których administrator może dostać się poprzez zielony przycisk „+” w katalogu samochodów.

4.5.1 Moduł Zarządzania samochodami

Moduł zarządzania samochodami pozwala na dodanie nowego pojazdu do oferty wypożyczalni. Odbiera się to poprzez wybór z listy konkretnego modelu, podanie dziennego kosztu wynajmu w złotówkach oraz wprowadzeniu numeru VIN i przekazaniu z komputera odpowiedniej grafiki. Podczas wyboru modelu z listy istnieje również możliwość jego usunięcia z wykorzystaniem przycisku „Usuń wybrany model”, bądź też dodania jeśli takowy nas interesujący nie znajduje się na dostępnej liście. Przycisk „Dodaj nowy model” przenosi do niżej opisanego modułu oddawania nowego modelu.



Rysunek 11 Panel dodawania samochodu

4.5.2 Moduł Zarządzania modelami

Moduł zarządzania modelami pozwala na utworzenie nowego modelu samochodu. Wymaga to poprawnego wypełnienia wszystkich pól w formularzu i zatwierdzenia guzikiem „Dodaj model”. Podczas wyboru marki modelu istnieje możliwość usunięcia marki, bądź też przejścia do modułu dodawania nowej marki.

Dodaj nowy model samochodu

Nazwa modelu

Rodzaj paliwa

Typ nadwozia

Rok produkcji

Silnik

Marka samochodu + -

Zatwierdź

Rysunek 12 Panel dodawania modelu

4.5.3 Moduł Zarządzania markami

Moduł zarządzania markami pozwala na dodanie nowej marki poprzez wypełnienie formularza i zatwierdzenie danych przyciskiem „Dodaj markę”.

Dodaj nową markę samochodu

Nowa Marka

Dodaj markę

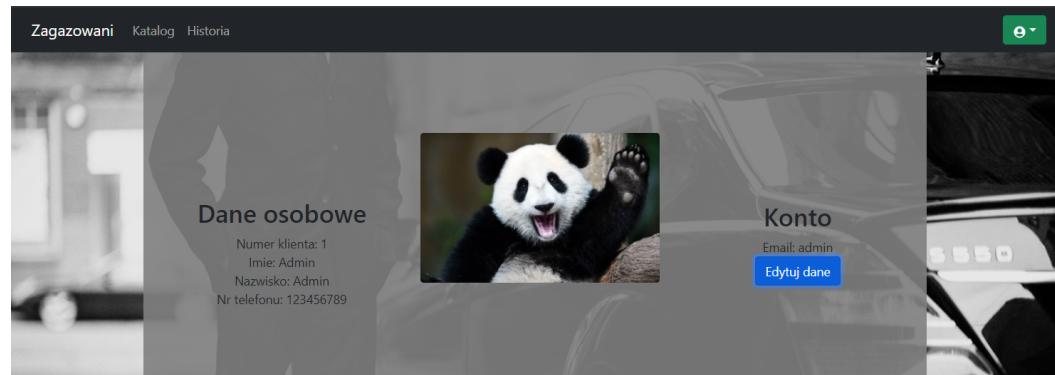
Rysunek 13 Panel dodawania marki

4.6. Profil użytkownika

Aplikacja umożliwia użytkownikowi podgląd swoich danych w celu ich walidacji, oraz możliwość ich edycji.

4.8.1 Podgląd danych

Podgląd danych użytkownika pozwala na podgląd takich danych jak: Numer klienta, Imię i Nazwisko oraz numer kontaktowy. Przycisk „Edytuj dane” otwiera moduł pozwalający na ich edycję.



Rysunek 14 Podgląd danych osobowych

4.8.2 Edycja danych

Panel edycji danych pozwala na modyfikację takich danych jak: imię, nazwisko, numer kontaktowy oraz hasło. Procedura polega na wypełnieniu formularza, a w nim podania aktualnego hasła dla potwierdzenia tożsamości i zatwierdzenia zmian guzikiem aktualizuj.

A screenshot of a data update form titled 'Uaktualnij swoje dane'. The form fields include:

- 'Aktualne Hasło': A text input field with placeholder 'Podaj hasło'.
- 'Nowe hasło' and 'Powtórz nowe Hasło': Two text input fields, each with a placeholder 'Podaj hasło'.
- 'Imię': A text input field with placeholder 'Admin'.
- 'Nazwisko': A text input field with placeholder 'Admin'.
- 'Telefon': A text input field with placeholder '758952425', which is highlighted with a blue border.
- A blue 'Aktualizuj' (Update) button at the bottom.

Rysunek 15 Panel aktualizacji danych

4.7. Rejestracja

Aplikacja zapewnia pracę z wieloma kontami użytkowników i administratorów.

4.7.1 Rejestracja użytkownika

Rejestracja nowego użytkownika odbywa się poprzez wypełnienie formularza. W nim należy podać takie parametry jak: email, imię, nazwisko, numer kontaktowy i hasło. Konto zostaje założone po zatwierdzeniu danych przyciskiem zarejestruj.

4.7.2 Rejestracja administratora

Rejestracja nowego administratora może zostać przeprowadzona jedynie z poziomu konta innego administratora. Procedura rejestracji jest identyczna jak w przypadku rejestracji standardowego użytkownika.

The screenshot shows a modal window titled "Dodaj nowego administratora". It contains fields for "Email", "Hasło" (Password), and "Potwierdź hasło" (Confirm password). Below these are fields for "Imię" (First name), "Nazwisko" (Last name), and "Telefon" (Phone). A blue "Zarejestruj" (Register) button is at the bottom.

Rysunek 16 Panel rejestracji

4.8. Logowanie

Logowanie do aplikacji odbywa się poprzez wypełnienie formularza, który wymaga od użytkownika podania adresu email oraz hasła. W przypadku podania błędnych danych zwrócony zostanie stosowny komunikat.

The image shows a user interface for logging in. At the top, the word "Logowanie" is centered above a small "x" icon. Below this, there are two input fields: one labeled "Email address" with a placeholder "Enter email" and another labeled "Password" with a placeholder "Password". At the bottom of the form is a large blue rectangular button with the white text "Zaloguj".

Rysunek 17 Pandel logowania

5. API

API używane w aplikacji zostało zaprojektowane zgodnie z założeniem RESTApi. Podzielone zostało na 6 głównych kategorii.

5.1. Autoryzacja i uwierzytelnienie

Moduł odpowiada za autoryzację i uwierzytelnienie użytkownika, jak i pozwala uzyskać dostęp do jego danych.

5.1.1. Rejestracja użytkownika

- a) Metoda: **POST**,
- b) URL: **/api/auth/signup**
- c) Ciało żądania:
name: string,
lastName: string,
phone: number(12),
password: string,
email: string
- d) Nagłówki:
 - **Content-Type:** ‘application/json’
- e) Wymagania dodatkowe:
 - Unikatowy adres email
- f) Odpowiedź (**JSON**):
 - W przypadku sukcesu (**200 OK**):
message: “User was registered successfully”
 - W przypadku niepowodzenia (statusy **4xx, 5xx**):
message: Treść komunikatu błędu

5.1.2. Rejestracja administratora

- a) Metoda: **POST**,
- b) URL: **/api/auth/signup/admin**
- c) Ciało żądania:

```
  name: string,  
  lastName: string,  
  phone: number(12),  
  password: string,  
  email: string
```

- d) Nagłówki:
 - **Content-Type**: 'application/json'
 - **x-access-token**: token
- e) Wymagania dodatkowe:
 - Unikatowy adres email,
 - Uprawnienia administratora,
 - Token.
- f) Odpowiedź (**JSON**):
 - W przypadku sukcesu (**200 OK**):
message: "User was registered successfully"
 - W przypadku niepowodzenia (statusy **4xx, 5xx**):
message: Treść komunikatu błędu

5.1.3. Logowanie

- a) Metoda: **POST**,
- b) URL: **/api/auth/signin**
- c) Ciało żądania:

```
  email: string,  
  password: string,
```
- d) Nagłówki:
 - **Content-Type**: 'application/json'
- e) Wymagania dodatkowe:
 - Email i hasło zarejestrowanego użytkownika
- f) Odpowiedź (**JSON**):
 - W przypadku sukcesu (**200 OK**):

```
{  
    "id": 1,  
    "email": "admin",  
    "name": "Admin",  
    "lastName": "Admin",  
    "phone": "123456789",  
    "role": "admin",  
    "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiawF0IjoxNjI0MTk3MjUyLCJleHAiOjE2MjQyODM2NTJ9.  
BKON1-ugFCaJC0Gc07AVL7kS0FVqdIXcg_HE70PMMTA"  
}
```

Rysunek 18 Logowanie użytkownika - 200 OK

- W przypadku niepowodzenia (statusy **4xx, 5xx**):
message: Treść komunikatu błędu

5.1.4. Aktualizacja danych

- a) Metoda: **PUT**,
- b) URL: **/api/auth/update**
- c) Ciało żądania:

```
password: string,  
name: string,  
lastName: string,  
phone: number(12),  
currentPassword: string
```

- d) Nagłówki:
 - **Content-Type**: 'application/json'
 - **x-access-token**: token
- e) Wymagania dodatkowe:
 - Token
- f) Odpowiedź (**JSON**):
 - W przypadku sukcesu (**200 OK**):
message: „User data updated”
 - W przypadku niepowodzenia (statusy **4xx, 5xx**):
message: Treść komunikatu błędu

5.1.5. Pobranie danych

- a) Metoda: **GET**,
- b) URL: **/api/auth/data**
- c) Ciało metody: -
- d) Nagłówki:
 - **x-access-token**: token
- e) Wymagania dodatkowe:
 - Token
- f) Odpowiedź (**JSON**):
 - W przypadku sukcesu (**200 OK**):

```
{  
    "user": {  
        "id": 1,  
        "name": "Admin",  
        "lastName": "Admin",  
        "email": "admin",  
        "phone": "123456789",  
        "role": "admin"  
    }  
}
```

Rysunek 19 Pobranie danych użytkownika - 200 OK

- W przypadku niepowodzenia (statusy **4xx, 5xx**):
message: Treść komunikatu błąd

5.2. Katalog Marek Samochodów

Moduł odpowiada za obsługę katalogu marek samochodów. Jego głównym zadaniem jest obsługa tabeli „Makes” z bazy danych.

5.2.1. Pobranie wszystkich marek

- a) Metoda: **GET**,
- b) URL: **/api/catalog/makes**
- c) Ciało żądania: -
- d) Nagłówki: -
- e) Wymagania dodatkowe: -
- f) Odpowiedź (**JSON**):
 - W przypadku sukcesu (**200 OK**):

```
{  
    "make": [  
        {  
            "id": 1,  
            "name": "BMW"  
        },  
        {  
            "id": 2,  
            "name": "AUDI"  
        }  
    ]  
}
```

Rysunek 20 Pobranie marek samochodów - 200 OK

- W przypadku niepowodzenia (statusy **4xx, 5xx**):
message: Treść komunikatu błędu

5.2.2. Pobranie jednej marki

- a) Metoda: **GET**,
- b) URL: **/api/catalog/makes/<make_id>**
- c) Ciało żądania: -
- d) Nagłówki: -
- e) Wymagania dodatkowe: -
- f) Odpowiedź (**JSON**):
 - W przypadku sukcesu (**200 OK**):

```
{  
    "make": {  
        "id": 1,  
        "name": "BMW"  
    }  
}
```

Rysunek 21 Pobranie marki samochodu - 200 OK

- W przypadku niepowodzenia (statusy **4xx, 5xx**):
message: Treść komunikatu błędu

5.2.3. Dodanie nowej marki

- a) Metoda: **POST**,
- g) URL: **/api/catalog/makes**
- h) Ciało żądania:
name: string
- i) Nagłówki:
 - **Content-Type**: 'application/json'
 - **x-access-token**: token
- j) Wymagania dodatkowe:
 - nazwa nowej marki musi być unikatowa,
 - token,
 - użytkownik musi być administratorem.
- k) Odpowiedź (**JSON**):
 - W przypadku sukcesu (**200 OK**):
message: „Make created successfully!”
 - W przypadku niepowodzenia (statusy **4xx, 5xx**):
message: Treść komunikatu błędu

5.2.4. Edycja marki

- a) Metoda: **PUT**,
- b) URL: **/api/catalog/makes/<make_id>**
- c) Ciało żądania:
name: string
- d) Nagłówki:
 - **Content-Type**: 'application/json'
 - **x-access-token**: token
- e) Wymagania dodatkowe:
 - nazwa nowej marki musi być unikatowa,
 - token,
 - użytkownik musi być administratorem.
- f) Odpowiedź (**JSON**):
 - W przypadku sukcesu (**200 OK**):
message: „Make updated!”
 - W przypadku niepowodzenia (statusy **4xx, 5xx**):
message: Treść komunikatu błędu

5.2.5. Usunięcie marki

- g) Metoda: **DELETE**,
- h) URL: **/api/catalog/makes/<make_id>**
- i) Ciało żądania: -
- j) Nagłówki:
 - **Content-Type**: 'application/json'
 - **x-access-token**: token
- k) Wymagania dodatkowe:
 - token,
 - użytkownik musi być administratorem,
 - marka nie może być używana w żadnym modelu samochodu.
- l) Odpowiedź (**JSON**):
 - W przypadku sukcesu (**200 OK**):
message: „Make deleted!”
 - W przypadku niepowodzenia (statusy **4xx, 5xx**):
message: Treść komunikatu błędu

5.3. Katalog Modeli Samochodów

Moduł odpowiada za obsługę katalogu modeli samochodów. Jego głównym zadaniem jest obsługa tabeli „CarMakes” z bazy danych.

5.3.1. Pobranie wszystkich modeli

- a) Metoda: **GET**,
- b) URL: **/api/catalog/models**
- c) Ciało żądania: -
- d) Nagłówki: -
- e) Wymagania dodatkowe: -
- f) Odpowiedź (**JSON**):
 - W przypadku sukcesu (**200 OK**):



```
{  
    "model": [  
        {  
            "id": 1,  
            "name": "X5",  
            "fuel": "ON",  
            "body": "SUV",  
            "productionYear": 2020,  
            "enginePower": 200,  
            "MakeId": 1,  
            "Make": {  
                "id": 1,  
                "name": "BMW"  
            }  
        },  
        {  
            "id": 2,  
            "name": "Q7",  
            "fuel": "ON",  
            "body": "SUV",  
            "productionYear": 2019,  
            "enginePower": 220,  
            "MakeId": 2,  
            "Make": {  
                "id": 2,  
                "name": "Audi"  
            }  
        }  
    ]  
}
```

Rysunek 22 Pobranie wszystkich modeli samochodów - 200 OK

- W przypadku niepowodzenia (statusy **4xx, 5xx**):
message: Treść komunikatu błędu

5.3.2. Pobranie jednego modelu

- a) Metoda: **GET**,
- b) URL: **/api/catalog/models/<model_id>**
- c) Ciało żądania: -
- d) Nagłówki: -
- e) Wymagania dodatkowe: -
- f) Odpowiedź (**JSON**):

- W przypadku sukcesu (**200 OK**):

```
{  
    "model": {  
        "id": 1,  
        "name": "X5",  
        "fuel": "ON",  
        "body": "SUV",  
        "productionYear": 2020,  
        "enginePower": 200,  
        "MakeId": 1,  
        "Make": {  
            "id": 1,  
            "name": "BMW"  
        }  
    }  
}
```

Rysunek 23 Pobranie modelu samochodu - 200 OK

- W przypadku niepowodzenia (statusy **4xx, 5xx**):
message: Treść komunikatu błędu

5.3.3. Dodanie nowego modelu

- a) Metoda: **POST**,
- b) URL: **/api/catalog/models**
- c) Ciało żądania:
 - name:** string
 - fuel:** string (PB/ON/GAZ)
 - body:** string (SUV, LIMUZINE, WAGON)
 - enginePower:** integer
 - productionYear:** integer,
 - MakeId:** integer
- d) Nagłówki:
 - **Content-Type:** 'application/json'
 - **x-access-token:** token
- e) Wymagania dodatkowe:
 - cały zestaw parametrów modelu nie może się powtórzyć,
 - token,
 - użytkownik musi być administratorem.
- f) Odpowiedź (**JSON**):
 - W przypadku sukcesu (**200 OK**):
message: „Car model created successfully!”
 - W przypadku niepowodzenia (statusy **4xx, 5xx**):
message: Treść komunikatu błędu

5.3.4. Edycja modelu

- a) Metoda: **PUT**,
- b) URL: **/api/catalog/models/<model_id>**
- c) Ciało żądania:
 - name**: string
 - fuel**: string (PB/ON/GAZ)
 - body**: string (SUV, LIMUZINE, WAGON)
 - enginePower**: integer
 - productionYear**: integer,
 - MakelId**: integer
- d) Nagłówki:
 - **Content-Type**: 'application/json'
 - **x-access-token**: token
- e) Wymagania dodatkowe:
 - cały zestaw parametrów modelu nie może się powtórzyć,
 - token,
 - użytkownik musi być administratorem.
- f) Odpowiedź (**JSON**):
 - W przypadku sukcesu (**200 OK**):
message: „Car model updated!”
 - W przypadku niepowodzenia (statusy **4xx, 5xx**):
message: Treść komunikatu błędu

5.3.5. Usunięcie modelu

- a) Metoda: **DELETE**,
- b) URL: **/api/catalog/models/<model_id>**
- c) Ciało żądania: -
- d) Nagłówki:
 - **Content-Type**: 'application/json'
 - **x-access-token**: token
- e) Wymagania dodatkowe:
 - token,
 - użytkownik musi być administratorem,
 - model nie może być używany w żadnym samochodzie.
- f) Odpowiedź (**JSON**):
 - W przypadku sukcesu (**200 OK**):
message: „Car model deleted!”
 - W przypadku niepowodzenia (statusy **4xx, 5xx**):
message: Treść komunikatu błędu

5.4. Katalog Samochodów

Moduł odpowiada za obsługę katalogu samochodów. Jego głównym zadaniem jest obsługa tabeli „Cars” z bazy danych.

5.4.1. Pobranie wszystkich samochodów

- a) Metoda: **GET**,
- b) URL: **/api/catalog/cars<?key=value>**
- c) Parametry zapytania:
 - bez parametrów – pobranie wszystkich samochodów,
 - **availability=1** – pobranie tylko aktualnie dostępnych samochodów,
 - **availability=0** – pobranie tylko aktualnie niedostępnych samochodów.
- d) Ciało żądania: -
- e) Nagłówki: -
- f) Wymagania dodatkowe: -
- g) Odpowiedź (**JSON**):
 - W przypadku sukcesu (**200 OK**):

```
{  
    "car": [  
        {  
            "id": 1,  
            "cost": 250,  
            "VIN": 2334433,  
            "availability": true,  
            "CarModelId": 1,  
            "CarModel": {  
                "id": 1,  
                "name": "X5",  
                "fuel": "ON",  
                "body": "SUV",  
                "productionYear": 2020,  
                "enginePower": 200,  
                "MakeId": 1,  
                "Make": {  
                    "id": 1,  
                    "name": "BMW"  
                }  
            },  
            {  
                "id": 2,  
                "cost": 275,  
                "VIN": 2334434,  
                "availability": false,  
                "CarModelId": 2,  
                "CarModel": {  
                    "id": 2,  
                    "name": "X6",  
                    "fuel": "ON",  
                    "body": "SUV",  
                    "productionYear": 2021,  
                    "enginePower": 220,  
                    "MakeId": 2,  
                    "Make": {  
                        "id": 2,  
                        "name": "BMW"  
                    }  
                }  
            }  
        }  
    ]  
}
```

Rysunek 24 Pobranie wszystkich samochodów - 200 OK

- W przypadku niepowodzenia (statusy **4xx, 5xx**):
message: Treść komunikatu błędu

5.4.2. Pobranie jednego samochodu

- a) Metoda: **GET**,
- b) URL: **/api/catalog/cars/<car_id>**
- c) Ciało żądania: -
- d) Nagłówki: -
- e) Wymagania dodatkowe: -
- f) Odpowiedź (**JSON**):

- W przypadku sukcesu (**200 OK**):

```
{  
    "car": {  
        "id": 1,  
        "cost": 250,  
        "VIN": 2334433,  
        "availability": true,  
        "CarModelId": 1,  
        "CarModel": {  
            "id": 1,  
            "name": "X5",  
            "fuel": "ON",  
            "body": "SUV",  
            "productionYear": 2020,  
            "enginePower": 200,  
            "MakeId": 1,  
            "Make": {  
                "id": 1,  
                "name": "BMW"  
            }  
        }  
    }  
}
```

Rysunek 25 Pobranie samochodu - 200 OK

- W przypadku niepowodzenia (statusy **4xx, 5xx**):
message: Treść komunikatu błędu

5.4.3. Dodanie nowego samochodu

- a) Metoda: **POST**,
- b) URL: **/api/catalog/cars**
- c) Ciało żądania:
 - cost: integer** (*cena za dzień wypożyczenia*)
 - VIN: integer**
 - availability: boolean**
 - CarModelId: integer**
- d) Nagłówki:
 - **Content-Type**: ‘application/json’
 - **x-access-token**: token
- e) Wymagania dodatkowe:
 - numer VIN samochodu musi być unikatowy,
 - token,
 - użytkownik musi być administratorem.
- f) Odpowiedź (**JSON**):
 - W przypadku sukcesu (**200 OK**):
message: „Car created successfully!”
 - W przypadku niepowodzenia (statusy **4xx, 5xx**):
message: Treść komunikatu błędu

5.4.4. Edycja samochodu

- a) Metoda: **PUT**,
- b) URL: **/api/catalog/cars/<car_id>**
- c) Ciało żądania:
 - cost: integer** (*cena za dzień wypożyczenia*)
 - VIN: integer**
 - availability: boolean**
 - CarModelId: integer**
- d) Nagłówki:
 - **Content-Type**: ‘application/json’
 - **x-access-token**: token
- e) Wymagania dodatkowe:
 - numer VIN samochodu musi być unikatowy,
 - token,
 - użytkownik musi być administratorem.
- f) Odpowiedź (**JSON**):
 - W przypadku sukcesu (**200 OK**):
message: „Car updated!”
 - W przypadku niepowodzenia (statusy **4xx, 5xx**):
message: Treść komunikatu błędu

5.4.5. Usunięcie samochodu

- a) Metoda: **DELETE**,
- b) URL: **/api/catalog/cars/<car_id>**
- c) Ciało żądania: -
- d) Nagłówki:
 - **Content-Type**: 'application/json'
 - **x-access-token**: token
- e) Wymagania dodatkowe:
 - token,
 - użytkownik musi być administratorem,
 - samochód nie może być wypożyczony teraz ani nigdy wcześniej.
- f) Odpowiedź (**JSON**):
 - W przypadku sukcesu (**200 OK**):
message: „Car deleted!”
 - W przypadku niepowodzenia (statusy **4xx, 5xx**):
message: Treść komunikatu błędu

5.5. Moduł Transakcji

Moduł odpowiada za obsługę transakcji (wypożyczenia samochodów). Jego głównym zadaniem jest obsługa tabeli „Transactions” z bazy danych.

5.5.1. Zainicjowanie transakcji

- a) Metoda: **POST**,
- b) URL: **/api/transactions**
- c) Ciało żądania: -
 - CarId**: string,
 - rentDate**: string (YYYY-MM-DD)
- d) Nagłówki:
 - **Content-Type**: ‘application/json’
 - **x-access-token**: token
- e) Wymagania dodatkowe:
 - token,
 - transakcja zostaje przypisana do właściciela tokena
 - samochód musi być dostępny
- f) Odpowiedź (**JSON**):
 - W przypadku sukcesu (**200 OK**):
message: „Transaction created!”
 - W przypadku niepowodzenia (statusy **4xx, 5xx**):
message: Treść komunikatu błędu

5.5.2. Zamknięcie transakcji

- g) Metoda: **PUT**,
- h) URL: **/api/transactions/<transaction_id>**
- i) Ciało żądania: -
 - cost**: integer,
 - returnDate**: string (YYYY-MM-DD)
- j) Nagłówki:
 - **Content-Type**: ‘application/json’
 - **x-access-token**: token
- k) Wymagania dodatkowe:
 - token,
 - uprawnienia administratora
- l) Odpowiedź (**JSON**):
 - W przypadku sukcesu (**200 OK**):
message: „Transaction closed!”
 - W przypadku niepowodzenia (statusy **4xx, 5xx**):
message: Treść komunikatu błędu

5.5.3. Pobranie wszystkich transakcji użytkownika

- a) Metoda: **GET**,
- b) URL: **/api/transactions**
- c) Ciało żądania: -
- d) Nagłówki:
 - **Content-Type**: 'application/json'
 - **x-access-token**: token
- e) Wymagania dodatkowe:
 - token
 - system zwróci transakcje właściciela wysłanego tokena
- f) Odpowiedź (**JSON**):
 - W przypadku sukcesu (**200 OK**):

```
{
  "transaction": [
    {
      "id": 1,
      "cost": 1250,
      "rentDate": "2021-05-15",
      "returnDate": "2021-05-18",
      "CarId": 1,
      "UserId": 1,
      "User": {
        "email": "admin"
      },
      "Car": {
        "id": 1,
        "cost": 250,
        "VIN": 2334433,
        "availability": true,
        "CarModelId": 1,
        "CarModel": {
          "id": 1,
          "name": "X5",
          "fuel": "ON",
          "body": "SUV",
          "productionYear": 2020,
          "enginePower": 200,
          "MakId": 1,
          "Make": {
            "id": 1,
            "name": "BMW"
          }
        }
      },
      {
        "id": 2,
        "cost": 2000,
      }
    }
  ]
}
```

Rysunek 26 Pobranie wszystkich transakcji użytkownika - 200 OK

- W przypadku niepowodzenia (statusy **4xx, 5xx**):
message: Treść komunikatu błędu

5.5.4. Pobranie wszystkich transakcji w systemie

- a) Metoda: **GET**,
- b) URL: **/api/transactions/all**
- c) Ciało żądania: -
- d) Nagłówki:
 - **Content-Type**: 'application/json'
 - **x-access-token**: token
- e) Wymagania dodatkowe:
 - token,
 - żądanie musi zostać wykonane przez administratora
- f) Odpowiedź (**JSON**):
 - W przypadku sukcesu (**200 OK**):

```
{
  "transaction": [
    {
      "id": 1,
      "cost": 1250,
      "rentDate": "2021-05-15",
      "returnDate": "2021-05-18",
      "CarId": 1,
      "UserId": 1,
      "User": {
        "email": "admin"
      },
      "Car": {
        "id": 1,
        "cost": 250,
        "VIN": 2334433,
        "availability": true,
        "CarModelId": 1,
        "CarModel": {
          "id": 1,
          "name": "X5",
          "fuel": "ON",
          "body": "SUV",
          "productionYear": 2020,
          "enginePower": 200,
          "MakId": 1,
          "Make": {
            "id": 1,
            "name": "BMW"
          }
        }
      },
      {
        "id": 2,
        "cost": 2000,
        "rentDate": "2021-05-16",
        "returnDate": "2021-05-19",
        "CarId": 2,
        "UserId": 2,
        "User": {
          "email": "user"
        },
        "Car": {
          "id": 2,
          "cost": 300,
          "VIN": 123456789,
          "availability": false,
          "CarModelId": 2,
          "CarModel": {
            "id": 2,
            "name": "Q7",
            "fuel": "OFF",
            "body": "SUV",
            "productionYear": 2021,
            "enginePower": 300,
            "MakId": 2,
            "Make": {
              "id": 2,
              "name": "Audi"
            }
          }
        }
      }
    ]
  ]
}
```

Rysunek 27 Pobranie transakcji wszystkich użytkowników - 200 OK

- W przypadku niepowodzenia (statusy **4xx, 5xx**):
message: Treść komunikatu błędu

5.5.5. Pobranie jednej transakcji

- g) Metoda: **GET**,
- h) URL: **/api/transactions/<transaction_id>**
- i) Ciało żądania: -
- j) Nagłówki:
 - **Content-Type**: 'application/json'
 - **x-access-token**: token
- k) Wymagania dodatkowe:
 - token,
 - numer transakcji musi istnieć
 - żądanie musi zostać wykonane przez właściciela transakcji lub administratora
- l) Odpowiedź (**JSON**):
 - W przypadku sukcesu (**200 OK**):

```
{  
    "transaction": {  
        "id": 1,  
        "cost": 1250,  
        "rentDate": "2021-05-15",  
        "returnDate": "2021-05-18",  
        "CarId": 1,  
        "UserId": 1,  
        "User": {  
            "email": "admin"  
        },  
        "Car": {  
            "id": 1,  
            "cost": 250,  
            "VIN": 2334433,  
            "availability": true,  
            "CarModelId": 1,  
            "CarModel": {  
                "id": 1,  
                "name": "X5",  
                "fuel": "ON",  
                "body": "SUV",  
                "productionYear": 2020,  
                "enginePower": 200,  
                "MakId": 1,  
                "Make": {  
                    "id": 1,  
                    "name": "BMW"  
                }  
            }  
        }  
    }  
}
```

Rysunek 28 Pobranie jednej transakcji - 200 OK

- W przypadku niepowodzenia (statusy **4xx, 5xx**):
message: Treść komunikatu błędu

5.6. Moduł obsługi plików JPG

Moduł odpowiada za obsługę plików JPG. Jego głównym celem jest przechowywanie plików JPG samochodów oferowanych przez wypożyczalnię.

5.6.1. Pobranie pliku z zasobów serwera (np. plik JPG)

- a) Metoda: **GET**,
- b) URL: **/public/images/<file_name>**
- c) Ciało żądania: -
- d) Nagłówki: -
- e) Wymagania dodatkowe: -
- f) Odpowiedź (**IMAGE**):
 - W przypadku sukcesu (**200 OK**):
OBRAZEK
 - W przypadku niepowodzenia (**JSON**) (statusy **4xx, 5xx**):
message: Treść komunikatu błędu

5.6.2. Upload plików JPG

- a) Metoda: **POST**,
- b) URL: **/upload/jpg**
- c) Ciało żądania:
 - file:** image.jpg
 - filename:** string (np. „car_1”)
- d) Nagłówki: -
- e) Wymagania dodatkowe: -
 - Musi to być plik JPG
 - Nazwa pliku musi być unikatowa (w przeciwnym przypadku zostanie nadpisany istniejący plik)
- f) Odpowiedź (**IMAGE**):
 - W przypadku sukcesu (**200 OK**):
message: „Successfully uploaded file!”
 - W przypadku niepowodzenia (**JSON**) (statusy **4xx, 5xx**):
message: Treść komunikatu błędu

6. Informacje dodatkowe

Aplikacja posiada wstępnie zapełnioną bazę danych w celu prezentacji możliwości systemu. Aby odtworzyć bazę, należy skorzystać z dostarczonego skryptu **/database/script.sql**.

6.1. Domyślne konta użytkowników

Serwer aplikacji oferuje domyślne konta użytkowników. Są to dane ogólnodostępne i zaraz po pierwszym zalogowaniu zaleca się jak najszybsze zmienienie hasła.

6.1.1. Administrator

Konto administratora posiada pełne uprawnienia do obsługi aplikacji, jak i umożliwia zarejestrowanie nowego administratora w portalu. Aby zalogować się z pełnymi uprawnieniami, należy skorzystać z poniższych poświadczeń:

- Emial: admin
- Hasło: admin

6.1.2. Klient

Konto klienta posiada oferuje uprawnienia klienta imitujące potencjalnego użytkownika portalu. Zachowuje się, jak konto założone z poziomu aplikacji przez nowego użytkownika. Aby zalogować się z uprawnieniami klienta, należy skorzystać z poniższych poświadczeń:

- Emial: example@user.com
- Hasło: 123

6.2. Domyślny katalog samochodów

Platforma zapewnia wstępnie 3 samochodów wraz z plikami JPG prezentującymi pojazdy.

6.3. Domyślna historia transakcji

Domyślnie każdy z użytkowników (example@user.com i admin) posiadają po jednej zakończonej transakcji wypożyczenia samochodu.

7. Instrukcja instalacji

Cały system oparty jest o środowisko JavaScript. Do jego uruchomienia nie jest wymagany żaden edytor tekstu. Poniżej znajduje się instrukcja instalacji oprogramowania.

7.1. Lokalnie

System działa na bazie MySQL. W celu poprawnego skonfigurowania systemu, należy:

7.1.1. Baza danych

System działa na bazie MySQL. W celu poprawnego skonfigurowania systemu, należy:

- Zainstalować i skonfigurować MySQL Serwer w najnowszej wersji (do dalszej konfiguracji serwera będzie potrzebne:
 - Nazwa serwera bazy danych,
 - Nazwa użytkownika,
 - Hasło.
- Pliki instalacyjne można znaleźć pod adresem:
<https://dev.mysql.com/downloads/mysql/>.
- Wywołać skrypt tworzący bazę danych (**/LOKALNIE/database/script.sql**). Można to zrobić na przykład poprzez CLI MySQL lub używając programu MySQL Workbench.

7.1.2. Serwer

Serwer jest oparty o technologię NodeJS. W celu poprawnej konfiguracji programu należy:

- Zainstalować środowisko NodeJS, które można znaleźć na przykład pod linkiem:
<https://nodejs.org/en/download/>
- W folderze **/LOKALNIE/server** wywołać polecenie terminala **npm install**.
- Wejść w plik **/LOKALNIE/server/.env** (w tym celu należy odkryć pliki ukryte w systemie operacyjnym, lub skorzystać z edytorów tekstu typu *Atom* lub *VisualStudio Code*)
- W pliku .env wprowadzić dane konfiguracyjne bazy danych (**DB_HOST**, **DB_USER**, **DB_PASSWORD**, zgodnie z ustawnionymi wcześniej parametrami w punkcie 7.1.).
- W folderze **/LOKALNIE/server** wywołać polecenie terminala **npm run start**.
- Server powinien uruchomić się pod domyślnymi adresami:
 - <http://localhost:3000>
 - http://<lp_komputera>:3000

7.1.3. Klient

Klient oparty jest o technologię ReactJS. W celu poprawnej konfiguracji programu należy:

- Zainstalować NodeJS, jeśli nie zrobiono tego wcześniej (punkt 7.2.)
- W folderze **/LOKALNIE/client** wywołać polecenie terminala **npm install**.
- W pliku **/LOKALNIE/client/src/config/api_endpoint.json** wprowadzić adres serwera. Domyślnie jest to adres <http://localhost:3000>
- W folderze **/LOKALNIE/client** wywołać polecenie terminala **npm run start2**

- Server powinien uruchomić się pod domyślnym adresem:
 - <http://localhost:3001>

7.2. Kontener

Projekt został również przystosowany do uruchomienia w środowisku witalizacyjnym Docker. Zakłada się, że użytkownik posiada zainstalowane środowisko Docker na maszynie docelowej i połączenie z Internetem w momencie instalacji. W celu uruchomienia projektu, należy:

- W ukrytym pliku **/DOCKER/.env**:
 - Ustawić wartość stałej **REACT_APP_SERVER_IP** na lokalny adres IP komputera, na którym będzie uruchamiana aplikacja (np. 192.168.1.10),
 - Wartość stałej **REACT_APP_SERVER_PORT** musi być równoznaczna z wartością stałej **NODE_LOCAL_PORT** (najlepiej nie zmieniać, chyba że na zadanym porcie jest aktywna inna usługa).
- W folderze **/DOCKER** wywołać polecenie terminala **docker-compose up**. Polecenie to pobierze wszystkie potrzebne moduły i biblioteki, a następnie uruchomi projekt.

Strona internetowa jest teraz dostępna pod adresami:

- <http://localhost:4001>,
- http://<adres_ip_komputera>:4001.

W przypadku użycia adresu wraz z adresem IP komputera, strona jest osiągalna z całej sieci lokalnej (na przykład z telefonu).

8. Podsumowanie

Projekt „Car Rental”, to aplikacja typu Klient-Serwer napisany całkowicie w technologiach i frameworkach języka JavaScript. Aplikacja, ze względu na swój webowy charakter, zapewnia multiplatformowość. Usługa, niezależnie od systemu operacyjnego (Windows, Unix, iOS, Android), czy urządzenia końcowego (PC, tablet, smartfon) do działania wymaga tylko połączenia z Internetem.

Projekt ten, poza oferowaną funkcjonalnością, pokazuje też potęgę aplikacji webowych i technologii JavaScript. Aby przygotować aplikację internetową, nie jest potrzebna droga maszyna developerska, a do samego pisania kodu wystarczy darmowe IDE, jak Visual Studio Code. Dzięki zastosowaniu języka skryptowego, nie trzeba czekać na każdorazową komplikację programu, a więc wprowadzanie zmian czy to po stronie Front-, jak i Backendu dokonuje się w mgnieniu oka.

Do debugowania API aplikacji niezastąpiony okazał się Postman – oprogramowanie pozwalające na wywoływanie zapytań HTTP, rozbudowane o dowolne nagłówki i ciało żądania.

Najtrudniejsze okazało się stworzenie samej strony internetowej. W projekcie nie użyto gotowych szablonów, co okazało się błędem – podejście zakładające rozbudowę i edycję już istniejących template’ów znacznie uprościłoby i przyśpieszyłoby pracę, a sam wygląd stron byłby znacznie przystępniejszy wizualnie dla współczesnego odbiorcy.