

Kierunek: **Informatyka techniczna**

**PROJEKT**  
**BAZY DANYCH 2**

**System bazodanowy wspierający firmę zajmującą się  
wypożyczaniem samochodów.**

AUTORZY:

Dawid Drozd  
Indeks: 252762

Władysław Nowak  
Indeks: 252700

PROWADZĄCY ZAJĘCIA:

Dr inż. Robert Wójcik, K<sub>30</sub>W<sub>04</sub>D<sub>03</sub>

OCENA PRACY:

# Spis treści

<b>Spis rysunków</b>	<b>4</b>
<b>Spis listingów</b>	<b>6</b>
<b>1. Wstęp</b>	<b>7</b>
1.1. Cel projektu	7
1.2. Zakres projektu	7
<b>2. Analiza wymagań</b>	<b>8</b>
2.1. Opis działania i funkcje systemu	8
2.2. Wymagania funkcjonalne	8
2.2.1. Diagram przypadków użycia	9
2.2.2. Scenariusze wybranych przypadków użycia	9
2.3. Wymagania niefunkcjonalne	11
2.3.1. Wykorzystywane technologie i narzędzia	11
2.3.2. Wymagania dotyczące bezpieczeństwa systemu	11
2.4. Przyjęte założenia projektowe	11
2.4.1. Założenia architektoniczne przyjęte podczas realizacji systemu	11
2.4.2. Wykorzystywane technologie, narzędzia projektowania oraz implemen-	
tacji systemu	11
2.4.3. Schemat komunikacji, struktura systemu	12
<b>3. Projekt systemu</b>	<b>13</b>
3.1. Projekt bazy danych	13
3.1.1. Analiza rzeczywistości	13
3.1.2. Model logiczny	13
3.1.3. Model fizyczny i ograniczenia integralności danych	15
3.1.4. Inne elementy schematu - widoki	16
3.2. Projekt aplikacji użytkownika	17
3.2.1. Architektura aplikacji i diagramy projektowe	17
3.2.2. Interfejs graficzny i struktura menu	18
3.2.3. Projekt wybranych funkcji systemu	20
3.2.4. Metoda podłączenia do bazy danych - integracja z bazą danych	21
<b>4. Implementacja systemu</b>	<b>22</b>
4.1. Realizacja bazy danych	22
4.1.1. Tworzenie tabel	22
4.1.2. Tworzenie widoków	23
4.1.3. Implementacja wyzwalaczy	25
4.2. Realizacja serwera bazy danych	25
4.2.1. Model encji	26
4.2.2. Serwis autentykacji	27
4.2.3. Serwis dostępu do danych	29
4.2.4. Implementacja komunikacji z serwerem	32
4.3. Realizacja elementów aplikacji	33

---

4.3.1. Autoryzacja . . . . .	33
4.3.2. Implementacja interfejsu dostępu do bazy danych . . . . .	34
<b>5. Testowanie systemu . . . . .</b>	<b>35</b>
5.1. Instalacja systemu . . . . .	35
5.1.1. Instalacja i uruchamianie systemu zarządzania bazą danych . . . . .	35
5.1.2. Instalacja i uruchamianie serwera bazy danych . . . . .	35
5.1.3. Instalacja i uruchamianie serwera aplikacji . . . . .	35
5.2. Testowanie opracowanych funkcji systemu . . . . .	36
5.2.1. Testowanie funkcji zrealizowanych w panelu logowania użytkownika . .	36
5.2.2. Testowanie funkcji zrealizowanych w panelu logowania pracownika . .	39
5.2.3. Testowanie funkcji zrealizowanych w panelu użytkownika . . . . .	41
5.2.4. Testowanie funkcji zrealizowanych w panelu pracownika . . . . .	42
5.3. Wnioski z przeprowadzonych testów . . . . .	48
<b>6. Podsumowanie . . . . .</b>	<b>49</b>
<b>Literatura . . . . .</b>	<b>50</b>

# Spis rysunków

2.1. Diagram przypadków użycia. . . . .	9
2.2. Schemat komunikacji w systemie. . . . .	12
3.1. Schemat logiczny projektowanej bazy danych. . . . .	14
3.2. Schemat fizyczny projektowanej bazy danych. . . . .	15
3.3. Widok dostępne samochody . . . . .	16
3.4. Widok przedstawiający wypożyczone samochody . . . . .	17
3.5. Strona logowania . . . . .	18
3.6. Strona rejestracji . . . . .	18
3.7. Strona główna (użytkownik) . . . . .	19
3.8. Panel administratora . . . . .	19
3.9. Okno dodawania/edycji pojazdu . . . . .	20
3.10. Przykładowa ścieżka żądania dodania nowego samochodu . . . . .	21
5.1. Logowanie użytkownika z poprawnymi danymi. . . . .	36
5.2. Logowanie użytkownika z poprawnymi danymi. - wynik . . . . .	36
5.3. Logowanie użytkownika z niepoprawną nazwą - wynik. . . . .	37
5.4. Logowanie użytkownika z niepoprawnym hasłem - wynik. . . . .	37
5.5. Zawartość tabeli <i>UZYTKOWNICY</i> w bazie danych przed rejestracją nowego użytkownika. . . . .	38
5.6. Wpisane dane w panelu rejestracji użytkownika. . . . .	38
5.7. Zawartość tabeli <i>UZYTKOWNICY</i> w bazie danych po rejestracji nowego użytkownika. . . . .	38
5.8. Logowanie pracownika z poprawnymi danymi. . . . .	39
5.9. Logowanie pracownika z poprawnymi danymi. - wynik . . . . .	39
5.10. Logowanie pracownika z niepoprawną nazwą użytkownika - wynik . . . . .	40
5.11. Logowanie pracownika z niepoprawnym hasłem - wynik. . . . .	40
5.12. Logowanie użytkownika z niewystarczającym poziomem uprawnień. . . . .	41
5.13. Wygląd panelu użytkownika po zalogowaniu. . . . .	41
5.14. Zawartość widoku <i>DOSTEPNE_SAMOCHODY</i> w bazie danych. . . . .	42
5.15. Wygląd panelu pracownika po zalogowaniu. . . . .	42
5.16. Zawartość tabeli <i>SAMOCHODY</i> w bazie danych. . . . .	43
5.17. Panel pracownika przed usunięciem samochodu. . . . .	43
5.18. Zawartość tabeli <i>SAMOCHODY</i> w bazie danych przed usunięciem samochodu. . . . .	44
5.19. Panel pracownika po usunięciu samochodu. . . . .	44
5.20. Zawartość tabeli <i>SAMOCHODY</i> w bazie danych po usunięciu samochodu. . . . .	45
5.21. Panel pracownika przed modyfikacją samochodu. . . . .	45
5.22. Zawartość tabeli <i>SAMOCHODY</i> w bazie danych przed modyfikacją samochodu . . . . .	45
5.23. Dane dotyczące samochodu, które mają zostać zapisane po modyfikacji. . . . .	46
5.24. Panel pracownika po modyfikacji samochodu. . . . .	46
5.25. Zawartość tabeli <i>SAMOCHODY</i> w bazie danych po modyfikacji samochodu. . . . .	46
5.26. Panel pracownika przed dodaniem nowego samochodu. . . . .	47

5.27. Zawartość tabeli <i>SAMOCHODY</i> w bazie danych przed dodaniem nowego samochodu.	47
5.28. Menu dodawania samochodu z wpisanymi atrybutami dodawanego samochodu. . .	47
5.29. Panel pracownika po dodaniu nowego samochodu. . . . .	48
5.30. Zawartość tabeli <i>SAMOCHODY</i> w bazie danych po dodaniu nowego samochodu. .	48

# Spis listingów

3.1. Przykładowy obiekt opisujący samochód . . . . .	20
4.1. Kod definiujący tabelę <i>MODELE</i> . . . . .	22
4.2. Kod definiujący tabelę <i>WYPOZYCZALNIE</i> . . . . .	22
4.3. Kod definiujący tabelę <i>SAMOCHODY</i> . . . . .	22
4.4. Kod definiujący tabelę <i>KLIENCI</i> . . . . .	23
4.5. Kod definiujący tabelę <i>WYPOZYCZENIA</i> . . . . .	23
4.6. Kod definiujący tabelę <i>LOG_WYPOZYCZENIA</i> . . . . .	23
4.7. Kod definiujący tabelę <i>UZYTKOWNICY</i> . . . . .	23
4.8. Kod definiujący widok <i>DOSTEPNE_SAMOCHODY</i> . . . . .	23
4.9. Kod definiujący widok <i>NIEDOSTEPNE_SAMOCHODY</i> . . . . .	24
4.10. Kod definiujący widok <i>WYPOZYCZONE_SAMOCHODY</i> . . . . .	25
4.11. Kod definiujący wyzwalacze dla tabeli <i>LOG_WYPOZYCZEN</i> . . . . .	25
4.12. Kod definiujący model encji <i>KLIENCI</i> . . . . .	26
4.13. Implementacja serwisu autentykacji. . . . .	27
4.14. Implementacja serwisu dostępu do danych. . . . .	29
4.15. Implementacja komunikacji z serwerem. . . . .	32
4.16. Metoda realizująca autoryzację użytkownika. . . . .	34

# Rozdział 1

## Wstęp

### 1.1. Cel projektu

Celem projektu jest zaprojektowanie i implementacja bazy danych, serwera bazy danych, oraz interfejsu użytkownika umożliwiających zarządzanie siecią wypożyczalni samochodów. Baza danych powinna być udostępniana klientom oraz pracownikom poprzez stronę WWW. W ramach projektu należało także zaprojektować podstawowy system autoryzacji, aby klienci mieli tylko ograniczony dostęp do bazy danych oraz, aby osoby nieupoważnione nie miały jakiegokolwiek dostępu do zasobów wypożyczalni.

### 1.2. Zakres projektu

Proces wykonywania projektu można podzielić na kilka najważniejszych części: analiza wymagań, projekt systemu, jego implementacja oraz ostatni etap — testowanie systemu.

W pierwszej kolejności wymagane było przeprowadzenie analizy wymagań systemu. W tej części projektowej wymagane było stworzenie oraz przedstawienie diagramów oraz scenariuszy przypadków użycia, opisać wykorzystywane technologie oraz wymagania dotyczące bezpieczeństwa systemu. Bardzo ważną częścią tego etapu było też przedstawienie ogólnych założeń architektonicznych — między innymi schematu komunikacji.

Następnym etapem było stworzenie projektu systemu, czyli zaprojektowanie bazy danych — na podstawie założeń projektowych stworzony został model logiczny (system podzielony został na encje i relacje między nimi), model fizyczny oraz wprowadzone zostały widoki. W ramach tego etapu stworzony został również projekt aplikacji użytkownika: opisana została jej architektura, interfejs graficzny oraz opisana została metoda podłączenia do bazy danych.

Najważniejszą częścią projektu był etap implementacji. Etap ten można podzielić na 3 najważniejsze części: realizacja bazy danych, czyli instalacja oraz odpowiednia jej konfiguracja; realizacja serwera bazy danych, czyli stworzenie serwera, który mógłby komunikować się bezpośrednio z bazą danych, zajmował by się też autoryzacją oraz przyjmowałby żądania aplikacji użytkownika; oraz ostatnia część: stworzenie interfejsu graficznego oraz aplikacji użytkownika.

Ostatnim etapem było przeprowadzenie testów, czyli zweryfikowanie, czy projekt systemu bazodanowego spełnia przyjęte założenia projektowe oraz czy nie posiada błędów implementacyjnych.

# Rozdział 2

## Analiza wymagań

### 2.1. Opis działania i funkcje systemu

System będzie wykorzystywał relacyjną bazę danych i umożliwiał zarządzanie samochodami dostępnymi w ofercie sieci wypożyczalni samochodów. Dostęp do bazy danych możliwy będzie z poziomu przeglądarki internetowej komunikującej się z bazą danych za pomocą pośredniczącego serwera WWW. Główną funkcją systemu będzie umożliwienie wykonywania operacji odczytu, modyfikacji, zapisu oraz usunięcia danych w bazie. Możliwość wykonywania poszczególnych operacji ograniczona będzie w zależności od przynależności użytkownika do danej grupy użytkowników z określonymi uprawnieniami (np. pracownik, klient).

### 2.2. Wymagania funkcjonalne

#### Zarządzanie samochodami

- Przeglądanie informacji o samochodach.
- Dodanie nowego samochodu.
- Usunięcie istniejącego samochodu.
- Aktualizacja danych dotyczących samochodu.
- Wypożyczenie samochodu.
- Odbiór samochodu od klienta.

#### Zarządzanie klientami

- Przeglądanie informacji o klientach.
- Dodanie nowego klienta.
- Usunięcie istniejącego klienta.
- Aktualizacja danych istniejącego klienta.

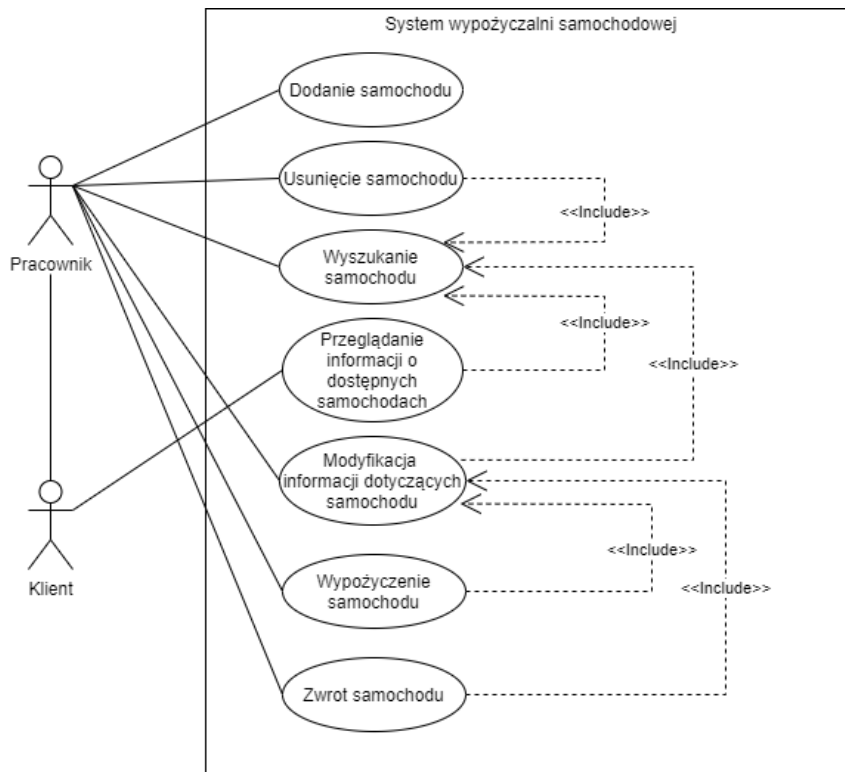
#### Zarządzanie wypożyczalniami

- Przeglądanie informacji o wypożyczalniach.
- Dodanie nowej wypożyczalni.
- Usunięcie istniejącej wypożyczalni.
- Aktualizacja danych dotyczących wypożyczalni



### 2.2.1. Diagram przypadków użycia

Na rysunku 2.1 przedstawiono diagram przypadków użycia systemu.



Rys. 2.1: Diagram przypadków użycia.

### 2.2.2. Scenariusze wybranych przypadków użycia

Użyte skróty:

PU - przypadek użycia

WS - warunki wstępne

WK - warunki końcowe

#### PU Dodanie samochodu

Cel: Dodanie nowego samochodu do oferty wypożyczalni

WS: Inicjalizacja przez kliknięcie przycisku "Dodaj samochód".

WK: Dodanie samochodu o podanych parametrach do oferty wypożyczalni.

Przebieg:

1. Należy podać atrybuty: nazwa samochodu, model, rok produkcji, cena.
2. Dodanie do bazy danych nowego samochodu o zadanych atrybutach.

#### PU Usunięcie samochodu

Cel: Usunięcie samochodu z oferty wypożyczalni

WS: Inicjalizacja przez kliknięcie przycisku "Usuń samochód".

WK: Usunięcie samochodu z listy pojazdów w ofercie.

Przebieg:

1. Należy podać atrybuty pozwalające zidentyfikować konkretny pojazd.
2. Należy wywołać PU Wyszukanie samochodu i sprawdzić, czy pojazd istnieje.
3. Jeśli pojazd istnieje, należy usunąć go z bazy danych.

### **PU Wyszukanie samochodu**

Cel: Wyszukanie samochodu w ofercie wypożyczalni.

WS: Inicjalizacja przez kliknięcie przycisku "Wyszukaj samochód"

WK: Zwrócenie informacji o znalezionym pojeździe.

Przebieg:

1. Należy podać atrybuty pozwalające zidentyfikować konkretny pojazd.
2. Jeśli pojazd o podanych parametrach istnieje, zwrócić informacje o nim, w przeciwnym razie wyświetlić komunikat "Nie znaleziono pojazdu".

### **PU Przeglądanie informacji o dostępnych samochodach**

Cel: Zwrócenie listy dostępnych samochodów.

WS: Inicjalizacja przez uruchomienie programu przez klienta.

WK: Zwrócenie listy dostępnych pojazdów.

Przebieg:

1. Należy wywołać PU Wyszukanie samochodu i wyszukać wszystkie pojazdy które nie są aktualnie wypożyczone.
2. Należy wyświetlić listę dostępnych pojazdów.

### **PU Modyfikacja informacji dotyczących samochodu**

Cel: Zmodyfikowanie danych wybranego pojazdu.

WS: Inicjalizacja przez kliknięcie przycisku "modyfikuj pojazd"

WK: Zmodyfikowanie danych pojazdu.

Przebieg:

1. Należy wywołać PU Wyszukanie samochodu.
2. Jeżeli samochód zostanie znaleziony, należy podać parametry pojazdu, które mają ulec modyfikacji, oraz ich nowe wartości, w przeciwnym razie należy wyświetlić komunikat "Pojazd nie został znaleziony".
3. Należy wyświetlić zmodyfikowane dane pojazdu.

### **PU Wypożyczenie samochodu**

Cel: Zmiana statusu samochodu na: „wypożyczone”, dodanie nowego wypożyczenia.

WS: Przypadek użycia wywołany przez pracownika na życzenie klienta. Inicjalizacja przez kliknięcie przycisku "Wypożycz".

WK: Ustawienie statusu wypożyczanego pojazdu na "wypożyczony", dodanie nowego wypożyczenia.

Przebieg:

1. Należy wywołać PU Modyfikacja informacji dotyczących samochodu.
2. Należy zaktualizować status pojazdu na „wypożyczone”.

### **PU Zwrot samochodu**

CEL: Zmiana statusu samochodu na: „dostępny”. Zmiana statusu wypożyczenia na "nieaktywne".

WS: Przypadek użycia wywołany przez pracownika na życzenie klienta. Inicjalizacja przez kliknięcie przycisku "Przyjmij zwrot".

WK: Zmiana statusu samochodu na: „zwrócone”. Zmiana statusu wypożyczenia na "nieaktywne".

Przebieg:

1. Należy wywołać PU Modyfikacja informacji dotyczących samochodu.
2. Należy zaktualizować status pojazdu na „dostępny”
3. Należy zmienić status wypożyczenia na "nieaktywne".

## 2.3. Wymagania niefunkcjonalne

### 2.3.1. Wykorzystywane technologie i narzędzia

Baza danych zostanie stworzona w oparciu o system zarządzania bazą danych MySQL. Interfejs użytkownika umożliwiający wykonywanie operacji na danych zawartych w bazie danych będzie zrealizowany przy użyciu biblioteki ReactJS (JavaScript). Komunikację pomiędzy interfejsem użytkownika a bazą danych będzie zapewniał serwer bazy danych wykorzystujący technologię Java.

### 2.3.2. Wymagania dotyczące bezpieczeństwa systemu

Każdy dostęp do bazy danych poprzedzony będzie autoryzacją użytkownika. Użytkownik w celu uzyskania dostępu do systemu będzie musiał potwierdzić swoją tożsamość poprzez zalogowanie się w serwisie. Hasła będą przechowywane w formie zaszyfrowanej przy użyciu funkcji skrótu MD5. Po poprawnej identyfikacji użytkownikowi zostanie przypisany losowo wygenerowany token sesji, który umożliwia dostęp do danych.

## 2.4. Przyjęte założenia projektowe

### 2.4.1. Założenia architektoniczne przyjęte podczas realizacji systemu

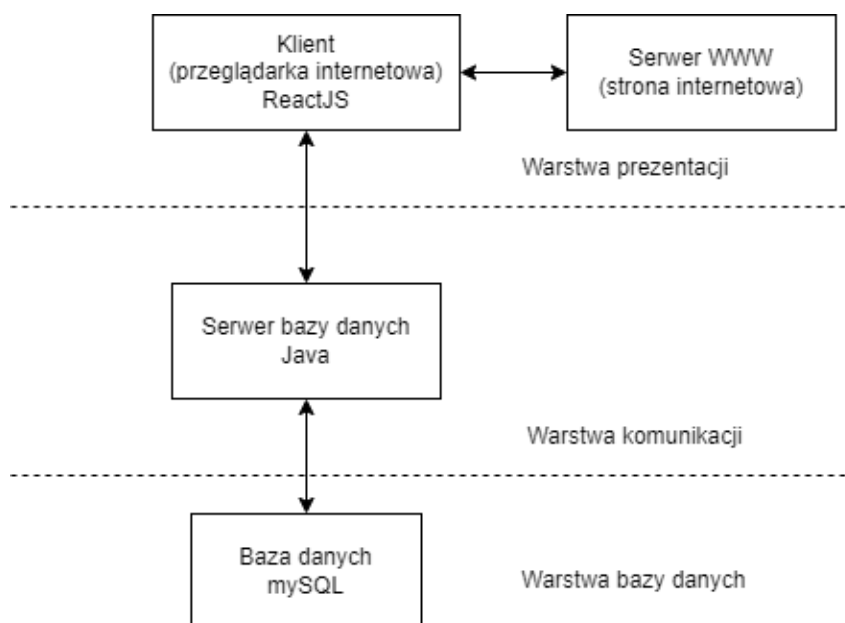
Projekt będzie realizowany w 3-warstwowej architekturze komunikacji klient/serwer(rys. 2.2). Pośrednikiem w komunikacji pomiędzy aplikacją a bazą danych będzie serwer bazy danych zrealizowany w technologii Java. Aplikacja będzie komunikowała się z serwerem bazy danych przy użyciu zapytań protokołu HTTP zgodnych ze stylem architektury oprogramowania REST[9]. Serwer bazy danych będzie weryfikował spójność każdego zapytania, pobierał, zapisywał, usuwał lub modyfikował odpowiednie dane w bazie danych oraz odpowiadał aplikacji danymi w formacie JSON[10]. Serwer bazy danych będzie umożliwiał dostęp do danych tylko autoryzowanym użytkownikom. W zastosowanym modelu przetwarzanie danych jest wykonywane głównie po stronie bazy danych, w niektórych przypadkach po stronie klienta WWW.

### 2.4.2. Wykorzystywane technologie, narzędzia projektowania oraz implementacji systemu

Baza danych będzie obsługiwana za pośrednictwem systemu zarządzania bazą danych MySQL[11], z którym komunikować się będzie serwer bazy danych (Java). Interfejs użytkownika będzie zrealizowany w postaci aplikacji przeglądarkowej opartej na bibliotece ReactJS[5] języka JavaScript. Aplikacja przeglądarkowa będzie udostępniona na serwerze WWW, natomiast będzie ona uruchamiana w przeglądarce użytkownika. Aplikacja będzie komunikowała się z serwerem za pomocą REST API[9].

### 2.4.3. Schemat komunikacji, struktura systemu

Na rysunku 2.2 przedstawiono strukturę komunikacji w systemie.



Rys. 2.2: Schemat komunikacji w systemie.

# Rozdział 3

## Projekt systemu

### 3.1. Projekt bazy danych

#### 3.1.1. Analiza rzeczywistości

Firma posiada kilka oddziałów *wypożyczalni*. W każdym z oddziałów są dostępne różne *samochody* do wynajęcia. Firma oferuje wiele *modeli* samochodów. *Klienci* mogą wypożyczać samochody po uprzednim zarejestrowaniu w systemie. Każdy z *klientów* może mieć wiele aktywnych *wypożyczeń* przypisanych do jego konta.

#### 3.1.2. Model logiczny

W wyniku analizy rzeczywistości wyróżniono poniższe encje:

##### **Użytkownicy**

Tabela przechowuje informacje dotyczące danych logowania użytkowników. Zostanie ona użyta do zrealizowania uwierzytelniania użytkownika chcącego wykorzystać funkcje systemu do których dostęp jest ograniczony. Warto zaznaczyć że użytkownikami będą tylko pracownicy wypożyczalni. Interfejs dostępny dla klientów będzie dostępny bez konieczności logowania i będzie umożliwiał przeglądanie dostępnych samochodów.

##### **Wypożyczalnie**

Tabela przechowuje informacje o oddziałach wypożyczalni. Zawiera informacje o lokalizacji poszczególnych oddziałów wypożyczalni.

##### **Modele**

Tabela przechowuje informacje o konkretnych modelach samochodów. Przechowywane są: marka, model, spalanie, liczba miejsc, liczba drzwi, pojemność zbiornika, rodzaj paliwa, pojemność silnika, moc silnika oraz rodzaj karoserii. Tabela zawiera również informację o cenie wynajmu tego modelu samochodu za dzień.

##### **Samochody**

Tabela przechowuje informacje o samochodach które przynależą do wypożyczalni. Zawiera informacje o modelu samochodu, numerze rejestracyjnym, dacie produkcji, ostatniej dacie prze-

glądu oraz przebiegu. Z informacji zawartych w tabeli można również dowiedzieć się w którym oddziale wypożyczalni znajduje się samochód.

## Klienci

Tabela przechowuje informacje o klientach wypożyczalni. Zawiera informacje o danych osobowych i kontaktowych klienta.

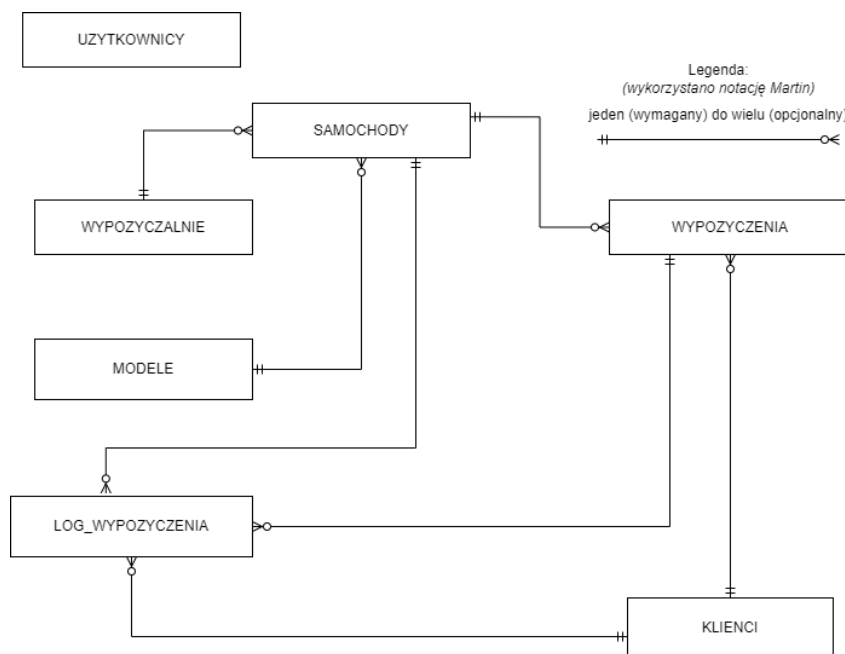
## Wypożyczenia

Tabela przechowuje informacje o wszystkich wypożyczeniach, zarówno aktualnych, zakończonych jak i zaplanowanych. Zawiera informacje o statusie wypożyczenia, kliencie wypożyczającym i samochodzie który jest wypożyczany. W skład informacji o wypożyczeniu wchodzi również daty rozpoczęcia i zakończenia wypożyczenia.

## Log wypożyczenia

Tabela przechowuje wszystkie wartości które zostały wpisane do tabeli wypożyczenia. Inaczej mówiąc, jeżeli wypożyczenie zostanie dodane lub zaktualizowane, w tabeli log wypożyczenia zostanie zapisany stan wypożyczenia po dodaniu lub zaktualizowaniu. Dzięki temu tabela przechowuje każdy poprzedni stan każdego z wypożyczeń, co umożliwia przeglądanie historii edycji każdego wypożyczenia. Log wypożyczenia może zostać wykorzystany w przypadku gdy tabela wypożyczeń zostanie edytowana w niewłaściwy sposób. Tabela wypożyczeń zawiera bardzo istotne informacje, czyli między innymi to, kto jest w posiadaniu danego samochodu, przez co log wypożyczeń może okazać się przydatny w wielu sytuacjach. Funkcjonalność logu wypożyczenia została zrealizowana przy użyciu wyzwalaczy (ang. trigger).

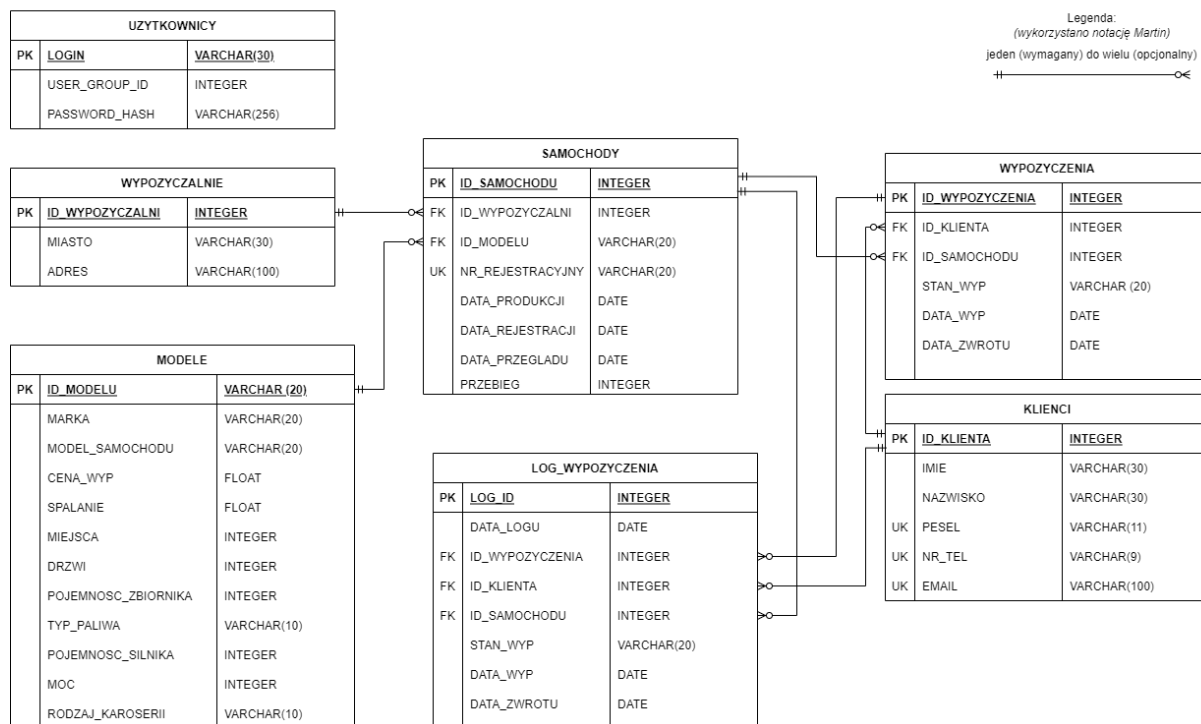
Na rysunku 3.1 przedstawiono uproszczony model logiczny projektowanej bazy danych.



Rys. 3.1: Schemat logiczny projektowanej bazy danych.

### 3.1.3. Model fizyczny i ograniczenia integralności danych

Na rysunku 3.2 przedstawiono fizyczny model relacji encji projektowanej bazy danych.



Rys. 3.2: Schemat fizyczny projektowanej bazy danych.

Poza ograniczeniami wynikającymi z typów danych widocznych na schemacie fizycznym przedstawionym na rysunku 3.2 na poniżej wymienione atrybuty nałożono dodatkowe ograniczenia. W każdej z tabel również wartości kluczy obcych muszą być istniejącymi wartościami kluczy głównych w tabelach powiązanych relacją.

#### Użytkownicy

- *USER\_GROUP\_ID* może przyjmować wartości 0 lub 1.

#### Wypożyczalnie

- Brak dodatkowych ograniczeń.

#### Modele

- *TYP\_PALIWA* może przyjmować tylko następujące wartości: "DIESEL", "BENZYNA", "GAZ", "BENZYNA+GAZ", "ELEKTRYCZNY"
- *RODZAJ\_KAROSERII* może przyjmować tylko następujące wartości: "SEDAN", "KOMBI", "HATCHBACK", "SUV", "TERENOWY", "SPORT"

#### Samochody

- *DATA\_REJESTRACJI* nie może być wcześniejsza niż *DATA\_PRODUKCJI*
- *DATA\_PRZEGLADU* nie może być wcześniejsza niż *DATA\_REJESTRACJI*

### Klienci

- *PESEL* - 11-cyfrowa liczba, gdzie ostatnia cyfra jest cyfrą kontrolną obliczaną według [6]

### Wypożyczenia

- *STAN\_WYP* może przyjmować tylko następujące wartości:  
"AKTUALNE", "ZAKONCZONE",
- *DATA\_ZWROTU* nie może być wcześniejsza niż *DATA\_WYP*

### Log wypożyczenia

- Ponieważ jest to tabela logu, która jest wypełniana przez wyzwalacz (ang. *trigger*), tabela jest dostępna w trybie tylko do odczytu.

Walidacja poprawności danych odbywać się będzie na poziomie aplikacji. System zarządzania bazą danych oraz serwer bazy danych zapewniają jedynie poprawność typów danych.

### 3.1.4. Inne elementy schematu - widoki

Widok *DOSTĘPNE\_SAMOCHODY* (rys. 3.3) przedstawia informacje dotyczące samochodów które nie są aktualnie wypożyczone. Widok *NIEDOSTĘPNE\_SAMOCHODY* (rys. 3.3) przedstawia natomiast informacje o wypożyczonych samochodach. Widoki mogą być wykorzystane przykładowo przez klienta do zapoznania się z informacjami o samochodach i ich dostępności.

DOSTĘPNE_SAMOCHODY		NIEDOSTĘPNE_SAMOCHODY	
ID SAM.	int unsigned	ID SAM.	int unsigned
STATUS	varchar(22)	STATUS	varchar(22)
MIASTO	varchar(30)	PLANOWANA DATA ZWROTU	date
MARKA	varchar(20)	MIASTO	varchar(30)
MODEL	varchar(20)	MARKA	varchar(20)
DATA PRODUKCJI	date	MODEL	varchar(20)
CENA	float	DATA PRODUKCJI	date
SPALANIE	float	CENA	float
L. MIEJSC	int unsigned	SPALANIE	float
L. DRZWI	int unsigned	L. MIEJSC	int unsigned
POJEMNOSC ZBIORNIKA	int unsigned	L. DRZWI	int unsigned
TYP PALIWA	varchar(10)	POJEMNOSC ZBIORNIKA	int unsigned
POJEMNOSC SILNIKA	int unsigned	TYP PALIWA	varchar(10)
MOC	int unsigned	POJEMNOSC SILNIKA	int unsigned
RODZAJ KAROSERII	varchar(10)	MOC	int unsigned
		RODZAJ KAROSERII	varchar(10)

Rys. 3.3: Widok dostępne samochody

Widok *WYPOŻYCZONE\_SAMOCHODY* (rys. 3.4) przedstawia informacje o aktualnych wypożyczeniach - tych które nie zostały jeszcze zakończone. Przedstawia informacje o wypożyczanym samochodzie, statusie wypożyczenia i o kliencie wypożyczającym. Widok może być



przydatny dla pracownika do kontrolowania czy samochody których termin oddania upłynął zostały oddane.

WYPOZYCZONE_SAMOCODY	
ID SAM.	int unsigned
STATUS	varchar(11)
MIASTO	varchar(30)
MARKA	varchar(20)
MODEL	varchar(20)
NR REJESTRACYJNY	varchar(20)
DATA PRODUKCJI	date
CENA	float
ID KLIENTA	int unsigned
IMIE	varchar(30)
NAZWISKO	varchar(30)
DATA WYPOZYCZENIA	date
DATA ZWROTU	date

Rys. 3.4: Widok przedstawiający wypożyczone samochody

## 3.2. Projekt aplikacji użytkownika

Aplikacja użytkownika została wykonana w języku *Javascript* przy pomocy *framework'u React* pozwalającego na łatwe oraz szybkie tworzenie aplikacji SPA (*Single Page Application*). Biblioteka ta ze względu na swoją popularność posiada wiele materiałów pomocniczych, takich jak gotowe komponenty HTML.

Aplikacja została podzielona na ścieżki, każda ścieżka odpowiada jednej z kluczowych funkcjonalności wypożyczalni. Poniżej przedstawiona została każda ze ścieżek oraz jej krótki opis.

- / - strona główna, gdy użytkownik jest zalogowany wyświetli widok klienta, w przeciwnym wypadku przekieruje na */login*.
- */login* - strona logowania klienta.
- */admin* - strona główna administracji, w przypadku braku autoryzacji administratora przekieruje na */admin/login*.
- */admin/login* - strona logowania administratora.

### 3.2.1. Architektura aplikacji i diagramy projektowe

Aplikacja zaprojektowana została jako wspomniane wcześniej *Single Page Application*, nazwa ta nie posiada swojego tłumaczenia w języku polskim i oznacza, że cały interfejs użytkownika zbudowany jest na jednym pliku *.html* pobranym przez przeglądarkę z serwera WWW. Plik *.html* **nie** jest wcześniej renderowany przez serwer, czy wypełniany danymi z bazy danych. Oznacza to, że każda instancja aplikacji działająca na przeglądarce użytkownika musi "w biegu" pozyskać dane z bazy danych. Takie podejście powoduje, że wymagane jest wprowadzenie dodatkowej warstwy — komunikacja między aplikacją a bazą musi odbywać się przez serwer pośredniczący (tzw. *backend*), który zajmuje się całą logiką biznesową — będzie odpowiedzialny za weryfikację spójności danych wysyłanych z/do bazy, odpowiednie przekształcanie tych danych oraz weryfikację uprawnień użytkowników.

Komunikacja między aplikacją, a *backend'em* jest bezstanowa i odbywa się za pomocą zapytań HTTP. Konieczne było stworzenie jednorodnego interfejsu (*Rest API*[9]), który pozwalałby

na manipulację danymi. Dzięki użyciu protokołu HTTP zapytania mogły zostać podzielone na *GET* (pobieranie danych), *POST* (dodawanie danych), *PUT* (edycję danych), *DELETE* (usuwanie danych). Każde zapytanie wysłane z aplikacji posiada jedną z wymienionych metod w raz z opcjonalnym ładunkiem (*payload*) w formacie *JSON*[10]. Po wysłaniu zapytania przez aplikację, serwer odpowiednio ją przetwarza, oraz wysyła odpowiedź, która w zależności od kodu statusu HTTP[1] oznacza udane lub nieudane zapytanie wraz z potencjalnym ładunkiem (wiadomość błędu w przypadku nieudanego zapytania, czy dane pozyskiwane przez klienta w przypadku pomyślnego zapytania).

### 3.2.2. Interfejs graficzny i struktura menu

Interfejs graficzny został wykonany przy pomocy biblioteki *Material UI* [2] zapewniającej gotowe do użycia komponenty *React* (stylizowane przyciski, kontrolki, pola tekstowe czy listy). Dzięki temu cały interfejs użytkownika wygląda czysto oraz jednolicie. Poniżej przedstawione zostały zrzuty ekranu najważniejszych części interfejsu użytkownika w aplikacji. Rysunki 3.5 oraz 3.6 przedstawiają stronę logowania oraz rejestracji.



Rys. 3.5: Strona logowania



Rys. 3.6: Strona rejestracji

Strona główna wyświetlana jest tylko gdy użytkownik jest zalogowany. Użytkownik posiada jedynie możliwość przeglądania listy dostępnych pojazdów będącej widokiem. Rysunek 3.7 przedstawia stronę główną użytkownika.

Wypożyczalnia samochodów												WYLOGUJ SIĘ
Dostępne samochody												
ID	Dostępność	Miasto	Marka	Model	Data produkcji	Cena (za dzień)	Spalanie[L/100km]	L. Miejsc	L. Drzwi	Pojemność zbiornika	Rodzaj paliwa	
1	DOSTĘPNY	WROCLAW	NISSAN	QUASHQAI	2013-08-28	200	5	5	5	75	DIESEL	
2	DOSTĘPNY	WARSZAWA	TOYOTA	SUPRA	2012-02-27	250	15	2	3	70	BENZYNĄ	
3	DOSTĘPNY	KRAKOW	NISSAN	QUASHQAI	2013-03-23	200	5	5	5	75	DIESEL	
4	DOSTĘPNY	POZNAN	BMW	E90	2011-06-27	150	12	5	5	65	BENZYNĄ	
5	DOSTĘPNY	GDANSK	TOYOTA	SUPRA	2020-06-20	250	15	2	3	70	BENZYNĄ	
6	DOSTĘPNY	WROCLAW	TOYOTA	SUPRA	2014-07-15	250	15	2	3	70	BENZYNĄ	
7	DOSTĘPNY	WARSZAWA	MITSUBISHI	ASX	2018-01-17	175	7	5	5	55	BENZYNĄ	
8	DOSTĘPNY	KRAKOW	VOLKSWAGEN	GOLF V	2017-03-12	100	7	5	5	60	DIESEL	

Rys. 3.7: Strona główna (użytkownik)

Panel administratora pod względem implementacyjnym jest tak naprawdę rozbudowaną wersją strony użytkownika.

Wypożyczalnia samochodów

WYLOGUJ SIĘ

Lista samochodów

DODAJ SAMOCHÓD

Id	Lokalizacja	Model	Nr rejestracyjny	Data produkcji	Data rejestracji	Data przeglądu	Przebieg		
1	WROCLAW UL. JAGIELLY 11	QUASHQAI	DW 35542	2013-08-28	2013-09-22	2021-06-14	244634	EDYTUJ	USUŃ
2	WARSZAWA UL. KOSCIUSZKI 7	SUPRA	DW 45932	2012-02-27	2012-03-20	2021-06-28	457293	EDYTUJ	USUŃ
3	KRAKOW UL. PILUSUDSKIEGO 5	QUASHQAI	DW 64823	2013-03-23	2013-04-28	2021-02-28	231435	EDYTUJ	USUŃ
4	POZNAN UL. 3 MAJA 2	E90	DW 93880	2011-06-27	2011-07-11	2021-03-13	245959	EDYTUJ	USUŃ
5	GDANSK UL. 11 LISTOPADA 1	SUPRA	DW 85098	2020-06-20	2020-07-10	2021-07-16	570529	EDYTUJ	USUŃ
6	WROCLAW UL. JAGIELLY 11	SUPRA	DW 18401	2014-07-15	2014-08-18	2021-07-15	223357	EDYTUJ	USUŃ

Rys. 3.8: Panel administratora

Panel administratora został dodatkowo wyposażony o możliwość dodawania, usuwania oraz edycji dowolnego pojazdu. Okno dodawania oraz edycji przedstawione zostało na rysunku 3.9.

Rys. 3.9: Okno dodawania/edycji pojazdu

### 3.2.3. Projekt wybranych funkcji systemu

Każde żądanie wysyłane jest do serwera bazy danych (*backend'u*) wraz z tokenem autoryzacji uzyskanym w procesie logowania.

W procesie dodawania nowego pojazdu, po wypełnieniu przez użytkownika formularza z jego danymi, aplikacja wysyła do serwera bazy danych żądanie HTTP *POST* po adres `/data/samochody` zawierające w ciele wiadomości obiekt *JSON* przedstawiający samochód. Żądanie dodania nowego samochodu wymaga dodatkowo podania tokenu uwierzytelniania jako parametr zapytania (*Query String* — przykład ścieżki żądania przedstawiony na rysunku 3.10). Po wysłaniu żądania, serwer weryfikuje spójność otrzymanych danych oraz sprawdza, czy podany token istnieje w bazie zalogowanych użytkowników. W ostatnim kroku serwer odsyła odpowiednią odpowiedź na żądanie aplikacji.

Proces edycji jest bardzo podobny do procesu dodawania, z jedną najważniejszą różnicą — metoda zapytania to *PUT* oznaczająca aktualizację danych.

Listing 3.1: Przykładowy obiekt opisujący samochód

```
{
  "idSamochodu": null,
  "idWypożyczalni": {
    "id": 1,
    "miasto": "WROCLAW",
    "adres": "UL. JAGIELLY 11"
  },
  "idModelu": {
    "idModelu": "MIT_ASX",
    "marka": "MITSUBISHI",
    "modelSamochodu": "ASX",
    "cenaWyp": 175,
    "spalanie": 7,
    "miejsca": 5,
    "drzwi": 5,
    "pojemnoscZbiornika": 55,
    "typPaliwa": "BENZyna",
    "pojemnoscSilnika": 2200,
    "moc": 180,
  },
}
```

```
"rodzajKaroserii": "SUV"  
},  
"nrRejestracyjny": "019283091283",  
"dataProdukcji": "2022-01-28",  
"dataRejestracji": "2022-01-28",  
"dataPrzeglądu": "2022-01-28",  
"przebieg": "213"  
}
```

`http://localhost:8080/data/samochody?authToken=0flPwUg10AMYLYXDTKIJ`

Rys. 3.10: Przykładowa ścieżka żądania dodania nowego samochodu

### 3.2.4. Metoda podłączenia do bazy danych - integracja z bazą danych

Aplikacja będzie komunikowała się z bazą danych za pośrednictwem serwera bazy danych stworzonego przy użyciu technologii *Java Spring Boot*. Komunikacja będzie odbywała się z wykorzystaniem protokołu *HTTP*. Dane przesyłane będą w formacie *JSON*.

# Rozdział 4

## Implementacja systemu

### 4.1. Realizacja bazy danych

#### 4.1.1. Tworzenie tabel

Poniższe listingi (4.1 - 4.7) przedstawiają definicje tabel w implementowanej bazie danych.

Listing 4.1: Kod definiujący tabelę *MODELE*

```
create table MODELE(  
    ID_MODELU varchar(20) primary key unique,  
    MARKA varchar(20),  
    MODEL_SAMOCODU varchar(20),  
    CENA_WYP float,  
    SPALANIE float,  
    MIEJSCA int unsigned,  
    DRZWI int unsigned,  
    POJEMNOSC_ZBIORNIKA int unsigned,  
    TYP_PALIWA varchar(10),  
    POJEMNOSC_SILNIKA int unsigned,  
    MOC int unsigned,  
    RODZAJ_KAROSERII varchar(10)  
);
```

Listing 4.2: Kod definiujący tabelę *WYPOZYCZALNIE*

```
create table WYPOZYCZALNIE(  
    ID_WYPOZYCZALNI int unsigned primary key unique auto_increment,  
    MIASTO varchar(30),  
    ADRES varchar(100)  
);
```

Listing 4.3: Kod definiujący tabelę *SAMOCODY*

```
create table SAMOCODY(  
    ID_SAMOCODU int unsigned primary key unique auto_increment,  
    ID_WYPOZYCZALNI int unsigned,  
    foreign key (ID_WYPOZYCZALNI) references WYPOZYCZALNIE(ID_WYPOZYCZALNI)  
    ↪ ,  
    ID_MODELU varchar(20),  
    foreign key (ID_MODELU) references MODELE(ID_MODELU),  
    NR_REJESTRACYJNY varchar(20) unique,  
    DATA_PRODUKCJI date,  
    DATA_REJESTRACJI date,  
    DATA_PRZEGLADU date,  
    PRZEBIEG int unsigned  
);
```

Listing 4.4: Kod definiujący tabelę *KLIENCI*

```
create table KLIENCI(
  ID_KLIENTA int unsigned primary key unique auto_increment,
  IMIE varchar(30),
  NAZWISKO varchar(30),
  PESEL varchar(11) unique,
  NR_TEL varchar(9) unique,
  EMAIL varchar(100) unique
);
```

Listing 4.5: Kod definiujący tabelę *WYPOZYCZENIA*

```
create table WYPOZYCZENIA(
  ID_WYPOZYCZENIA int unsigned primary key unique auto_increment,
  ID_KLIENTA int unsigned,
  foreign key (ID_KLIENTA) references KLIENCI(ID_KLIENTA),
  ID_SAMOCHODU int unsigned,
  foreign key (ID_SAMOCHODU) references SAMOCHODY(ID_SAMOCHODU),
  STAN_WYP varchar(20),
  DATA_WYP date,
  DATA_ZWROTU date
);
```

Listing 4.6: Kod definiujący tabelę *LOG\_WYPOZYCZENIA*

```
create table LOG_WYPOZYCZENIA(
  LOG_ID int unsigned primary key unique auto_increment,
  DATA_LOGU date,
  ID_WYPOZYCZENIA int unsigned,
  foreign key (ID_WYPOZYCZENIA) references WYPOZYCZENIA(ID_WYPOZYCZENIA),
  ID_KLIENTA int unsigned,
  foreign key (ID_KLIENTA) references KLIENCI(ID_KLIENTA),
  ID_SAMOCHODU int unsigned,
  foreign key (ID_SAMOCHODU) references SAMOCHODY(ID_SAMOCHODU),
  STAN_WYP varchar(20),
  DATA_WYP date,
  DATA_ZWROTU date
);
```

Listing 4.7: Kod definiujący tabelę *UZYTKOWNICY*

```
create table UZYTKOWNICY(
  LOGIN varchar(30) primary key unique,
  USER_GROUP_ID int unsigned,
  PASSWORD_HASH varchar(256)
);
```

### 4.1.2. Tworzenie widoków

Poniższe listingi (4.8 - 4.10) przedstawiają tworzenie widoków w implementowanej bazie danych.

Listing 4.8: Kod definiujący widok *DOSTEPNE\_SAMOCHODY*

```
create or replace view DOSTEPNE_SAMOCHODY as
select
  distinct S.ID_SAMOCHODU AS "ID_SAM",
  CASE WA.STAN_WYP WHEN 'ZAKONCZONE' THEN 'DOSTEPNY'
                   WHEN 'ZAPLANOWANE' THEN 'DOSTEPNY/ZAREZERWOWANY'
                   WHEN 'AKTUALNE' THEN 'NIEDOSTEPNY'
  END
```

```

AS "STATUS",
WE.MIASTO AS "MIASTO",
M.MARKA AS "MARKA",
M.MODEL_SAMOCZODU AS "MODEL",
S.DATA_PRODUKCJI AS "DATA_PRODUKCJI",
M.CENA_WYP AS "CENA",
M.SPALANIE AS "SPALANIE",
M.MIEJSCA AS "L_MIEJSC",
M.DRZWI AS "L_DRZWI",
M.POJEMNOSC_ZBIORNIKA AS "POJEMNOSC_ZBIORNIKA",
M.TYP_PALIWA AS "TYP_PALIWA",
M.POJEMNOSC_SILNIKA AS "POJEMNOSC_SILNIKA",
M.MOC AS "MOC",
M.RODZAJ_KAROSERII AS "RODZAJ_KAROSERII"
FROM SAMOCZODY S
JOIN WYPOZYCZENIA WA
ON WA.ID_SAMOCZODU = S.ID_SAMOCZODU
JOIN MODELE M
ON S.ID_MODELU = M.ID_MODELU
JOIN WYPOZYCZALNIE WE
ON WE.ID_WYPOZYCZALNI = S.ID_WYPOZYCZALNI
WHERE S.ID_SAMOCZODU NOT IN (
SELECT
    DISTINCT S.ID_SAMOCZODU AS "ID_SAM"
FROM SAMOCZODY S
JOIN WYPOZYCZENIA WA
ON WA.ID_SAMOCZODU = S.ID_SAMOCZODU
WHERE WA.STAN_WYP = 'AKTUALNE'
);

```

Listing 4.9: Kod definiujący widok *NIEDOSTEPNE\_SAMOCZODY*

```

create or replace view NIEDOSTEPNE_SAMOCZODY as
select
    distinct S.ID_SAMOCZODU AS "ID_SAM",
    CASE WA.STAN_WYP WHEN 'ZAKONCZONE' THEN 'DOSTEPNY'
                     WHEN 'ZAPLANOWANE' THEN 'DOSTEPNY/ZAREZERWOWANY'
                     WHEN 'AKTUALNE' THEN 'NIEDOSTEPNY'
    END
AS "STATUS",
WA.DATA_ZWROTU AS "PLANOWANA_DATA_ZWROTU",
WE.MIASTO AS "MIASTO",
M.MARKA AS "MARKA",
M.MODEL_SAMOCZODU AS "MODEL",
S.DATA_PRODUKCJI AS "DATA_PRODUKCJI",
M.CENA_WYP AS "CENA",
M.SPALANIE AS "SPALANIE",
M.MIEJSCA AS "L_MIEJSC",
M.DRZWI AS "L_DRZWI",
M.POJEMNOSC_ZBIORNIKA AS "POJEMNOSC_ZBIORNIKA",
M.TYP_PALIWA AS "TYP_PALIWA",
M.POJEMNOSC_SILNIKA AS "POJEMNOSC_SILNIKA",
M.MOC AS "MOC",
M.RODZAJ_KAROSERII AS "RODZAJ_KAROSERII"
FROM SAMOCZODY S
JOIN WYPOZYCZENIA WA
ON WA.ID_SAMOCZODU = S.ID_SAMOCZODU
JOIN MODELE M
ON S.ID_MODELU = M.ID_MODELU
JOIN WYPOZYCZALNIE WE
ON WE.ID_WYPOZYCZALNI = S.ID_WYPOZYCZALNI

```



```
WHERE WA.STAN_WYP = 'AKTUALNE';
```

Listing 4.10: Kod definiujący widok *WYPOZYCZONE\_SAMOCODY*

```
CREATE OR REPLACE VIEW WYPOZYCZONE_SAMOCODY AS
SELECT
  DISTINCT S.ID_SAMOCODU AS "ID_SAM",
  CASE WA.STAN_WYP WHEN 'AKTUALNE' THEN 'WYPOZYCZONY'
  END
  AS "STATUS",
  WE.MIASTO AS "MIASTO",
  M.MARKA AS "MARKA",
  M.MODEL_SAMOCODU AS "MODEL",
  S.NR_REJESTRACYJNY AS "NR_REJESTRACYJNY",
  S.DATA_PRODUKCJI AS "DATA_PRODUKCJI",
  M.CENA_WYP AS "CENA",
  K.ID_KLIENTA AS "ID_KLIENTA",
  K.IMIE AS "IMIE",
  K.NAZWISKO AS "NAZWISKO",
  WA.DATA_WYP AS "DATA_WYPOZYCZENIA",
  WA.DATA_ZWROTU AS "DATA_ZWROTU"
FROM WYPOZYCZENIA WA
JOIN KLIENCI K
ON K.ID_KLIENTA = WA.ID_KLIENTA
JOIN SAMOCODY S
ON WA.ID_SAMOCODU = S.ID_SAMOCODU
JOIN MODELE M
ON S.ID_MODELU = M.ID_MODELU
JOIN WYPOZYCZALNIE WE
ON WE.ID_WYPOZYCZALNI = S.ID_WYPOZYCZALNI
WHERE WA.STAN_WYP = 'AKTUALNE'
ORDER BY WA.DATA_ZWROTU;
```

### 4.1.3. Implementacja wyzwalaczy

Na listingu 4.11 przedstawiono kod definiujący wyzwalacze, które umożliwiają logowanie zmian występujących w tabeli *WYPOZYCZENIA*.

Listing 4.11: Kod definiujący wyzwalacze dla tabeli *LOG\_WYPOZYCZEN*

```
create trigger LOG_WYPOZYCZENIA_INS_TRIGGER
after insert on WYPOZYCZENIA
for each row
  insert into LOG_WYPOZYCZENIA
  values (NULL, sysdate(), new.ID_WYPOZYCZENIA, new.ID_KLIENTA, new.
    ↪ ID_SAMOCODU,
    new.STAN_WYP, new.DATA_WYP, new.DATA_ZWROTU);

create trigger LOG_WYPOZYCZENIA_UP_TRIGGER
after update on WYPOZYCZENIA
for each row
  insert into LOG_WYPOZYCZENIA
  values (NULL, sysdate(), new.ID_WYPOZYCZENIA, new.ID_KLIENTA, new.
    ↪ ID_SAMOCODU,
    new.STAN_WYP, new.DATA_WYP, new.DATA_ZWROTU);
```

## 4.2. Realizacja serwera bazy danych

Serwer bazy danych został utworzony z wykorzystaniem technologii Java Spring Boot.

### 4.2.1. Model encji

Realizację modelu encji w serwerze bazy danych przedstawiono na przykładzie encji *KLIENCI*. Przykład jest widoczny w poniższym listingu (4.12).

Listing 4.12: Kod definiujący model encji *KLIENCI*.

```
package db2.carrental.entities;

import javax.persistence.*;
import java.io.Serializable;
import java.util.List;

@Entity
@Table(name = "klienci")
public class Klienci implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @Column(name = "ID_KLIENTA", nullable = false)
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer idKlienta;

    @Column(name = "IMIE")
    private String IMIE;

    @Column(name = "NAZWISKO")
    private String NAZWISKO;

    @Column(name = "PESEL")
    private String PESEL;

    @Column(name = "NR_TEL")
    private String nrTel;

    @Column(name = "EMAIL")
    private String EMAIL;

    @OneToMany(mappedBy = "idKlienta", cascade = CascadeType.ALL)
    private List<Wypozyczenia> wypozyczenia;

    @OneToMany(mappedBy = "idKlienta", cascade = CascadeType.ALL)
    private List<LogWypozyczenia> logWypozyczenia;

    public void setIdKlienta(Integer idKlienta) {
        this.idKlienta = idKlienta;
    }
    public Integer getIdKlienta() {
        return idKlienta;
    }
    public void setIMIE(String IMIE) {
        this.IMIE = IMIE;
    }
    public String getIMIE() {
        return IMIE;
    }
    public void setNAZWISKO(String NAZWISKO) {
        this.NAZWISKO = NAZWISKO;
    }
    public String getNAZWISKO() {
        return NAZWISKO;
    }
}
```

```

    }
    public void setPESEL(String PESEL) {
        this.PESEL = PESEL;
    }
    public String getPESEL() {
        return PESEL;
    }
    public void setNrTel(String nrTel) {
        this.nrTel = nrTel;
    }
    public String getNrTel() {
        return nrTel;
    }
    public void setEmail(String EMAIL) {
        this.EMAIL = EMAIL;
    }
    public String getEmail() {
        return EMAIL;
    }
    @Override
    public String toString() {
        return "Klienci{" +
            "idKlienta=" + idKlienta + '\n' +
            "IMIE=" + IMIE + '\n' +
            "NAZWISKO=" + NAZWISKO + '\n' +
            "PESEL=" + PESEL + '\n' +
            "nrTel=" + nrTel + '\n' +
            "EMAIL=" + EMAIL + '\n' +
            '}';
    }
}

```

### 4.2.2. Serwis autentykacji

W listingu 4.13 przedstawiono implementację serwisu autentykacji, który umożliwia autoryzację użytkownika poprzez stwierdzenie zgodności podanych danych użytkownika (tj. login, hasło) z danymi istniejącymi w bazie. Jeżeli podane dane są prawidłowe, zostaje tworzona nowa sesja której identyfikatorem jest unikalny losowo generowany token. Aplikacja użytkownika w odpowiedzi na żądanie zalogowania otrzymuje wcześniej wspomniany token, który musi być przesyłany przy każdej próbie komunikacji z bazą danych. Serwis dostępu do danych który jest opisany w kolejnym podrozdziale odpowiada na żądania tylko wtedy, gdy istnieje otwarta sesja z identycznym tokenem jak przesłany podczas próby uzyskania dostępu do danych. Gdy przesłany token nie jest zgodny z żadnym z listy tokenów sesji, serwer bazy danych odpowiada błędem. Serwis autentykacji umożliwia również rejestrowanie nowych użytkowników, oraz zamykanie sesji (wylogowywanie).

Listing 4.13: Implementacja serwisu autentykacji.

```

package db2.carrental.services;

import db2.carrental.entities.AuthToken;
import db2.carrental.entities.Uzytkownicy;
import db2.carrental.repositories.UzytkownicyRepository;
import org.springframework.stereotype.Service;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;

```

```

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

@Service
public class AuthService {

    private final List<AuthToken> authTokens;
    private final SecureRandom random;
    private final UzytkownicyRepository uzytkownicyRepository;

    public AuthService(UzytkownicyRepository uzytkownicyRepository) {
        this.uzytkownicyRepository = uzytkownicyRepository;
        this.authTokens = new ArrayList<>();
        this.random = new SecureRandom();
    }

    public boolean isUserAuthenticated(String token, int permissionLevel) {
        return authTokens.stream().anyMatch(stream -> stream.getAuthToken()
            ↪ .equals(token)
            && stream.getPermissionLevel() >= permissionLevel);
    }

    public Optional<AuthToken> getTokenForEmail(String email){
        return authTokens.stream().filter(authToken -> authToken.getEmail()
            ↪ .equals(email)).findAny();
    }

    public Optional<AuthToken> getTokenObjForTokenStr(String token) {
        return authTokens.stream().filter(authToken -> authToken.
            ↪ getAuthToken().equals(token)).findAny();
    }

    public AuthToken createToken(String email, int permissionLevel){
        AuthToken newToken = new AuthToken(email, permissionLevel);
        authTokens.add(newToken);
        return newToken;
    }

    public AuthToken login(String email, String password, int
        ↪ permissionLevel) throws NoSuchAlgorithmException {
        Optional<Uzytkownicy> user = uzytkownicyRepository.findById(email);
        if (user.isEmpty()) throw new IllegalArgumentException("Could not
            ↪ login, user does not exist.");
        if (user.get().getUserGroupId() < permissionLevel)
            throw new IllegalArgumentException("Could not login, user
                ↪ permissions insufficient");
        String hashPswd = hashPassword(password);
        if (!user.get().getPasswordHash().equals(hashPswd))
            throw new IllegalArgumentException("Wrong password.");
        return createToken(email, permissionLevel);
    }

    private String hashPassword(String password) throws
        ↪ NoSuchAlgorithmException {
        MessageDigest md = MessageDigest.getInstance("MD5");

        md.update(password.getBytes());
    }

```

```

        byte[] bytes = md.digest();

        StringBuilder sb = new StringBuilder();
        for (byte aByte : bytes) {
            sb.append(Integer.toString((aByte & 0xff) + 0x100, 16).
                ↪ substring(1));
        }
        return sb.toString();
    }

    public void registerUser(Uzytkownicy user) throws
        ↪ NoSuchAlgorithmException {
        if (uzytkownicyRepository.findById(user.getId()).isPresent()) throw
            ↪ new IllegalArgumentException("Account already registered.");
        user.setPasswordHash(hashPassword(user.getPasswordHash()));
        uzytkownicyRepository.save(user);
    }

    public void revoke(String token) {
        Optional<AuthToken> authToken = getTokenObjForTokenStr(token);
        if (authToken.isEmpty()) throw new IllegalArgumentException("Token
            ↪ does not exist.");
        authTokens.remove(authToken);
    }
}

```

### 4.2.3. Serwis dostępu do danych

W listingu 4.14 przedstawiono implementację serwisu umożliwiającego wykonywanie podstawowych operacji *CRUD* (ang. *Create Read Update Delete* - tworzenie, odczyt, modyfikacja, usunięcie) na danych zawartych w bazie danych. Serwis korzysta z autentykacji zaimplementowanej w serwisie autentykacji przedstawionym w poprzednim podrozdziale.

Listing 4.14: Implementacja serwisu dostępu do danych.

```

package db2.carrental.services;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.dao.EmptyResultDataAccessException;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Service;

import java.lang.reflect.Method;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.NoSuchElementException;

@Service
public class DataService {

    private final AuthService authService;

    @Autowired
    public DataService(AuthService authService) {
        this.authService = authService;
    }
}

```

```

public <T extends JpaRepository> ResponseEntity<Object> getAll(T
    ↪ repository, String authToken, int permissionLevel){
    Map<String, Object> resp = new LinkedHashMap<>();
    if (authService.isUserAuthenticated(authToken, permissionLevel)) {
        resp.put("status", "success");
        resp.put("answer", repository.findAll());
        return ResponseEntity.ok(resp);
    } else {
        resp.put("status", "error");
        resp.put("answer", "User authentication failed.");
        return ResponseEntity.badRequest().body(resp);
    }
}

public <T extends JpaRepository> ResponseEntity<Object> getOne(T
    ↪ repository, String id, String authToken, boolean parseInt, int
    ↪ permissionLevel) {
    try {
        Map<String, Object> resp = new LinkedHashMap<>();
        if (authService.isUserAuthenticated(authToken, permissionLevel)
            ↪ ) {
            resp.put("status", "success");
            resp.put("answer", repository.findById(parseInt? Integer.
                ↪ parseInt(id) : id).orElseThrow(NoSuchElementException
                ↪ ::new) );
            return ResponseEntity.ok().body(resp);
        } else {
            resp.put("status", "error");
            resp.put("answer", "User authentication failed.");
            return ResponseEntity.badRequest().body(resp);
        }
    } catch (NumberFormatException e) {
        Map<String, Object> resp = new LinkedHashMap<>();
        resp.put("status", "error");
        resp.put("answer", "Wrong id format." + (e.getMessage().length
            ↪ () > 0 ? "Integer.parseInt(): Exception message: " + e.
            ↪ getMessage() : ""));
        return ResponseEntity.badRequest().body(resp);
    } catch (Throwable e) {
        Map<String, Object> resp = new LinkedHashMap<>();
        resp.put("status", "error");
        resp.put("answer", "Data not found. " + (e.getMessage() != null
            ↪ && e.getMessage().length() > 0 ? " Exception message: "
            ↪ + e.getMessage() : ""));
        return ResponseEntity.badRequest().body(resp);
    }
}

public <T extends JpaRepository> ResponseEntity<Object> deleteOne(T
    ↪ repository, String id, String authToken, boolean parseInt, int
    ↪ permissionLevel) {
    try {
        Map<String, Object> resp = new LinkedHashMap<>();
        if (authService.isUserAuthenticated(authToken, permissionLevel)
            ↪ ) {
            resp.put("status", "success");
            resp.put("answer", "Succesfully deleted.");
            repository.deleteById(parseInt? Integer.parseInt(id) : id);
            return ResponseEntity.ok(resp);
        } else {

```

```

        resp.put("status", "error");
        resp.put("answer", "User authentication failed.");
        return ResponseEntity.badRequest().body(resp);
    }
} catch (NumberFormatException e) {
    Map<String, Object> resp = new LinkedHashMap<>();
    resp.put("status", "error");
    resp.put("answer", "Wrong id format." + (e.getMessage() != null
        ↪ && e.getMessage().length() > 0 ? "Integer.parseInt():
        ↪ Exception message: " + e.getMessage() : ""));
    return ResponseEntity.badRequest().body(resp);
} catch (EmptyResultDataAccessException e) {
    Map<String, Object> resp = new LinkedHashMap<>();
    resp.put("status", "error");
    resp.put("answer", "Data not found. " + (e.getMessage() != null
        ↪ && e.getMessage().length() > 0 ? " Exception message: "
        ↪ + e.getMessage() : ""));
    return ResponseEntity.badRequest().body(resp);
}
}

public <T extends JpaRepository, G> ResponseEntity<Object> post(T
    ↪ repository, G entity, String authToken, int permissionLevel) {
    try {
        Map<String, Object> resp = new LinkedHashMap<>();
        if (authService.isUserAuthenticated(authToken, permissionLevel)
            ↪ ) {
            resp.put("status", "success");
            resp.put("answer", "Successfully added.");
            Object ent = repository.save(entity);
            resp.put("answer", ent);
            return ResponseEntity.ok(resp);
        } else {
            resp.put("status", "error");
            resp.put("answer", "User authentication failed.");
            return ResponseEntity.badRequest().body(resp);
        }
    } catch (Exception e) {
        Map<String, Object> resp = new LinkedHashMap<>();
        resp.put("status", "error");
        resp.put("answer", "Exception occurred." + (e.getMessage() !=
            ↪ null && e.getMessage().length() > 0 ? " Exception message
            ↪ : " + e.getMessage() : ""));
        return ResponseEntity.badRequest().body(resp);
    }
}

public <T extends JpaRepository, G> ResponseEntity<Object> put(T
    ↪ repository, G newEntity, String id, String authToken, boolean
    ↪ parseInt, int permissionLevel) {
    try {
        Map<String, Object> resp = new LinkedHashMap<>();
        if (authService.isUserAuthenticated(authToken, permissionLevel)
            ↪ ) {
            resp.put("status", "success");
            resp.put("answer", "Successfully updated. ");
            G entity = (G) repository.findById(parseInt?Integer.
                ↪ parseInt(id):id).orElseThrow(NoSuchElementException::
                ↪ new);
            Class<?> cls = newEntity.getClass();

```

```

Method[] methods = cls.getMethods();
for(Method setter : methods){
    if(setter.getName().contains("set")){
        for(Method getter: methods){
            if(getter.getName().equals("get" + setter.
                ↪ getName().substring(3))){
                System.out.println("Invoking" + entity.
                    ↪ toString() + "." + setter.getName() +
                    ↪ "(" + getter.getName() + "(" +
                    ↪ newEntity.toString() + ")"); //TODO:
                    ↪ remove this line
                setter.invoke(entity, getter.invoke(
                    ↪ newEntity));
            }
        }
    }
}
repository.save(entity);
return ResponseEntity.ok().body(resp);
} else {
    resp.put("status", "error");
    resp.put("answer", "User authentication failed.");
    return ResponseEntity.badRequest().body(resp);
}
} catch (NumberFormatException e) {
    Map<String, Object> resp = new LinkedHashMap<>();
    resp.put("status", "error");
    resp.put("answer", "Wrong id format." + (e.getMessage().length
        ↪ () > 0 ? "Integer.parseInt(): Exception message: " + e.
        ↪ getMessage() : ""));
    return ResponseEntity.badRequest().body(resp);
} catch (Throwable e) {
    Map<String, Object> resp = new LinkedHashMap<>();
    resp.put("status", "error");
    resp.put("answer", "Data not found. " + (e.getMessage() != null
        ↪ && e.getMessage().length() > 0 ? " Exception message: "
        ↪ + e.getMessage() : ""));
    return ResponseEntity.badRequest().body(resp);
}
}
}

```

#### 4.2.4. Implementacja komunikacji z serwerem

Komunikacja z serwerem odbywa się przy użyciu protokołu HTTP. W listniegu 4.15 została przedstawiona implementacja metod wywoływanych podczas przesłania żądań *GET*, *POST*, *PUT*, *DELETE* na przykładzie encji klienta (dostęp do pozostałych encji został zaimplementowany analogicznie). Wywoływane metody korzystają z metod serwisu dostępu do danych.

Listing 4.15: Implementacja komunikacji z serwerem.

```

@RestController
@RequestMapping(path = "/data")
public class DataController {

    private final DataService dataService;
    private final KlienciRepository klienciRepository;
    \\.

```



```

@Autowired
public DataController(DataService dataService, KlienciRepository
    ↪ klienciRepository ...) {
    this.dataService = dataService;
    this.klienciRepository = klienciRepository;
    ...
}

@GetMapping("/klienci")
public ResponseEntity<Object> getAllClients(@RequestParam String
    ↪ authToken) {
    return dataService.getAll(klienciRepository, authToken, 1);
}
//...
@GetMapping("klienci/{id}")
public ResponseEntity<Object> getOneCustomer(@PathVariable("id") String
    ↪ id, @RequestParam String authToken) {
    return dataService.getOne(klienciRepository, id, authToken, true,
        ↪ 1);
}
//...
@DeleteMapping("klienci/{id}")
public ResponseEntity<Object> deleteCustomer(@PathVariable("id") String
    ↪ id, @RequestParam String authToken) {
    return dataService.deleteOne(klienciRepository, id, authToken, true
        ↪ , 1);
}
//...
@PostMapping("/klienci")
public ResponseEntity<Object> postClients(@RequestBody Klienci klienci,
    ↪ @RequestParam String authToken) {
    return dataService.post(klienciRepository, klienci, authToken, 1);
}
//...
@PutMapping("/klienci/{id}")
public ResponseEntity<Object> putCustomer(@RequestBody Klienci entity,
    ↪ @PathVariable String id, @RequestParam String authToken){
    return dataService.put(klienciRepository, entity, id, authToken,
        ↪ true, 1);
}
}

```

## 4.3. Realizacja elementów aplikacji

### 4.3.1. Autoryzacja

Na rysunku 4.16 przedstawiono metodę, która jest wykonywana po naciśnięciu przycisku *Zaloguj się* w panelu logowania. Logowanie odbywa się poprzez przesłania do serwera bazy danych loginu i hasła użytkownika, które następnie są wyszukiwane przez serwer w bazie danych. Jeżeli użytkownik o podanych danych istnieje i posiada odpowiedni poziom uprawnień, zostaje utworzona sesja, której serwer nadaje token sesji który jest losowo wygenerowaną sekwencją znaków. Serwer przechowuje wszystkie tokeny aktualnie otwartych sesji. Od momentu zalogowania token sesji jest przesyłany z każdym zapytaniem aplikacji do serwera w celu autoryzacji użytkownika. Jeżeli przesłany token sesji nie znajduje się na liście przechowywanej przez serwer, następuje odmowa dostępu i przesyłany jest komunikat o odmowie dostępu. Aplikacja obsługuje

komunikaty o odmowie dostępu przesyłane przez serwer i odpowiednio informuje użytkownika o niepowodzeniu wykonania operacji. Wylogowanie użytkownika powoduje usunięcie jego tokenu sesji z listy sesji w serwerze bazy danych, dzięki czemu przy próbie autoryzacji za pomocą tego tokenu po jego dezaktywacji nastąpi odmowa dostępu ze strony serwera bazy danych.

```
const login = async (data, permissionLevel) => {
  console.log(data);
  const requestString = new URLSearchParams(data);
  const url =
    permissionLevel === 0
      ? `${SERVER_ADDRESS}/auth/login`
      : `${SERVER_ADDRESS}/auth/admin`;
  console.log(`${url}?${requestString.toString()}`);
  try {
    setIsLoading( value: true);
    const response = await fetch( input: `${url}?${requestString.toString()}`);
    const resp = await response.json();
    if (!response.ok) {
      showErrorAlert(resp.message);
    } else {
      setAuthToken(resp.token);
      setIsLoggedIn( value: true);
      navigate("/");
      console.log(resp);
      showSuccessAlert("Pomyślnie zalogowano.");
      setIsLoading( value: false);
    }
  } catch (e) {
    showErrorAlert("Nie można było uzyskać połączenia z serwerem.");
  }
  setIsLoading( value: false);
};
```

Listing 4.16: Metoda realizująca autoryzację użytkownika.

### 4.3.2. Implementacja interfejsu dostępu do bazy danych

Interfejs dostępu do bazy danych wykorzystuje API serwera bazy danych umożliwiające wykonywanie operacji *GET*, *PUT*, *POST*, *DELETE*[1] na dowolnej encji w bazie danych. Interfejs został zbudowany z wykorzystaniem komponentów dostępnych w bibliotece *MUI*[2]. Z każdą wykonywaną operacją jest wykonywane zapytanie do serwera bazy danych. Do każdego zapytania jest dołączany token uwierzytelniający w celu poprawnej autoryzacji. Aplikacja obsługuje odpowiedzi od serwera bazy danych, w szczególności potwierdzenie wykonania operacji, odmowę wykonania operacji, bądź też odbieranie, przetwarzanie, oraz wyświetlanie w przyjaznej dla użytkownika formie danych przesyłanych przez serwer.

# Rozdział 5

## Testowanie systemu

### 5.1. Instalacja systemu

#### 5.1.1. Instalacja i uruchamianie systemu zarządzania bazą danych

Pierwszą czynnością, którą należy wykonać w celu instalacji systemu jest instalacja systemu zarządzania bazą danych MySQL. Procedura instalacji systemu na platformie Windows, oraz pliki instalacyjne są dostępne na stronie internetowej [3]. Po uruchomieniu systemu MySQL, należy przy użyciu narzędzia umożliwiającego połączenie się z bazą danych (np. MySQL Workbench [7]) uruchomić skrypt tworzący bazę danych zawarty w pliku "car\_rental\_script.sql". Po poprawnym wykonaniu skryptu baza danych jest gotowa do użycia. Warto zaznaczyć że uruchamianie skryptu jest konieczne tylko przy inicjalizacji bazy danych, nie należy wywoływać skryptu kolejny raz, ponieważ spowoduje to utratę danych wpisanych do bazy danych. Jedyną sytuacją w której skrypt może zostać uruchomiony kolejny raz, jest sytuacja, kiedy mamy na celu usunięcie wszystkich danych w tabelach, czyli inaczej mówiąc "zresetowanie" zawartości bazy danych. Każde kolejne uruchomienie bazy danych można wykonać przy użyciu polecenia:

```
mysqld --console
```

z linii komend uruchomionej w trybie administratora w folderze bin w lokalizacji systemu zarządzania bazą danych.

#### 5.1.2. Instalacja i uruchamianie serwera bazy danych

Aby uruchomienie serwera bazy danych było możliwe, na komputerze powinno być zainstalowane Środowisko Uruchomieniowe Javy (ang. *Java Runtime Environment*) w wersji 11 lub nowszej. Środowisko jest dostępne na stronie internetowej Oracle [8]. W celu uruchomienia serwera bazy danych należy uruchomić linię komend w trybie administratora, a następnie wywołać następujące polecenie:

```
java -jar car-rental-database-server.jar
```

#### 5.1.3. Instalacja i uruchamianie serwera aplikacji

Aby uruchomienie serwera aplikacji było możliwe, na komputerze powinno być zainstalowane środowisko NodeJS. Środowisko jest dostępne na stronie internetowej [4]. W celu uruchomienia serwera aplikacji, należy uruchomić linię komend w trybie administratora, a następnie wywołać następujące polecenie:

```
serve -s application-server
```

## 5.2. Testowanie opracowanych funkcji systemu

### 5.2.1. Testowanie funkcji zrealizowanych w panelu logowania użytkownika

#### Autoryzacja użytkownika

Na rysunku 5.1 przedstawiono wprowadzenie poprawnych danych do panelu logowania. Na rysunku 5.2 został przedstawiony wynik próby zalogowania z wprowadzeniem poprawnych danych. Użytkownik został poprawnie zalogowany i przeniesiony na stronę panelu użytkownika. Wyświetlone zostało również powiadomienie o pomyślnym przebiegu procesu autoryzacji.

**Logowanie**

Email \*  
user@example.com

Hasło \*  
....

ZALOGUJ SIĘ

[Rejestracja](#)

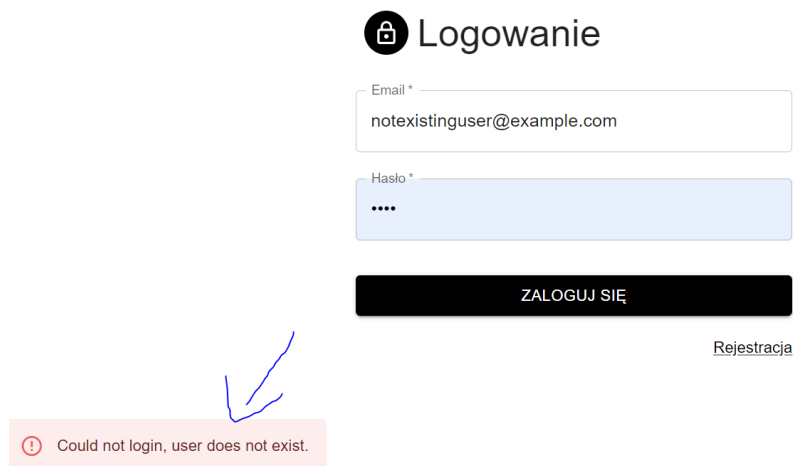
Rys. 5.1: Logowanie użytkownika z poprawnymi danymi.

Wypożyczalnia samochodów										WYLOGUJ SIĘ
Dostępne samochody										
ID	Dostępność	Miasto	Marka	Model	Data produkcji	Cena (za dzień)	Spalanie[L/100km]	L. Miejsc	L. Drzwi	Pojem zbiór
4	DOSTĘPNY	POZNAN	BMW	E90	2011-06-27	150	12	5	5	
9	DOSTĘPNY	POZNAN	BMW	E90	2017-02-24	150	12	5	5	
13	DOSTĘPNY	KRAKOW	BMW	E90	2012-05-19	150	12	5	5	
7	DOSTĘPNY	WARSZAWA	MITSUBISHI	ASX	2018-01-17	175	7	5	5	

✓ Pomyślnie zalogowano.

Rys. 5.2: Logowanie użytkownika z poprawnymi danymi. - wynik

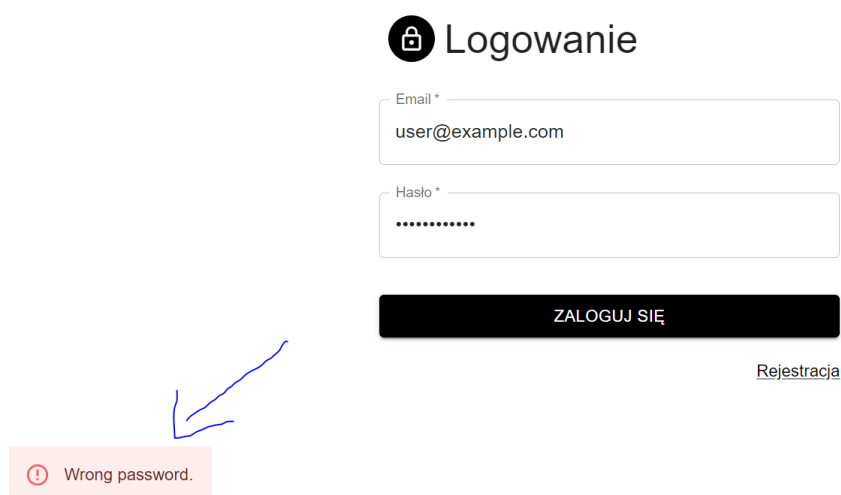
Na rysunku 5.3 przedstawiono wynik próby zalogowania z wprowadzeniem nieistniejącej nazwy użytkownika. Nastąpiła odmowa dostępu. Zostało wyświetlone powiadomienie o wprowadzeniu nieprawidłowej nazwy użytkownika.



The screenshot shows a login form titled "Logowanie" with a lock icon. It contains two input fields: "Email \*" with the value "notexistinguser@example.com" and "Hasło \*" with masked characters "....". Below the fields is a black button labeled "ZALOGUJ SIĘ". To the right of the button is a link labeled "Rejestracja". At the bottom left, a red error message box displays the text "Could not login, user does not exist." with a red exclamation mark icon. A blue hand-drawn arrow points from the error message box towards the email input field.

Rys. 5.3: Logowanie użytkownika z niepoprawną nazwą - wynik.

Na rysunku 5.4 przedstawiono wynik próby zalogowania z wprowadzeniem nieprawidłowego hasła i nazwy istniejącego użytkownika. Nastąpiła odmowa dostępu. Zostało wyświetlone powiadomienie o wprowadzeniu nieprawidłowego hasła.



The screenshot shows the same login form as in Figure 5.3. The "Email \*" field now contains the value "user@example.com". The "Hasło \*" field is masked with ".....". The "ZALOGUJ SIĘ" button and the "Rejestracja" link remain. At the bottom left, a red error message box displays the text "Wrong password." with a red exclamation mark icon. A blue hand-drawn arrow points from the error message box towards the password input field.

Rys. 5.4: Logowanie użytkownika z niepoprawnym hasłem - wynik.

## Rejestracja użytkownika

Na rysunku 5.5 przedstawiono zawartość tabeli *UZYTKOWNICY* w bazie danych przed rejestracją nowego użytkownika. Rysunek 5.6 przedstawia dane wpisane w panelu rejestracji użytkownika. Po rejestracji użytkownika w tabeli *UZYTKOWNICY* w bazie danych został dodany nowy rekord z danymi nowego użytkownika co zostało przedstawione na rysunku 5.7.

	LOGIN	USER_GROUP_ID	PASSWORD_HASH
▶	admin@example.com	1	21232f297a57a5a743894a0e4a801fc3
	user@example.com	0	ee11cbb19052e40b07aac0ca060c23ee
✱	NULL	NULL	NULL

Rys. 5.5: Zawartość tabeli *UZYTKOWNICY* w bazie danych przed rejestracją nowego użytkownika.



## Rejestracja

Email \*

Hasło \*

Powtórz hasło \*

ZAREJESTRUJ SIĘ

[Logowanie](#)

Rys. 5.6: Wpisane dane w panelu rejestracji użytkownika.

	LOGIN	USER_GROUP_ID	PASSWORD_HASH
▶	admin@example.com	1	21232f297a57a5a743894a0e4a801fc3
	newuser@example.com	0	0354d89c28ec399c00d3cb2d094cf093
	user@example.com	0	ee11cbb19052e40b07aac0ca060c23ee
✱	NULL	NULL	NULL

Rys. 5.7: Zawartość tabeli *UZYTKOWNICY* w bazie danych po rejestracji nowego użytkownika.

## 5.2.2. Testowanie funkcji zrealizowanych w panelu logowania pracownika

### Autoryzacja pracownika

Na rysunku 5.8 przedstawiono wprowadzenie poprawnych danych do panelu logowania. Na rysunku 5.9 został przedstawiony wynik próby zalogowania z wprowadzeniem poprawnych danych. Pracownik został poprawnie zalogowany i przeniesiony na stronę panelu pracownika. Wyświetlone zostało również powiadomienie o pomyślnym przebiegu procesu autoryzacji.

**Admin panel**

Email \*

admin@example.com

Hasło \*

.....

**ZALOGUJ SIĘ**

Rys. 5.8: Logowanie pracownika z poprawnymi danymi.

**Wypożyczalnia samochodów** **WYLOGUJ SIĘ**

**Lista samochodów** **DODAJ SAMOCHÓD**

Id	Lokalizacja	Model	Nr rejestracyjny	Data produkcji	Data rejestracji	Data przeglądu	Przebieg		
1	WROCLAW UL. JAGIELLY 11	QUASHQAI	DW 35542	2013-08-28	2013-09-22	2021-06-14	244634	EDYTUJ	USUŃ
2	WARSZAWA UL. KOSCIUSZKI 7	SUPRA	DW 45932	2012-02-27	2012-03-20	2021-06-28	457293	EDYTUJ	USUŃ
3	KRAKOW UL. PILSUDSKIEGO	QUASHQAI	DW 64823	2013-03-23	2013-04-28	2021-02-28	231435	EDYTUJ	USUŃ

Pomyślnie zalogowano.

Rys. 5.9: Logowanie pracownika z poprawnymi danymi. - wynik

Na rysunku 5.10 przedstawiono wynik próby zalogowania z wprowadzeniem nieistniejącej nazwy użytkownika. Nastąpiła odmowa dostępu. Zostało wyświetlone powiadomienie o wprowadzeniu nieprawidłowej nazwy użytkownika.

The screenshot shows the 'Admin panel' login interface. It features a title 'Admin panel' with a lock icon. Below the title are two input fields: 'Email \*' containing 'notexistinguser@example.com' and 'Hasło \*' with masked characters. A black button labeled 'ZALOGUJ SIĘ' is positioned below the fields. At the bottom, a red error message box states: 'Could not login, user does not exist.'

Rys. 5.10: Logowanie pracownika z niepoprawną nazwą użytkownika - wynik

Na rysunku 5.11 przedstawiono wynik próby zalogowania z wprowadzeniem nieprawidłowego hasła i nazwy istniejącego użytkownika. Nastąpiła odmowa dostępu. Zostało wyświetlone powiadomienie o wprowadzeniu nieprawidłowego hasła.

The screenshot shows the 'Admin panel' login interface. It features a title 'Admin panel' with a lock icon. Below the title are two input fields: 'Email \*' containing 'admin@example.com' and 'Hasło \*' with masked characters. A black button labeled 'ZALOGUJ SIĘ' is positioned below the fields. At the bottom, a red error message box states: 'Wrong password.' A blue checkmark is drawn next to the error message.

Rys. 5.11: Logowanie pracownika z niepoprawnym hasłem - wynik.



## Sprawdzenie uprawnień

Na rysunku 5.12 przedstawiono wynik próby zalogowania z podaniem poprawnych danych użytkownika który nie jest uprawniony do logowania się do panelu pracownika. Nastąpiła odmowa dostępu. Zostało wyświetlone powiadomienie o braku uprawnień.

The screenshot shows a login interface titled "Admin panel" with a lock icon. It contains two input fields: "Email \*" with the value "user@example.com" and "Hasło \*" with masked characters "....". Below the fields is a black button labeled "ZALOGUJ SIĘ". At the bottom, a red error message box states: "Could not login, user permissions insufficient".

Rys. 5.12: Logowanie użytkownika z niewystarczającym poziomem uprawnień.

### 5.2.3. Testowanie funkcji zrealizowanych w panelu użytkownika

#### Wyświetlenie panelu z dostępnymi samochodami

Na rysunku 5.13 przedstawiono wygląd panelu użytkownika po poprawnym zalogowaniu. Panel przedstawia dane pobrane z bazy danych, które są dostępne w widoku *DOSTEPNE\_SAMOCODY*. Na rysunku 5.14 przedstawiono wynik wykonania zapytania `SELECT * FROM DOSTEPNE_SAMOCODY` w środowisku *MySQL Workbench*. Dane przedstawione w panelu użytkownika zgadzają się z danymi uzyskanymi poprzez wykonanie zapytania.

Wypożyczalnia samochodów										WYLOGUJ SIĘ
Dostępne samochody										
ID	Dostępność	Miasto	Marka	Model	Data produkcji	Cena (za dzień)	Spalanie[L/100km]	L. Miejsc	L. Drzwi	Pojemr. zbiór
4	DOSTEPNY	POZNAN	BMW	E90	2011-06-27	150	12	5	5	
9	DOSTEPNY	POZNAN	BMW	E90	2017-02-24	150	12	5	5	
13	DOSTEPNY	KRAKOW	BMW	E90	2012-05-19	150	12	5	5	
7	DOSTEPNY	WARSZAWA	MITSUBISHI	ASX	2018-01-17	175	7	5	5	

Rys. 5.13: Wygląd panelu użytkownika po zalogowaniu.

ID_SAM	STATUS	MIASTO	MARKA	MODEL	DATA_PRODUKCJI	CENA	SPALANIE	L_MIEJSC	L_DRZWI	POJEMNOSC_ZBIORNIKA	TYP_PALIWA	POJEMNOSC_SILNIKA	MOC	RODZAJ_KAROSERII
4	DOSTEPNY	POZNAN	BMW	E90	2011-06-27	150	12	5	5	65	BENZYN	2000	150	SEDAN
9	DOSTEPNY	POZNAN	BMW	E90	2017-02-24	150	12	5	5	65	BENZYN	2000	150	SEDAN
13	DOSTEPNY	KRAKOW	BMW	E90	2012-05-19	150	12	5	5	65	BENZYN	2000	150	SEDAN
7	DOSTEPNY	WARSZAWA	MITSUBISHI	ASX	2018-01-17	175	7	5	5	55	BENZYN	2200	180	SUV
10	DOSTEPNY	GDANSK	MITSUBISHI	ASX	2010-02-15	175	7	5	5	55	BENZYN	2200	180	SUV
12	DOSTEPNY	WARSZAWA	MITSUBISHI	ASX	2019-05-22	175	7	5	5	55	BENZYN	2200	180	SUV
14	DOSTEPNY	POZNAN	MITSUBISHI	ASX	2015-06-17	175	7	5	5	55	BENZYN	2200	180	SUV
17	DOSTEPNY	WARSZAWA	MITSUBISHI	ASX	2019-05-25	175	7	5	5	55	BENZYN	2200	180	SUV
22	DOSTEPNY	WARSZAWA	MITSUBISHI	ASX	2012-04-10	175	7	5	5	55	BENZYN	2200	180	SUV
25	DOSTEPNY	GDANSK	MITSUBISHI	ASX	2011-07-15	175	7	5	5	55	BENZYN	2200	180	SUV
1	DOSTEPNY	WROCLAW	NISSAN	QUAS...	2013-08-28	200	5	5	5	75	DIESEL	1989	120	SUV
3	DOSTEPNY	KRAKOW	NISSAN	QUAS...	2013-03-23	200	5	5	5	75	DIESEL	1989	120	SUV
11	DOSTEPNY	WROCLAW	NISSAN	QUAS...	2010-07-28	200	5	5	5	75	DIESEL	1989	120	SUV
16	DOSTEPNY	WROCLAW	NISSAN	QUAS...	2010-07-10	200	5	5	5	75	DIESEL	1989	120	SUV
23	DOSTEPNY	KRAKOW	NISSAN	QUAS...	2013-01-26	200	5	5	5	75	DIESEL	1989	120	SUV
24	DOSTEPNY	POZNAN	NISSAN	QUAS...	2012-03-21	200	5	5	5	75	DIESEL	1989	120	SUV
2	DOSTEPNY	WARSZAWA	TOYOTA	SUPRA	2012-02-27	250	15	2	3	70	BENZYN	4000	450	COUPE
5	DOSTEPNY	GDANSK	TOYOTA	SUPRA	2020-06-20	250	15	2	3	70	BENZYN	4000	450	COUPE
6	DOSTEPNY	WROCLAW	TOYOTA	SUPRA	2014-07-15	250	15	2	3	70	BENZYN	4000	450	COUPE
8	DOSTEPNY	KRAKOW	VOLKSWAG...	GOLF V	2017-03-12	100	7	5	5	60	DIESEL	1600	89	HATCHBACK
19	DOSTEPNY	POZNAN	VOLKSWAG...	GOLF V	2016-02-10	100	7	5	5	60	DIESEL	1600	89	HATCHBACK

Rys. 5.14: Zawartość widoku *DOSTEPNE\_SAMOCCHODY* w bazie danych.

## 5.2.4. Testowanie funkcji zrealizowanych w panelu pracownika

### Wyświetlenie informacji o samochodach

Na rysunku 5.15 przedstawiono wygląd panelu pracownika po poprawnym zalogowaniu. Panel przedstawia dane pobrane z bazy danych dostępne w tabeli *SAMOCCHODY*. Na rysunku 5.16 przedstawiono wynik wykonania zapytania `SELECT * FROM SAMOCCHODY` w środowisku *MySQL Workbench*. Dane przedstawione w panelu pracownika są zgodne z danymi uzyskanymi poprzez wykonanie zapytania.

Wypożyczalnia samochodów								WYLOGUJ SIĘ	
Lista samochodów								DODAJ SAMOCCHÓD	
Id	Lokalizacja	Model	Nr rejestracyjny	Data produkcji	Data rejestracji	Data przeglądu	Przebieg		
1	WROCLAW UL. JAGIELLY 11	QUASHQAI	DW 35542	2013-08-28	2013-09-22	2021-06-14	244634	EDYTUJ	USUŃ
2	WARSZAWA UL. KOSCIUSZKI 7	SUPRA	DW 45932	2012-02-27	2012-03-20	2021-06-28	457293	EDYTUJ	USUŃ
3	KRAKOW UL. PILUSUDSKIEGO 5	QUASHQAI	DW 64823	2013-03-23	2013-04-28	2021-02-28	231435	EDYTUJ	USUŃ
4	POZNAN UL. 3 MAJA 2	E90	DW 93880	2011-06-27	2011-07-11	2021-03-13	245959	EDYTUJ	USUŃ

Rys. 5.15: Wygląd panelu pracownika po zalogowaniu.

	ID_SAMOCODU	ID_WYPOZYCZALNI	ID_MODELU	NR_REJESTRACYJNY	DATA_PRODUKCJI	DATA_REJESTRACJI	DATA_PRZEGŁADU	PRZEBIEG
1	1		NIS_Q	DW 35542	2013-08-28	2013-09-22	2021-06-14	244634
2	2		TOY_SUP	DW 45932	2012-02-27	2012-03-20	2021-06-28	457293
3	3		NIS_Q	DW 64823	2013-03-23	2013-04-28	2021-02-28	231435
4	4		BMW_E90	DW 93880	2011-06-27	2011-07-11	2021-03-13	245959
5	5		TOY_SUP	DW 85098	2020-06-20	2020-07-10	2021-07-16	570529
6	1		TOY_SUP	DW 18401	2014-07-15	2014-08-18	2021-07-15	223357
7	2		MIT_ASX	DW 26683	2018-01-17	2018-02-13	2021-07-22	237711
8	3		VW_G5	DW 78168	2017-03-12	2017-04-16	2021-05-28	126998
9	4		BMW_E90	DW 37421	2017-02-24	2017-03-20	2021-01-22	101755
10	5		MIT_ASX	DW 53849	2010-02-15	2010-03-28	2021-08-10	425575
11	1		NIS_Q	DW 52589	2010-07-28	2010-08-21	2021-08-27	458151
12	2		MIT_ASX	DW 60903	2019-05-22	2019-06-11	2021-07-13	311494
13	3		BMW_E90	DW 41786	2012-05-19	2012-06-27	2021-05-13	391266
14	4		MIT_ASX	DW 46983	2015-06-17	2015-07-26	2021-03-16	547332
15	5		MIT_ASX	DW 30408	2010-08-12	2010-09-26	2021-03-10	241901
16	1		NIS_Q	DW 94449	2010-07-10	2010-08-25	2021-07-27	242694
17	2		MIT_ASX	DW 22568	2019-05-25	2019-06-18	2021-08-11	123241
18	3		VW_G5	DW 69614	2011-01-19	2011-02-11	2021-04-11	287028
19	4		VW_G5	DW 26289	2016-02-10	2016-03-28	2021-01-28	554320
20	5		NIS_Q	DW 58206	2015-08-16	2015-09-22	2021-04-28	159067
21	1		TOY_SUP	DW 79920	2019-07-13	2019-08-27	2021-06-23	156243
22	2		MIT_ASX	DW 76695	2012-04-10	2012-05-22	2021-04-12	410277
23	3		NIS_Q	DW 87912	2013-01-26	2013-02-24	2021-08-22	186186
24	4		NIS_Q	DW 30367	2012-03-21	2012-04-23	2021-08-11	466248
25	5		MIT_ASX	DW 36324	2011-07-15	2011-08-14	2021-08-23	576701

Rys. 5.16: Zawartość tabeli *SAMOCODY* w bazie danych.

## Usunięcie samochodu

Na rysunku 5.17 przedstawiono wygląd panelu pracownika przed usunięciem samochodu. Na rysunku 5.18 przedstawiono zawartość tabeli *SAMOCODY* w bazie danych przed usunięciem samochodu. Został wciśnięty przycisk usuń przy samochodzie o *ID* równym 1. Na rysunku 5.19 przedstawiono wygląd panelu pracownika po usunięciu samochodu. Zgodnie z oczekiwaniem usuwany samochód zniknął z listy. Zostało również wyświetlone powiadomienie o powodzeniu wykonania operacji usuwania. Na rysunku 5.20 przedstawiono zawartość tabeli *SAMOCODY* w bazie danych po usunięciu samochodu. Zgodnie z oczekiwaniem rekord zawierający samochód o *ID* równym 1 został usunięty.

Wypożyczalnia samochodów								WYLOGUJ SIĘ
Lista samochodów								DODAJ SAMOCHÓD
Id	Lokalizacja	Model	Nr rejestracyjny	Data produkcji	Data rejestracji	Data przeglądu	Przebieg	
1	WROCLAW UL. JAGIELLY 11	QUASHQAI	DW 35542	2013-08-28	2013-09-22	2021-06-14	244634	<div>EDYTUJ</div> <div>USUŃ</div>
2	WARSZAWA UL. KOSCIUSZKI 7	SUPRA	DW 45932	2012-02-27	2012-03-20	2021-06-28	457293	<div>EDYTUJ</div> <div>USUŃ</div>
3	KRAKOW UL. PILUSDSKIEGO 5	QUASHQAI	DW 64823	2013-03-23	2013-04-28	2021-02-28	231435	<div>EDYTUJ</div> <div>USUŃ</div>

Rys. 5.17: Panel pracownika przed usunięciem samochodu.

	ID_SAMOCODU	ID_WYPOZYCZALNI	ID_MODELU	NR_REJESTRACYJNY	DATA_PRODUKCJI	DATA_REJESTRACJI	DATA_PRZEGŁADU	PRZEBIEG
▶	1	1	NIS_Q	DW 35542	2013-08-28	2013-09-22	2021-06-14	244634
	2	2	TOY_SUP	DW 45932	2012-02-27	2012-03-20	2021-06-28	457293
	3	3	NIS_Q	DW 64823	2013-03-23	2013-04-28	2021-02-28	231435
	4	4	BMW_E90	DW 93880	2011-06-27	2011-07-11	2021-03-13	245959
	5	5	TOY_SUP	DW 85098	2020-06-20	2020-07-10	2021-07-16	570529
	6	1	TOY_SUP	DW 18401	2014-07-15	2014-08-18	2021-07-15	223357
	7	2	MIT_ASX	DW 26683	2018-01-17	2018-02-13	2021-07-22	237711
	8	3	VW_G5	DW 78168	2017-03-12	2017-04-16	2021-05-28	126998
	9	4	BMW_E90	DW 37421	2017-02-24	2017-03-20	2021-01-22	101755
	10	5	MIT_ASX	DW 53849	2010-02-15	2010-03-28	2021-08-10	425575
	11	1	NIS_Q	DW 52589	2010-07-28	2010-08-21	2021-08-27	458151
	12	2	MIT_ASX	DW 60903	2019-05-22	2019-06-11	2021-07-13	311494
	13	3	BMW_E90	DW 41786	2012-05-19	2012-06-27	2021-05-13	391266
	14	4	MIT_ASX	DW 46983	2015-06-17	2015-07-26	2021-03-16	547332
	15	5	MIT_ASX	DW 30408	2010-08-12	2010-09-26	2021-03-10	241901
	16	1	NIS_Q	DW 94449	2010-07-10	2010-08-25	2021-07-27	242694
	17	2	MIT_ASX	DW 22568	2019-05-25	2019-06-18	2021-08-11	123241
	18	3	VW_G5	DW 69614	2011-01-19	2011-02-11	2021-04-11	287028
	19	4	VW_G5	DW 26289	2016-02-10	2016-03-28	2021-01-28	554320
	20	5	NIS_Q	DW 58206	2015-08-16	2015-09-22	2021-04-28	159067
	21	1	TOY_SUP	DW 79920	2019-07-13	2019-08-27	2021-06-23	156243
	22	2	MIT_ASX	DW 76695	2012-04-10	2012-05-22	2021-04-12	410277
	23	3	NIS_Q	DW 87912	2013-01-26	2013-02-24	2021-08-22	186186
	24	4	NIS_Q	DW 30367	2012-03-21	2012-04-23	2021-08-11	466248
	25	5	MIT_ASX	DW 36324	2011-07-15	2011-08-14	2021-08-23	576701

samochody 3 x

Rys. 5.18: Zawartość tabeli *SAMOCODY* w bazie danych przed usunięciem samochodu.

Wypożyczalnia samochodów								WYLOGUJ SIĘ	
Lista samochodów								DODAJ SAMOCHÓD	
Id	Lokalizacja	Model	Nr rejestracyjny	Data produkcji	Data rejestracji	Data przeglądu	Przebieg		
2	WARSZAWA UL. KOSCIUSZKI 7	SUPRA	DW 45932	2012-02-27	2012-03-20	2021-06-28	457293	EDYTUJ	USUŃ
3	KRAKOW UL. PILUSUDSKIEGO 5	QUASHQAI	DW 64823	2013-03-23	2013-04-28	2021-02-28	231435	EDYTUJ	USUŃ
4	POZNAN UL. 3 MAJA 2	E90	DW 93880	2011-06-27	2011-07-11	2021-03-13	245959	EDYTUJ	USUŃ
✓ Pomyślnie usunięto samochód z bazy danych.			DW 85098	2020-06-20	2020-07-10	2021-07-16	570529	EDYTUJ	USUŃ

Rys. 5.19: Panel pracownika po usunięciu samochodu.

	ID_SAMOCODU	ID_WYPOZYCZALNI	ID_MODELU	NR_REJESTRACYJNY	DATA_PRODUKCJI	DATA_REJESTRACJI	DATA_PRZEGLADU	PRZEBIEG
▶	2	2	TOY_SUP	DW 45932	2012-02-27	2012-03-20	2021-06-28	457293
	3	3	NIS_Q	DW 64823	2013-03-23	2013-04-28	2021-02-28	231435
	4	4	BMW_E90	DW 93880	2011-06-27	2011-07-11	2021-03-13	245959
	5	5	TOY_SUP	DW 85098	2020-06-20	2020-07-10	2021-07-16	570529
	6	1	TOY_SUP	DW 18401	2014-07-15	2014-08-18	2021-07-15	223357
	7	2	MIT_ASX	DW 26683	2018-01-17	2018-02-13	2021-07-22	237711
	8	3	VW_G5	DW 78168	2017-03-12	2017-04-16	2021-05-28	126998
	9	4	BMW_E90	DW 37421	2017-02-24	2017-03-20	2021-01-22	101755
	10	5	MIT_ASX	DW 53849	2010-02-15	2010-03-28	2021-08-10	425575
	11	1	NIS_Q	DW 52589	2010-07-28	2010-08-21	2021-08-27	458151
	12	2	MIT_ASX	DW 60903	2019-05-22	2019-06-11	2021-07-13	311494
	13	3	BMW_E90	DW 41786	2012-05-19	2012-06-27	2021-05-13	391266
	14	4	MIT_ASX	DW 46983	2015-06-17	2015-07-26	2021-03-16	547332
	15	5	MIT_ASX	DW 30408	2010-08-12	2010-09-26	2021-03-10	241901
	16	1	NIS_Q	DW 94449	2010-07-10	2010-08-25	2021-07-27	242694
	17	2	MIT_ASX	DW 22568	2019-05-25	2019-06-18	2021-08-11	123241
	18	3	VW_G5	DW 69614	2011-01-19	2011-02-11	2021-04-11	287028
	19	4	VW_G5	DW 26289	2016-02-10	2016-03-28	2021-01-28	554320
	20	5	NIS_Q	DW 58206	2015-08-16	2015-09-22	2021-04-28	159067
	21	1	TOY_SUP	DW 79920	2019-07-13	2019-08-27	2021-06-23	156243
	22	2	MIT_ASX	DW 76695	2012-04-10	2012-05-22	2021-04-12	410277
	23	3	NIS_Q	DW 87912	2013-01-26	2013-02-24	2021-08-22	186186
	24	4	NIS_Q	DW 30367	2012-03-21	2012-04-23	2021-08-11	466248
	25	5	MIT_ASX	DW 36324	2011-07-15	2011-08-14	2021-08-23	576701
	26	1	TOY_SUP	DW 63359	2017-07-20	2017-08-10	2021-01-12	521873

Rys. 5.20: Zawartość tabeli *SAMOCODY* w bazie danych po usunięciu samochodu.

## Modyfikacja danych dotyczących samochodu

Na rysunku 5.21 przedstawiono wygląd panelu pracownika przed modyfikacją samochodu. Rysunek 5.22 przedstawia zawartość tabeli *SAMOCODY* w bazie danych przed modyfikacją samochodu. Zmodyfikowany został samochód o *ID* równym 2. Nowe wartości atrybutów które zostały zapisane po modyfikacji są przedstawione na rysunku 5.23. Na rysunku 5.24 przedstawiono wygląd panelu pracownika po modyfikacji samochodu. Jak można zauważyć, zgodnie z oczekiwaniami atrybuty samochodu zostały zmienione, oraz zostało wyświetlone powiadomienie informujące o powodzeniu modyfikacji. Na rysunku 5.25 przedstawiono zawartość tabeli *SAMOCODY* w bazie danych po modyfikacji samochodu. Zgodnie z oczekiwaniami wartości atrybutów zostały zaktualizowane.

### Lista samochodów

DODAJ SAMOCHÓD							
Id	Lokalizacja	Model	Nr rejestracyjny	Data produkcji	Data rejestracji	Data przeglądu	Przebieg
2	WARSZAWA UL. KOSCIUSZKI 7	SUPRA	DW 45932	2012-02-27	2012-03-20	2021-06-28	457293
<div>EDYTUJ</div> <div>USUŃ</div>							

Rys. 5.21: Panel pracownika przed modyfikacją samochodu.

	ID_SAMOCODU	ID_WYPOZYCZALNI	ID_MODELU	NR_REJESTRACYJNY	DATA_PRODUKCJI	DATA_REJESTRACJI	DATA_PRZEGLADU	PRZEBIEG
▶	2	2	TOY_SUP	DW 45932	2012-02-27	2012-03-20	2021-06-28	457293
	3	3	NIS_Q	DW 64823	2013-03-23	2013-04-28	2021-02-28	231435
	4	4	BMW_E90	DW 93880	2011-06-27	2011-07-11	2021-03-13	245959
	5	5	TOY_SUP	DW 85098	2020-06-20	2020-07-10	2021-07-16	570529

Rys. 5.22: Zawartość tabeli *SAMOCODY* w bazie danych przed modyfikacją samochodu

Edytuj samochód

Wypożyczalnia \*  
GDANSK, UL. 11 LISTOPADA 1

Model samochodu \*  
BMW E90

Numer rejestracyjny \*  
XX XXXX

Data produkcji \*  
01.02.2021

Data rejestracji \*  
02.02.2021

Data przeglądu \*  
03.02.2021

Przebieg \*  
55555

Rys. 5.23: Dane dotyczące samochodu, które mają zostać zapisane po modyfikacji.

## Lista samochodów

DODAJ SAMOCHÓD

Id	Lokalizacja	Model	Nr rejestracyjny	Data produkcji	Data rejestracji	Data przeglądu	Przebieg		
2	GDANSK UL. 11 LISTOPADA 1	E90	XX XXXX	2021-02-01	2021-02-02	2021-02-03	55555	EDYTUJ	USUŃ
3	KRAKOW UL. PILUSUDSKIEGO 5	QUASHQAI	DW 64823	2013-03-23	2013-04-28	2021-02-28	231435	EDYTUJ	USUŃ
4	POZNAN UL. 3 MAJA 2	E90	DW 93880	2011-06-27	2011-07-11	2021-03-13	245959	EDYTUJ	USUŃ
✓ Pomyślnie zaktualizowano samochód		PRA	DW 85098	2020-06-20	2020-07-10	2021-07-16	570529	EDYTUJ	USUŃ

Rys. 5.24: Panel pracownika po modyfikacji samochodu.

	ID_SAMOCHODU	ID_WYPOZYCZALNI	ID_MODELU	NR_REJESTRACYJNY	DATA_PRODUKCJI	DATA_REJESTRACJI	DATA_PRZEGLEDU	PRZEBIEG
▶	2	5	BMW_E90	XX XXXX	2021-02-01	2021-02-02	2021-02-03	55555
	3	3	NIS_Q	DW 64823	2013-03-23	2013-04-28	2021-02-28	231435
	4	4	BMW_E90	DW 93880	2011-06-27	2011-07-11	2021-03-13	245959
	5	5	TOY_CPR	DW 85098	2020-06-20	2020-07-10	2021-07-16	570529

Rys. 5.25: Zawartość tabeli *SAMOCHODY* w bazie danych po modyfikacji samochodu.

## Dodanie nowego samochodu

Na rysunku 5.26 przedstawiono widok panelu pracownika przed dodaniem nowego samochodu. Rysunek 5.27 przedstawia zawartość tabeli *SAMOCODY* w bazie danych przed dodaniem nowego samochodu. Na rysunku 5.28 przedstawiono menu dodawania samochodu z wpisanymi atrybutami dodawanego samochodu. Po kliknięciu przycisku *DODAJ* informacje dotyczące nowego samochodu pojawiły się na liście w panelu pracownika (rys. 5.29). Nowy samochód został również dodany w bazie danych, co przedstawiono na rysunku 5.30.

27	WARSZAWA UL. KOSCIUSZKI 7	GOLF V	DW 56570	2016-08-13	2016-09-14	2021-01-14	110019	EDYTUJ	USUŃ
28	KRAKOW UL. PILUSUDSKIEGO 5	GOLF V	DW 98844	2013-07-19	2013-08-11	2021-03-24	217568	EDYTUJ	USUŃ
29	POZNAN UL. 3 MAJA 2	ASX	DW 57517	2011-04-28	2011-05-26	2021-04-18	237360	EDYTUJ	USUŃ
30	GDANSK UL. 11 LISTOPADA 1	SUPRA	DW 98351	2015-03-28	2015-04-28	2021-04-21	532039	EDYTUJ	USUŃ

Rys. 5.26: Panel pracownika przed dodaniem nowego samochodu.

26	1	TOY_SUP	DW 63359	2017-07-20	2017-08-10	2021-01-12	521873
27	2	VW_G5	DW 56570	2016-08-13	2016-09-14	2021-01-14	110019
28	3	VW_G5	DW 98844	2013-07-19	2013-08-11	2021-03-24	217568
29	4	MIT_ASX	DW 57517	2011-04-28	2011-05-26	2021-04-18	237360
30	5	TOY_SUP	DW 98351	2015-03-28	2015-04-28	2021-04-21	532039
		NULL	NULL	NULL	NULL	NULL	NULL

samochody 5 x

Rys. 5.27: Zawartość tabeli *SAMOCODY* w bazie danych przed dodaniem nowego samochodu.

Dodaj samochód

Wypożyczalnia \*  
WROCLAW, UL. JAGIELLY 11

Model samochodu \*  
NISSAN QUASHQAI

Numer rejestracyjny \*  
YY YYYY

Data produkcji \*  
01.02.2021

Data rejestracji \*  
02.02.2021

Data przeglądu \*  
03.02.2021

Przebieg \*  
11111

Rys. 5.28: Menu dodawania samochodu z wpisanymi atrybutami dodawanego samochodu.

28	KRAKOW UL. PIŁUSUDSKIEGO 5	GOLF V	DW 98844	2013-07-19	2013-08-11	2021-03-24	217568	EDYTUJ	USUŃ
29	POZNAN UL. 3 MAJA 2	ASX	DW 57517	2011-04-28	2011-05-26	2021-04-18	237360	EDYTUJ	USUŃ
30	GDANSK UL. 11 LISTOPADA 1	SUPRA	DW 98351	2015-03-28	2015-04-28	2021-04-21	532039	EDYTUJ	USUŃ
31	WROCLAW UL. JAGIELLY 11	QUASHQAI	YY YYYY	2021-02-01	2021-02-02	2021-02-03	11111	EDYTUJ	USUŃ

Rys. 5.29: Panel pracownika po dodaniu nowego samochodu.

27	2	VW_G5	DW 56570	2016-08-13	2016-09-14	2021-01-14	110019
28	3	VW_G5	DW 98844	2013-07-19	2013-08-11	2021-03-24	217568
29	4	MIT_ASX	DW 57517	2011-04-28	2011-05-26	2021-04-18	237360
30	5	TOY_SUP	DW 98351	2015-03-28	2015-04-28	2021-04-21	532039
31	1	NIS_Q	YY YYYY	2021-02-01	2021-02-02	2021-02-03	11111
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

samochody 6 x

Rys. 5.30: Zawartość tabeli *SAMOCODY* w bazie danych po dodaniu nowego samochodu.

## 5.3. Wnioski z przeprowadzonych testów

Przetestowane zostały najważniejsze funkcje systemu. W czasie przebiegu testów nie stwierdzono nieprawidłowości w działaniu systemu. Wszystkie przetestowane funkcje systemu działają zgodnie z oczekiwaniami.



## Rozdział 6

# Podsumowanie

System został zaimplementowany zgodnie z założeniami projektowymi. Umożliwia komunikację użytkownika z bazą danych z poziomu aplikacji z przyjaznym dla użytkownika interfejsem graficznym. System zapewnia również autoryzację użytkowników, dzięki czemu dane w bazie są chronione przed dostępem osób nieuprawnionych.

# Literatura

- [1] *Kod odpowiedzi HTTP*: [https://pl.wikipedia.org/wiki/Kod\\_odpowiedzi\\_HTTP](https://pl.wikipedia.org/wiki/Kod_odpowiedzi_HTTP).
- [2] *Material UI*: <https://mui.com/>.
- [3] *Strona internetowa*:  
<https://dev.mysql.com/doc/mysql-installation-excerpt/8.0/en/windows-install-archive.html>.
- [4] *Strona internetowa*: <https://nodejs.org/en/download/>.
- [5] *Strona internetowa*: <https://pl.reactjs.org/>.
- [6] *Strona internetowa*: <https://www.gov.pl/web/gov/czym-jest-numer-pesel>.
- [7] *Strona internetowa*: <https://www.mysql.com/products/workbench/>.
- [8] *Strona internetowa*: <https://www.oracle.com/pl/java/technologies/javase/jdk11-archive-downloads.html>.
- [9] B. Mehta. *REST. Najlepsze praktyki i wzorce w języku Java*.
- [10] B. Smith. *Beginning Json*. 2015.
- [11] L. Ullman. *MySQL. Szybki start*. 2007.