

POLITECHNIKA WROCŁAWSKA

WYDZIAŁ ELEKTRONIKI

## PROJEKT z BAZ DANYCH

**System zarządzania warsztatem samochodowym**

Termin zajęć: Środa, 9:15–11:00

AUTOR/AUTORZY:

Marcin Bober

Indeks: 249426

E-mail: [249426@student.pwr.edu.pl](mailto:249426@student.pwr.edu.pl)

Rafał Rzewucki

Indeks: 248926

E-mail: [248926@student.pwr.edu.pl](mailto:248926@student.pwr.edu.pl)

Wrocław 2020

PROWADZĄCY ZAJĘCIA:

dr inż. Roman Ptak, W4/K9

## Spis treści

1.	Wstęp .....	3
1.1	Cel projektu .....	3
1.2	Zakres projektu.....	3
2.	Analiza wymagań.....	3
2.1	Opis działania i schemat logiczny systemu.....	3
2.2	Wymagania funkcjonalne .....	3
2.3	Wymagania niefunkcjonalne .....	4
2.3.1	Wykorzystywane technologie i narzędzia .....	4
2.3.2	Wymagania dotyczące bazy danych.....	4
2.3.3	Wymagania dotyczące bezpieczeństwa systemu .....	5
2.4	Przyjęte założenia projektowe .....	5
3.	Projekt systemu .....	5
3.1	Projekt bazy danych .....	5
3.1.1	Analiza rzeczywistości i uproszczony model konceptualny.....	5
3.1.2	Model logiczny i normalizacja .....	5
3.1.3	Model fizyczny i ograniczenia integralności danych .....	7
3.1.4	Inne elementy schematu – mechanizmy przetwarzania danych .....	10
3.1.5	Projekt mechanizmów bezpieczeństwa na poziomie bazy danych.....	10
3.2	Projekt aplikacji użytkownika .....	12
3.2.1	Architektura aplikacji i diagramy projektowe .....	12
3.2.2	Interfejs graficzny i struktura menu .....	13
3.2.3	Projekt wybranych funkcji systemu.....	14
3.2.4	Metoda podłączania do bazy danych – integracja z bazą danych.....	14
3.2.5	Projekt zabezpieczeń na poziomie aplikacji .....	14
4.	Implementacja systemu baz danych .....	15
4.1	Tworzenie tabel i definiowanie ograniczeń.....	15
4.1.1	Tworzenie tabel .....	15
4.1.2	Tworzenie kluczy obcych – definiowanie ograniczeń (relacji).....	15
4.2	Implementacja mechanizmów przetwarzania danych .....	16
4.3	Implementacja uprawnień i innych zabezpieczeń.....	18
4.4	Testowanie bazy danych na przykładowych danych.....	18

# 1. Wstęp

## 1.1 Cel projektu

Warsztaty samochodowe często nie mogą sobie poradzić z zapisywaniem i zarządzaniem kolejką napraw pojazdów swoich klientów. Klienci warsztatów samochodowych chcieliby wiedzieć, kiedy ich pojazd będzie mógł zostać naprawiony. Celem projektu jest stworzenie systemu, który umożliwi zarządzanie kolejką napraw przez mechaników warsztatu, co będą mogli na bieżąco obserwować klienci warsztatu.

## 1.2 Zakres projektu

Serwis jest zaprojektowany dla dużego warsztatu samochodowego, aby zautomatyzować część czynności zarówno po stronie klientów warsztatu jak i po stronie mechaników. Został zaprojektowany jako baza do bardziej rozbudowanego serwisu. Będzie on przygotowany do tego, aby w przyszłości zostać rozbudowany o sklep z częściami, jak i o dodatkowy system powiadomień dla użytkowników o działaniach w serwisie.

Będzie on gromadził w bazie danych informacje na temat klientów warsztatu, naprawianych pojazdów oraz usterek, które zostały w nich naprawione. Co więcej będzie przechowywany rejestr mechaników oraz zapis wszystkich wydarzeń z systemu. Aby umożliwić korzystanie z systemu, zostanie stworzona aplikacja webowa, która w przejrzysty sposób ułatwi dostęp do danych.

# 2. Analiza wymagań

## 2.1 Opis działania i schemat logiczny systemu

Gdy klient wykryje usterkę w swoim pojeździe będzie mógł dodać nową naprawę bezpośrednio do kolejki napraw w warsztacie. Natychmiast po dodaniu nowej naprawy mechanik będzie mógł odpowiednio zmienić status usterki w systemie w zależności od postępu w jej usuwaniu. Klient przez cały czas ma wgląd do tego co aktualnie jest robione w jego sprawie i gdy mechanik zakończy naprawę jego pojazdu, klient będzie wiedział, kiedy może go odebrać.

## 2.2 Wymagania funkcjonalne

Możliwe będzie logowanie jako klient, mechanik lub jako administrator.

Obserwator ma możliwość do:

- przeglądania oferty warsztatu,
- dostęp do informacji kontaktowych.

Klient ma możliwość:

- dodawania swoich pojazdów do listy pojazdów,
- wprowadzania nowych aut do kolejki napraw,
- przeglądania stanu kolejki napraw,
- przeglądania statusu naprawy swojego pojazdu.

Mechanik ma uprawnienia do:

- organizacja wizyt w warsztacie,
- zarządzanie naprawą.

Administrator ma uprawnienia do:

- przypisywaniem ról do użytkowników,
- zarządzaniem rejestrem użytkowników oraz pojazdów.

## 2.3 Wymagania niefunkcjonalne

### 2.3.1 Wykorzystywane technologie i narzędzia

Aplikacja dostępowa zostanie zbudowana w języku wysokiego poziomu jakim jest PHP. Jest to prosty język wysokiego poziomu, który w prosty sposób pozwala zwizualizować dane dla użytkowników systemu oraz umożliwia interakcję z systemem. Dzięki niemu i wbudowanym w niego metodą możliwe jest połączenie z bazą danych oraz wykonywanie do niej zapytań.

### 2.3.2 Wymagania dotyczące bazy danych

Baza danych musi obsługiwać przechowywanie danych w formie tabel. Aplikacja będzie wymagać minimum 3 tabel o różnych rozmiarach, przy czym chcemy zachować możliwość rozbudowania aplikacji w przyszłości.

Szacunkowa liczba napraw w ciągu roku wynosi około 1000, przy czym musimy założyć, że każda naprawa będzie przeprowadzana na innym pojeździe. w najbardziej rozbudowanym przypadku każdy pojazd będzie miał innego właściciela, toteż zakładamy średnią ilość nowych użytkowników w ciągu roku na poziomie równym 1000. w sumie daje to około 3000 nowych rekordów w bazie danych w ciągu jednego roku. Dla każdego silnika baz danych jest to bardzo mała ilość, dlatego nie determinuje to wyboru silnika bazy.

Bardziej miarodajnym czynnikiem będzie ilość odczytów danych z bazy, ponieważ mechanicy w warsztacie prawdopodobnie będą cały czas zalogowani do systemu i co pewien okres czasu dane wyświetlane w serwisie będą musiały zostać zaktualizowane, tak aby zawsze można mieć dostęp do najnowszych danych.

### Częstość wykonywania operacji dla najważniejszych encji:

*Tabela 1 Analiza częstości wykonywania operacji na poszczególnych encjach*

	Wstawianie	Modyfikacja	Usuwanie	Wyszukiwanie
<b>Użytkownicy</b>	często	rzadko	b. rzadko	często
<b>Adresy</b>	często	rzadko	b. rzadko	często
<b>Kraje</b>	średnio	b. rzadko	b. rzadko	często
<b>Auta</b>	często	rzadko	b. rzadko	często
<b>Kolejka napraw</b>	często	często	b. rzadko	często
<b>Naprawy</b>	często	rzadko	b. rzadko	często
<b>Producenci Aut</b>	b. rzadko	b. rzadko	b. rzadko	często
<b>Modele aut</b>	b. rzadko	b. rzadko	b. rzadko	często
<b>Zapis działań w systemie</b>	b. często	b. rzadko	b. rzadko	często

Przy dodawaniu nowych danych do bazy będzie używany mechanizm transakcji, który umożliwi wycofanie wprowadzonych zmian, jeśli wystąpi jakikolwiek błąd przy dodawaniu. Jako serwer bazy danych został wybrany MariaDB wraz z mechanizmem składowania InnoDB z kilku istotnych powodów:

- jest on w pełni darmowy,
- jego uruchomienie nie wymaga praktycznie żadnej konfiguracji,
- nie ma wygórowanych wymagań sprzętowych,
- przechowuje dane również po zaniku zasilania,
- przez swoją prostotę budowy i działania operacje tak zapisu jak i odczytu z bazy będą wykonywane w bardzo krótkim czasie,
- posiada wbudowaną obsługę transakcji.

### 2.3.3 Wymagania dotyczące bezpieczeństwa systemu

Dla klientów zostanie zaprojektowany system logowania z użyciem adresów email i haseł. Hasła będą hashowane co zagwarantuje brak możliwości ich późniejszego odczytania nawet przez administratora serwisu.

Połączenia z serwerem będą szyfrowane co stanowi podstawę bezpieczeństwa przesyłania danych między urządzeniem klienta, a serwerem aplikacji.

## 2.4 Przyjęte założenia projektowe

- dostarczenie przystępnej aplikacji dla warsztatów i ich klientów,
- stworzenie stabilnej i odpornej na błędy aplikacji dostępowej do bazy danych,
- utworzenie aplikacji bazodanowej, która będzie gwarantować spójność oraz bezpieczeństwo danych.

## 3. Projekt systemu

### 3.1 Projekt bazy danych

#### 3.1.1 Analiza rzeczywistości i uproszczony model konceptualny

W rzeczywistości występują klienci warsztatu samochodowego, ich pojazdy oraz usterki tych pojazdów. Wymagany jest model, który pozwoli jak najefektywniej przechowywać dane o każdym przedmiocie.

Co więcej ważne jest również, aby mechanik, który rozpoczyna naprawę auta wiedział co wcześniej było zmieniane przy tym aucie, można powiedzieć, że pomocna dla niego będzie historia napraw danego auta.

Administrator serwisu musi mieć możliwość przejrzania wszystkich zdarzeń jakie zaszły w serwisie, aby być w stanie wykryć ewentualne błędy. Taka historia zdarzeń może być również pomocna, jeśli wystąpią jakiegokolwiek inne problemy, w których takie dane mogą być pomocne.

Odzwierciedlenie tej części rzeczywistości w postaci modelu bazy danych zostało przedstawione na modelu logicznym jak i fizycznym bazy danych.

#### 3.1.2 Model logiczny i normalizacja

Baza danych musi przechowywać podstawowe dane o użytkownikach serwisu, ich uprawnieniach do działań w serwisie, adresach zamieszkania oraz dane kontaktowe. Auta dodawane przez klientów powinny być przypisane do danego użytkownika oraz posiadać podstawowe parametry pojazdu.

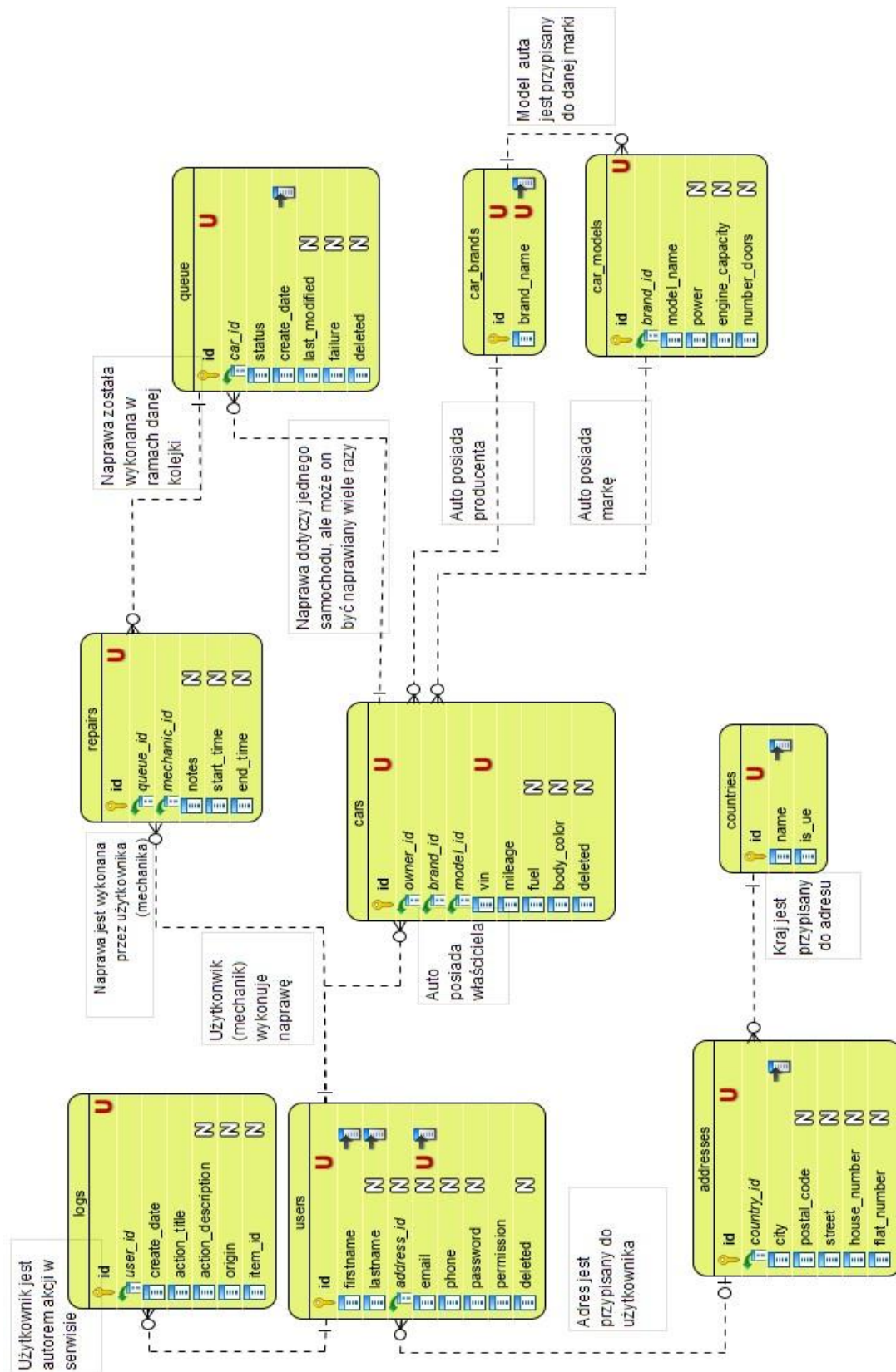
Kolejka napraw musi zawierać dane o tym jaki samochód był, jest lub będzie naprawiany, status naprawy, informację od klienta co się dzieje z autem, datę dodania do kolejki oraz dodatkowe informacje jak notatki od mechanika po zakończeniu naprawy.

Dane adresowe użytkowników mogą zostać zapisane w oddzielnej tabeli, aby można było w prosty sposób usystematyzować dane adresowe. Kraj w adresie może zostać znormalizowany w osobnej tabeli, która dodatkowo będzie zawierać informacje o tym czy dany kraj znajduje się w Unii Europejskiej. Taka informacja może być pomocna dla pracownika wystawiającego fakturę dla klienta jak i dla przyszłych funkcjonalności serwisu, które nie zostały uwzględnione w tym projekcie.

Również dane o producencie jak i modelu auta mogą zostać wyodrębnione i z racji na ich ograniczoną ilość mogą zostać usystematyzowane, aby były jednolite w całym serwisie.

### 3.1.3 Model fizyczny i ograniczenia integralności danych

Model konceptualny i logiczny bazy danych:



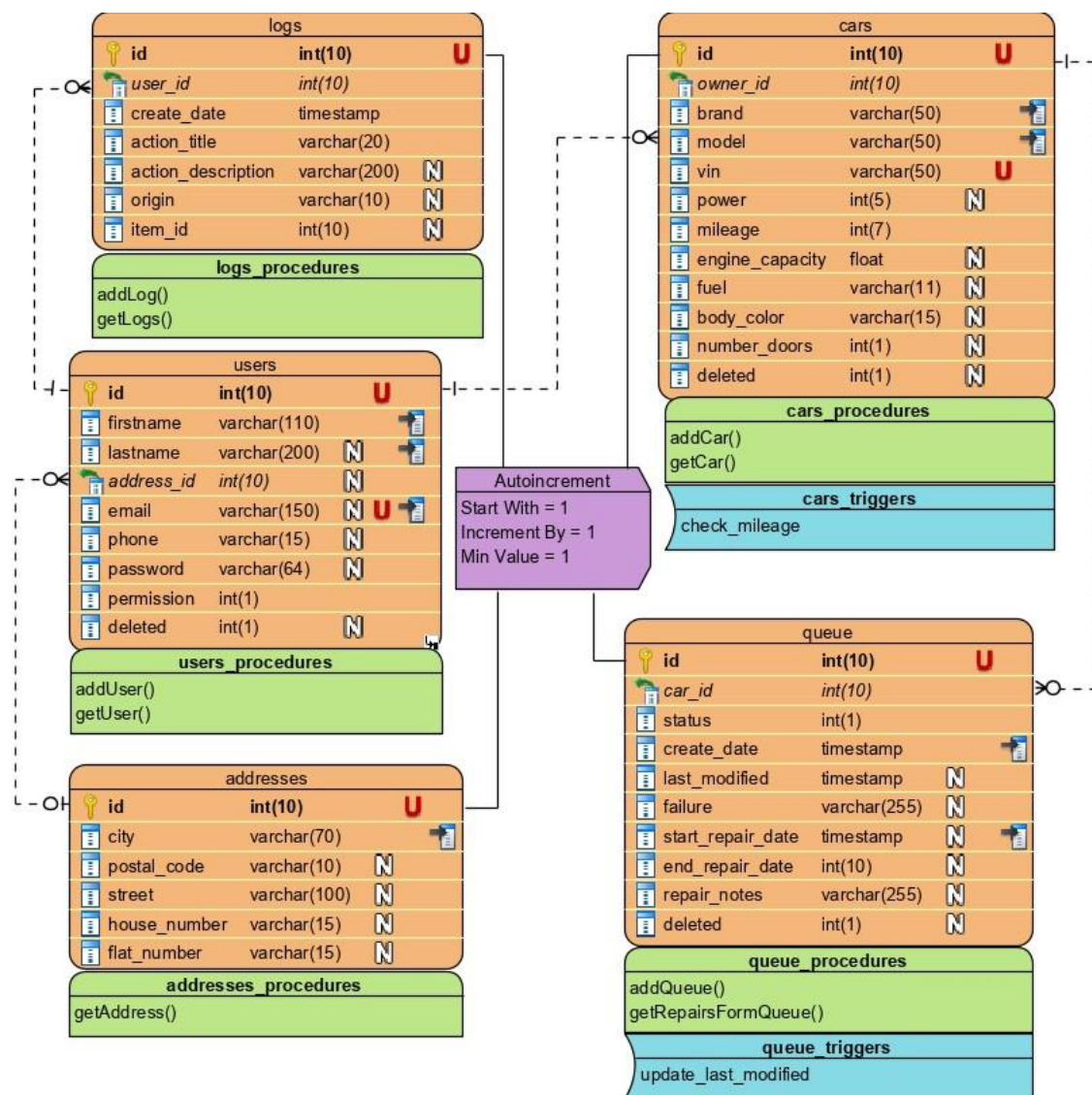
Rysunek 1 Model konceptualny bazy danych



## Relacje występujące w bazie danych:

- Relacja jeden do wielu między tabelą *users* a tabelą *cars* obrazuje fakt, że dany użytkownik serwisu jest posiadaczem danego pojazdu, jeden użytkownik może posiadać wiele pojazdów,
- Relacja jeden do wielu między *cars* a *queue* obrazuje sytuację, gdzie jeden samochód może być naprawiany wiele razy,
- Relacja jeden do wielu między *queue* a *repairs* pokazuje, która naprawa została wykonana w ramach danej zaplanowanej naprawy. Może być wiele napraw w ramach jednej kolejki,
- Relacja jeden do wielu między *users* a *repairs* zaznacza który mechanik wykonał daną naprawę,
- Relacja jeden do wielu między *car\_brands* oraz *car\_models* a *cars* pozwala, aby w bazie danych było wiele aut tego samego producenta, odpowiednio modeli,
- Relacja jeden do wielu między *addresses* a *users* daje możliwość, aby w systemie wiele użytkowników współdzieliło jeden adres.
- Relacja jeden do wielu między *countries* a *addresses* obrazuje sytuację, gdzie istnieje wiele adresów zamieszkania w jednym kraju.

## Model fizyczny bazy danych:



Rysunek 2 Model fizyczny bazy danych



#### Poczynione uproszczenia i uogólnienia:

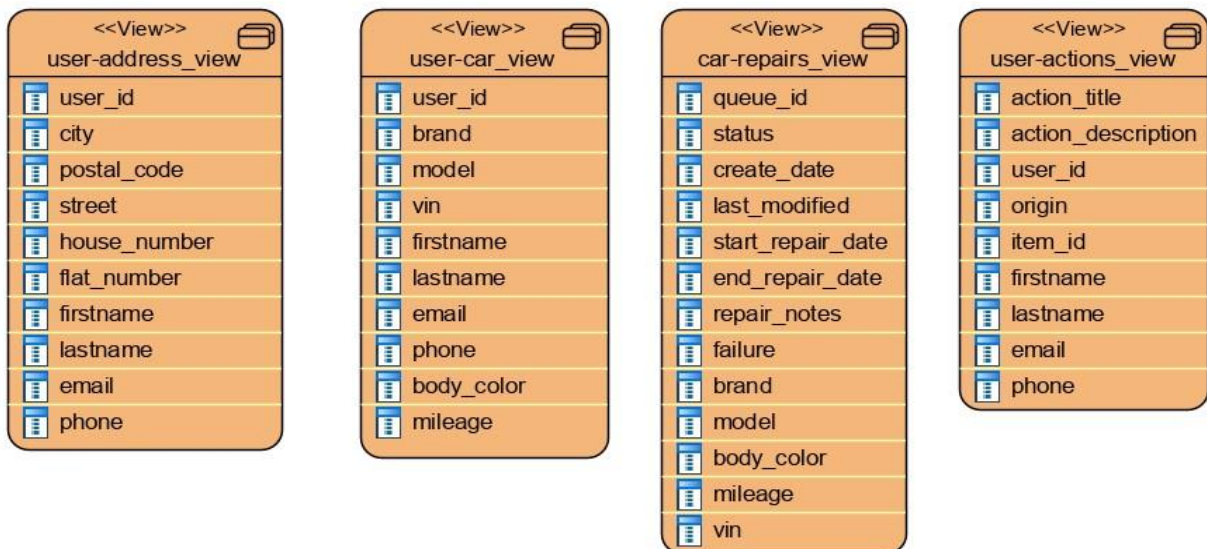
- Tabela *cars* mogłaby mieć jako klucz główny kolumnę *vin*, ponieważ jest on niesztucznym kluczem dla każdego rekordu jednak jest on bardzo długi, co mogłoby wpłynąć na wydajność, dlatego zdecydowano się na dodanie sztucznego klucza.
- Tabela zawierająca adresy użytkowników została scalona z tabelą użytkowników. Powodem takiego uproszczenia był fakt, że użytkownik w serwisie zawsze będzie miał podany jeden adres. Osobna tabela adresów byłaby uzasadniona w sytuacji gdy wiele użytkowników współdzieliło by jeden adres, jednak taka sytuacja występowała by szczególnie często tylko w przypadku dużych firm gdzie każdy pracownik firmy byłby osobnym użytkownikiem w serwisie, jednak w rzeczywistości takie firmy mają specjalne osoby oddelegowane do zajmowania się takimi sprawami, co powoduje, że w serwisie nawet dla dużych firm zarejestrowanych pod jednym adresem nie będzie to powodowało nadmiarowości danych w serwisie.
- Kraj z adresu został całkowicie usunięty. Powodem takiej decyzji było założenie, że system będzie obsługiwał klientów głównie z Polski. Nawet jeśli zdarzy się wyjątek użytkownika z poza granic Polski, to jest małe prawdopodobieństwo, że adres z innego państwa będzie miał swój odpowiednik w Polsce
- Producenci aut jak i modele aut zostały zaimplementowane bezpośrednio w tabeli *cars*. Jest to spowodowane faktem, że rozbicie to na osobne tabele wymagało by moderacji za każdym razem, gdy na rynku pojawi się nowy model pojazdu. Stwarzało by to dodatkowe obowiązki dla administratora systemu oraz możliwe okresowe problemy dla klientów. Takie rozwiązanie pozwoli klientom wpisać markę i model pojazdu bezpośrednio do bazy danych. Co więcej chcemy, aby użytkownicy mieli swobodę w dodawaniu swoich aut, które mogły ulec modyfikacją po opuszczeniu fabryki.
- Tabela z danymi poszczególnych napraw została również uproszczona do kilku istotnych pól w tabeli kolejki napraw. Dawała ona jedynie dwie dodatkowe możliwości, które w tym serwisie są zbędne. Można było z niej odczytać, który mechanik wykonywał naprawę, jednak taką informację można również uzyskać z zapisu zdarzeń w systemie. Można było mieć dwie naprawy w jednej kolejce napraw, jednak taka sytuacja występowała by niezwykle rzadko, ponadto można ją uprościć do tego, że notatka po naprawie w kolejce napraw będzie odpowiednio dłuższa.

#### Projekt procedur składowanych:

- `INT(10) addUser ( firstname VARCHAR(110), lastname VARCHAR(200), email VARCHAR(150), phone VARCHAR(15), password VARCHAR(64), city VARCHAR(70), postalcode VARCHAR(10), street VARCHAR(100), housenumber VARCHAR(15), flatnumber VARCHAR(15))`
- `ROW getUser(user_id INT(10))`
- `ROW getAddress(address_id INT(10))`
- `INT(10) addLog ( user_id INT(10), title VARCHAR(20), description VARCHAR(200), origin VARCHAR(10), item_id INT(10))`
- `TABLE getLogs (howmuch INT(10), start_from INT(10))`
- `INT(10) addCar ( owner_id INT(10), brand VARCHAR(50), model VARCHAR(50), vin VARCHAR(50), power VARCHAR(50), mileage INT(7), engine_capacity FLOAT(4), fuel VARCHAR(11), body_color VARCHAR(15), number_doors INT(1))`

- `INT(10) addCar ( owner_id INT(10), brand VARCHAR(50), model VARCHAR(50), vin VARCHAR(50), power VARCHAR(50), mileage INT(7), engine_capacity FLOAT(4), fuel VARCHAR(11), body_color VARCHAR(15), number_doors INT(1))`
- `ROW getCar(car_id INT(10))`
- `INT(10) addQueue ( car_id INT(10), failure VARCHAR(255))`
- `TABLE getRepairsFormQueue (howmuch INT(10), start_from INT(10))`

Projekt widoków w fizycznej bazie danych:



Rysunek 3 Projekt widoków bazy danych

- `user-address_view` – służy do wygodnego pobierania danych użytkownika wraz z danymi adresowymi,
- `user-car_view` – systematyzuje wybieranie danych aut danego użytkownika,
- `car-repairs_view` – pobieranie kolejki napraw wraz z danymi samochodu,
- `user-actions_view` – umożliwia proste pobieranie danych o działaniach użytkownika w systemie.

#### 3.1.4 Inne elementy schematu – mechanizmy przetwarzania danych

W tabeli *users* zostanie utworzony index na kolumnach *firstname*, *lastname* oraz *email* zostanie utworzony index, który usprawni wyszukiwanie użytkowników.

Indeks zostanie również ustanowiony w tabeli *queue* odpowiadającej kolejce napraw na kolumnie *create\_date*. Pomoże on sortować dane w zależności od daty dodania do kolejki.

Opis wyzwalaczy zaprojektowanych w fizycznym modelu bazy danych:

- `check update_last_modified` – uruchamiany po modyfikacji rekordu. Aktualizuje datę ostatniej modyfikacji rekordu w kolejce napraw.
- `check_mileage` - uruchamiany przed dodaniem i modyfikacją rekordu. Sprawdza, czy nowy przebieg auta będzie większy od poprzedniego.

#### 3.1.5 Projekt mechanizmów bezpieczeństwa na poziomie bazy danych

Zabezpieczenie przed nieuprawnionym dostępem będzie się odbywać poprzez zabezpieczenie bazy danych hasłem, które będzie przechowywane w formie zaszyfrowanej. Dostęp do hasła jak i

do bazy danych będzie miał administrator, właściciel bazy danych oraz osoby wskazane przez właściciela.

Zabezpieczenie przed utratą danych będzie zapewnione poprzez wykonywanie cyklicznych kopii zapasowych całej bazy danych. Po uruchomieniu usługi w warsztacie można również rozszerzyć bezpieczeństwo poprzez przechowywanie dodatkowej bazy danych w chmurze jako kopię tylko do odczytu, która również może umożliwić odzyskać dane w przypadku awarii serwera umieszczonego w warsztacie.

Aby serwis mógł współpracować z bazą danych wymagane jest utworzenie dwóch użytkowników. Użytkownik zwykły oraz użytkownik przeznaczony dla administratora, z rozszerzonymi uprawnieniami. Projekt uprawnień znajduje się poniżej.

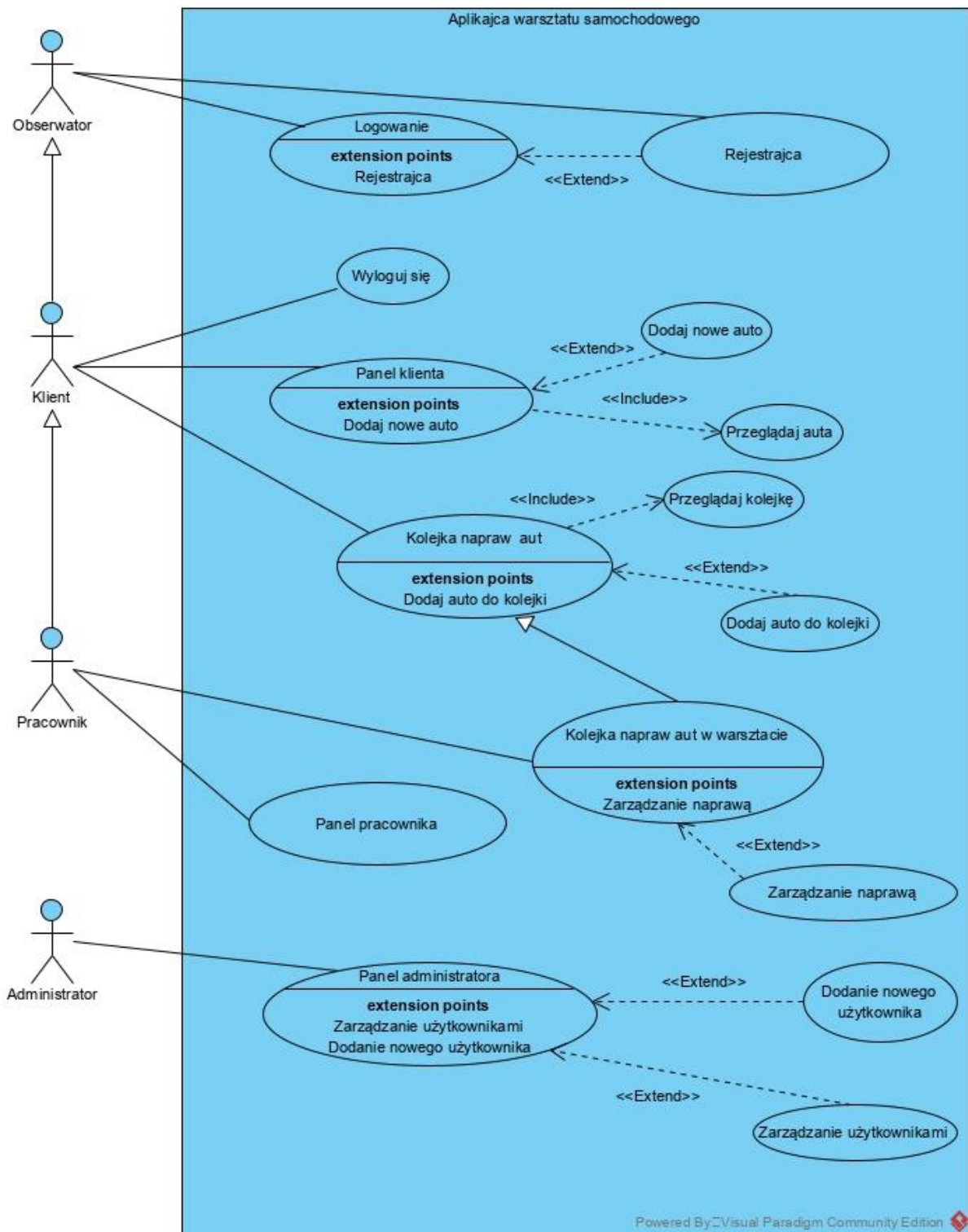
*Tabela 2 Poszczególne uprawnienia użytkowników*

	Wstawianie		Modyfikacja		Usuwanie		Wyszukiwanie	
Użytkownik	Zwykły	Admin	Zwykły	Admin	Zwykły	Admin	Zwykły	Admin
<b>users</b>	Tak	Tak	Tak	Tak	Nie	Tak	Tak	Tak
<b>addresses</b>	Tak	Tak	Tak	Tak	Nie	Tak	Tak	Tak
<b>cars</b>	Tak	Tak	Tak	Tak	Nie	Tak	Tak	Tak
<b>queue</b>	Tak	Tak	Tak	Tak	Nie	Tak	Tak	Tak
<b>logs</b>	Tak	Tak	Nie	Tak	Nie	Tak	Nie	Tak

## 3.2 Projekt aplikacji użytkownika

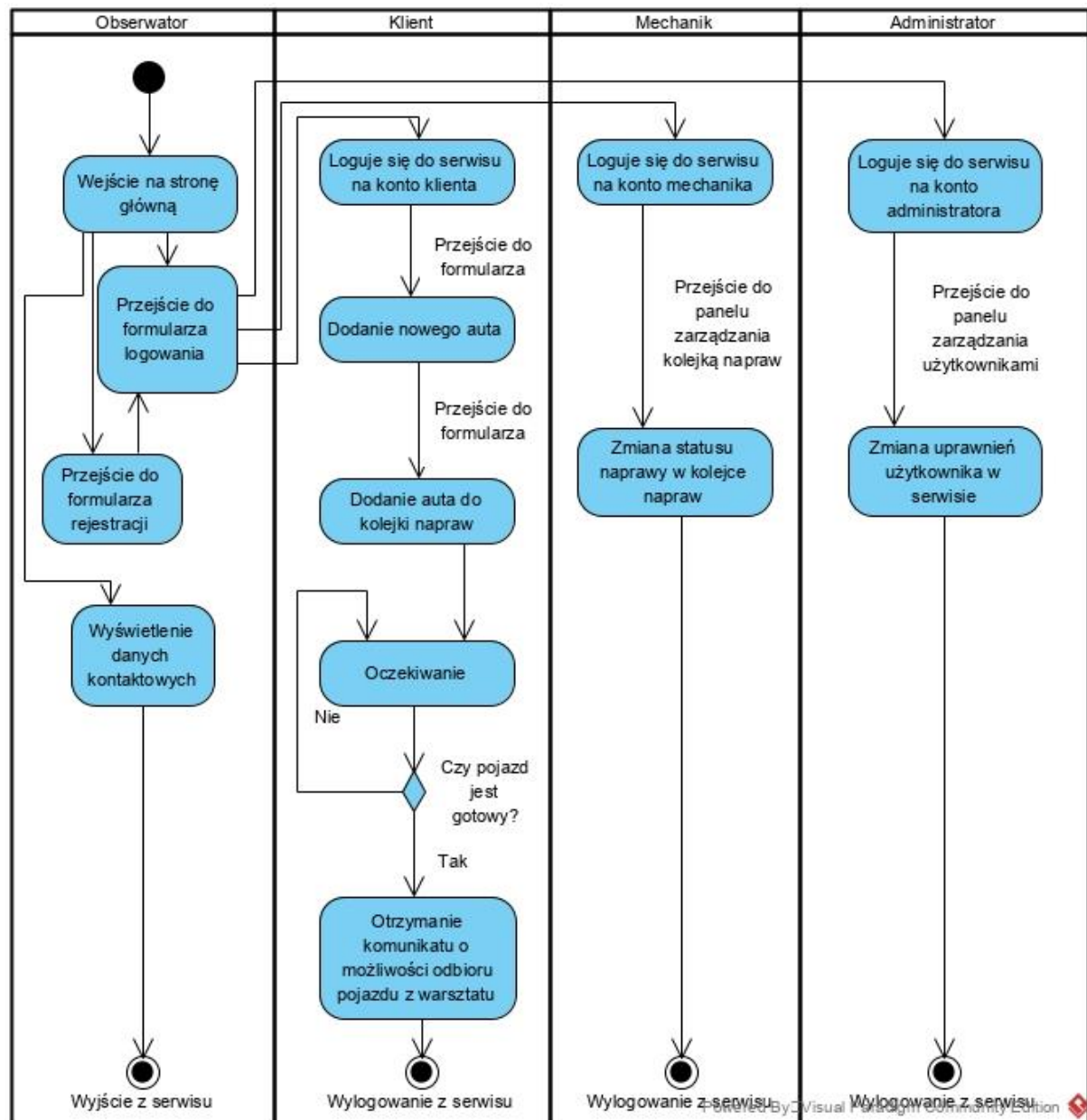
### 3.2.1 Architektura aplikacji i diagramy projektowe

#### Diagram przypadków użycia:



Rysunek 4 Diagram przypadków użycia

## Diagram czynności:



Rysunek 5 Diagram czynności

### 3.2.2 Interfejs graficzny i struktura menu

Jako że jesteśmy w posiadaniu prototypu aplikacji dostępowej, zdecydowaliśmy się nie umieszczać makiet.

Użytkownik ma możliwość przejścia w następujące miejsca w serwisie:

- Strona główna – wyświetla stronę główną serwisu,
- Kolejka napraw – wyświetla aktualną kolejkę napraw w warsztacie
- Panel Klienta – przechodzi do panelu klienta:
  - Dodaj auto – klient ma możliwość dodać nowe auto do serwisu,
  - Dodaj auto do kolejki – w tym miejscu klient może dodać swój pojazd do kolejki napraw.

- Kontakt – przenosi na stronę, na której znajdują się dane kontaktowe,
- Zaloguj – przenosi do formularza umożliwiającego zalogowanie,
- Wyloguj – wylogowanie użytkownika z serwisu.

### 3.2.3 Projekt wybranych funkcji systemu

Najważniejsze funkcjonalności serwisu w formie nagłówków metod, które je będą realizować:

- **pobierzUzytkownikow()** – zwraca dane wszystkich użytkowników zarejestrowanych w serwisie,
- **pobierzUzytkownika(email)** – zwraca dane użytkownika o podanym adresie e-mail
- **zarejestrujUzytkownika(email, haslo, imie, nazwisko)** – tworzy nowego użytkownika w serwisie,
- **zalogujUzytkownika(id\_uzytkownika)** – loguje użytkownika do serwisu
- **dodajAuto(id\_wlasciciela, marka, model, vin, przebieg)** – dodaje auto o podanych danych i przypisuje je do użytkownika,
- **pobierzKolejke()** – pobierz aktualną kolejkę napraw,
- **dodajDoKolejki(id\_auta, usterka)** – dodaje auto do kolejki napraw,
- **pobierzAuto(id\_auta)** – pobiera informacje o aucie,

Funkcjonalności istotne z punktu widzenia mechaników warsztatu:

- **zmienStatusNaprawy(id\_naprawy)** – zmienia status podanej naprawy odpowiednio na przyjęto do realizacji, w trakcie naprawy i do odbioru,
- **zakonczNaprawe(id\_naprawy)** – kończy realizację naprawy, mechanik ma możliwość wprowadzić informacje o tym co zostało zrobione podczas naprawy.

Funkcjonalności istotne z punktu widzenia administratora serwisu:

- **zmienUprawnieniaUzytkownika(id\_uzytkownika)** – zmienia uprawnienia użytkownika odpowiednio klient, mechanik, administrator.
- **pobierzZdarzeniaSerwisu()** – umożliwia pobranie wszystkich zdarzeń z serwisu, aby można było je wyświetlić administratorowi.

### 3.2.4 Metoda podłączania do bazy danych – integracja z bazą danych

Połączenie z bazą danych będzie odbywać się poprzez rozszerzenie języka PHP o nazwie PDO. Zapewnia ono powłokę, którą można użyć, aby komunikować się z bazą danych i wykonywać wszelkie zapytania. za pomocą tego rozszerzenia możliwa jest również pełna obsługa transakcji bazy danych.

### 3.2.5 Projekt zabezpieczeń na poziomie aplikacji

Każdy użytkownik będzie miał przypisany indywidualny zestaw adresu e-mail i hasła, dzięki którym możliwe będzie zalogowanie do serwisu. Dodatkowo aplikacja będzie monitorowała zmianę adresu IP zalogowanego użytkownika, aby wykryć przechwycenie sesji przez innego użytkownika

Program przed dodaniem nowego użytkownika do bazy, jego email zostanie sprawdzony funkcje „filter\_input” z filtrem „FILTER\_VALIDATE\_EMAIL”, co więcej hasło zostanie zaszyfrowane poprzez użycie wbudowanej w język PHP funkcji password\_hash(). Domyślnie używa ona algorytmu bcrypt, który gwarantuje optymalną złożoność szyfrowania.



## 4. Implementacja systemu baz danych

### 4.1 Tworzenie tabel i definiowanie ograniczeń

#### 4.1.1 Tworzenie tabel

Tabele zostały tworzone bezpośrednio z poziomu języka SQL. Jak już wyżej zostało to wyszczególnione wyżej jako serwer bazy danych został wybrany MariaDB. Domyślny silnik bazy danych nie został zmieniony, dlatego poniższe kody źródłowe działają dla silnika InnoDB. Kod, który umożliwił utworzenie tabeli użytkowników znajduje się poniżej:

```
CREATE TABLE users
(
    id            INT(10) NOT NULL auto_increment, -- klucz podstawowy
    firstname     VARCHAR(110) NOT NULL,
    lastname      VARCHAR(200),
    address_id    INT(10),
    email         VARCHAR(150),
    phone         VARCHAR(15),
    password      VARCHAR(64),
    permission    INT(1) NOT NULL,
    deleted       INT(1),
    -- tworzenie indeksów
    PRIMARY KEY (id),
    INDEX (firstname, lastname),
    UNIQUE INDEX (email)
);
```

Kod do utworzenia pozostałych tabel był analogiczny do powyższego, dlatego dla przejrzystości dokumentacji został on pominięty.

#### 4.1.2 Tworzenie kluczy obcych – definiowanie ograniczeń (relacji)

Definiowanie kluczy obcych związanych z tabelą użytkowników odbyło się za pomocą poniższych poleceń w języku SQL:

- tworzenie powiązania auto – właściciel auta

```
ALTER TABLE cars
ADD CONSTRAINT fkcars777498
FOREIGN KEY (owner_id) - owner_id jako id właściciela auta
REFERENCES users (id);
```

- tworzenie powiązania adres – właściciel adresu

```
ALTER TABLE users
ADD CONSTRAINT fkusers787940
FOREIGN KEY (address_id) - id adresu, który dodał użytkownik
REFERENCES addresses (id);
```

- tworzenie powiązania zdarzenie – autor zdarzenia w serwisie

```
ALTER TABLE logs
ADD CONSTRAINT fklogs223551
FOREIGN KEY (user_id) - autor zdarzenia w serwisie
REFERENCES users (id);
```



Pozostałe klucze obce tworzone były w sposób analogiczny, dlatego dla przejrzystości dokumentacji zostały one pominięte.

## 4.2 Implementacja mechanizmów przetwarzania danych

W projekcie bazy danych zostało uwzględnionych kilka mechanizmów przetwarzania danych. Między innymi wyzwalacze i procedury.

### 4.2.1 Tworzenie procedur

Procedura umożliwiająca dodawanie nowego użytkownika serwisu do bazy:

```
DELIMITER $$ CREATE
OR
replace PROCEDURE addUser ( firstname varchar(110), lastname varchar
(200), email varchar(150), phone varchar(15), password varchar(64),
city varchar(70), postalcode varchar(10), street varchar(100), house
number varchar(15), flatnumber varchar(15))
BEGIN
    start TRANSACTION; -- rozpoczynanie transakcji
INSERT INTO `addresses` -- dodawanie adresu
(
    `city`,
    `postal_code`,
    `street`,
    `house_number`,
    `flat_number`
)
VALUES
(
    city,
    postalcode,
    street,
    housenumber,
    flatnumber
); INSERT INTO `users` -- dodawanie użytkownika
(
    `firstname`,
    `lastname`,
    `address_id`,
    `email`,
    `phone`,
    `password`,
    `permission`
)
VALUES
(
    firstname,
    lastname,
    (
        SELECT last_insert_id() - id adresu
        FROM `addresses` limit 1),
    email,
```

```

        phone,
        password,
        0
    ); COMMIT; END
$$

```

Jest to jedna z najbardziej rozbudowanych procedur, ponieważ dodaje rekordy bezpośrednio do dwóch tabel. Tak zbudowana procedura wręcz wymaga użycia mechanizmu transakcji, ponieważ gdyby jedno z zapytań się nie powiodło, nie można dokończyć kolejnego. Mogłoby to spowodować niespójność danych oraz zagrozić stabilności aplikacji dostępowej podczas korzystania z tej bazy danych.

Procedura umożliwiająca pobranie wszystkich podstawowych informacji o użytkowniku:

```

DELIMITER $$ CREATE OR replace PROCEDURE getUser (user_id INT(10)) b
egin
SELECT *
FROM `users`
LEFT JOIN `addresses` -- razem z adresem
ON `addresses`.`id` = `users`.`address_id`
WHERE `users`.`id` = user_id ;END $$

```

Pozostałe procedury są bliźniaczo podobne do powyższych jednak dużo prostsze, dlatego nie umieszczam kodu SQL, który jest odpowiedzialny za ich tworzenie, aby nie zaciemniać podstawowych informacji o projekcie.

#### 4.2.2 Wyzwalacze

Zaimplementowane wyzwalacze pozwalają automatycznie aktualizować niektóre z ważnych informacji w bazie danych.

Wyzwalacz automatyzujący aktualizację daty ostatniej modyfikacji w tabeli kolejki napraw:

```

CREATE TRIGGER update_last_modified after UPDATE
ON `queue`
FOR EACH row
UPDATE `queue`
SET `last_modified` = Now(); -- aktualizacja daty

```

Wyzwalacz sprawdzający czy nowo wprowadzony przebieg jest większy od obecnego. Jest on konieczny, ponieważ polskie prawo zabrania cofania liczników. Jeśli taki błąd wystąpi, będzie można szybko zareagować na błędnie wprowadzoną wartość bądź też zweryfikować wskazania licznika

```

CREATE trigger check_mileage before UPDATE
on `cars` for each row begin if new.mileage < old.mileage THEN signa
l sqlstate '46000'
SET message_text = 'New mileage could not be lower than current o
ne';END if;END

```

#### 4.2.3 Widoki

Predefiniowane widoki tabel znacząco wpływają na szybkość dostępu do danych, dlatego zostały one również zaimplementowane w tym projekcie. Poniższy kod tworzy widok łączący tabelę użytkowników wraz z tabelą ich adresów. Ponieważ każdy widok tworzy się w podobny sposób pozostała część zaimplementowanych widoków została pominięta, aby uniknąć powtarzania kodu w dokumentacji.

```

CREATE OR REPLACE view `user-address_view`
AS

```

```

SELECT `users`.*,
       `city`,
       `postal_code`,
       `street`,
       `house_number`,
       `flat_number`
FROM   `users` -- tabela użytkowników
LEFT JOIN `addresses` -- razem z tabelą adresów
        ON `addresses`.`id` = `users`.`address_id`

```

#### 4.3 Implementacja uprawnień i innych zabezpieczeń

Aby w większym stopniu zabezpieczyć dane przed nieuprawnioną modyfikacją, utworzono w bazie danych dwóch użytkowników, którzy cechują się różnym poziomem uprawnień do tabel. Pełny obraz uprawnień został pokazany w Tabeli 2 Poszczególne uprawnienia użytkowników.

Poniżej znajduje się polecenie SQL umożliwiające utworzenie użytkownika dla administratora serwisu:

```

CREATE USER 'super_user'@'localhost'
identified BY 'super_user_password_124';

```

Nowo utworzony użytkownik nie ma jeszcze żadnych praw dostępu do bazy danych, dlatego kolejnym krokiem jest dodanie mu praw dostępu do poszczególnych tabel. Służy do tego polecenie:

```

GRANT SELECT, INSERT, UPDATE, DELETE -- tylko wymagane uprawnienia
ON `serwis_aco`.* TO 'super_user'@'localhost';

```

Gwiazdka w tym przypadku oznacza wszystkie tabele w wybranej bazie danych. Zwykły użytkownik tworzony jest w analogiczny sposób do powyższego. Modyfikacji znacząco ulegają uprawnienia do poszczególnych tabel np. aby dodać odpowiednie uprawnienia dla tabeli zdarzeń (*logs*) należy użyć polecenia, które upoważnia go jedynie do wstawiania nowych rekordów do tabeli:

```

GRANT INSERT -- tylko dodawanie
ON `serwis_aco`.`logs` TO 'normal_user'@'localhost';

```

Zabezpieczenie przed utratą danych, czyli między innymi okresowa kopia zapasowa zależy od funduszy jakie zostaną przeznaczone na realizację tego projektu. Jeśli będą one znaczące, będzie można wykupić płatną usługę przetrzymującą wykonane kopie zapasowe i dbającą o ich bezpieczeństwo. Uruchamianie takich kopii zależy w dużej mierze od wybranej usługi, dlatego opis implementacji kopii zapasowych nie został opisany w niniejszej dokumentacji.

#### 4.4 Testowanie bazy danych na przykładowych danych

Aby lepiej zobrazować przeprowadzone przykładowe testy posłużono się narzędziem do zarządzania bazą danych o nazwie *phpmyadmin*. Pozwala ono w sposób graficzny zaprezentować dane zawarte w tabelach jak i pokazać błędy, które zaszły podczas wykonywania zapytania.

##### 4.4.1 Testy wydajnościowe i pojemnościowe.

Z racji na bardzo mały przyrost danych w ciągu roku (około 3000 rekordów) testy pojemnościowe nie zostały wykonane. Taka ilość danych przy tak nierozbudowanej aplikacji nie sprawi najmniejszego problemu nawet po 30 latach użytkowania serwisu (około 100 000 rekordów).

Do analizy wydajności systemu bazy danych przyjęto zachowania innych aplikacji, których statystyki użytkowania są dostępne w Internecie. Z danych uśrednionych wynika, że czas podstawowych operacji na tabelach nie zmienia się znacząco dla działania aplikacji dla 100 000 rekordów. Na podstawie tych danych uznano, że przeprowadzanie dodatkowych testów wydajnościowych jest zbędne.

#### 4.4.2 Test dodawania nowego auta z wykorzystaniem procedury addCar().

✓ Pokazano wiersze 0 - 0 (1 ogółem, Wykonanie zapytania trwało 0.0038 sekund(y).)

```
CALL addCar ( 1, 'Daewoo', 'Matiz', '12345678901234567', '30', 270000, 1.2, 'benzyna+gas', 'różowy', 5 )
```

☐ Pokaż wszystko

Liczba wierszy: 

25

Filtrowanie wierszy: 

Przeszukaj tę tabelę

+ Opcje

LAST\_INSERT\_ID()

3

Rysunek 6 Poprawne wywołanie procedury dodawania auta

Jak pokazano na powyższym rysunku udało się pomyślnie dodać nowe auto do bazy danych. Można również wywołać błąd podając do procedury niewystarczającą ilość parametrów. Efekt takiego testu został pokazany na poniższym rysunku:

1 CALL addCar(1, 'Daewoo', 'Matiz', '12345678901234567', '30', 270000, 1.2, 'benzyna+gas')|

SELECT \*

SELECT

INSERT

UPDATE

DELETE

Wyczyść

Format

Uzyskaj automatycznie zapisane zapytanie

☐ Parametry wiązania

[ Separator ; ]

☐ Pokaż to zapytanie tutaj ponownie

☐ Zachowaj pole zapytania

☐ F

**Błąd**

Zapytanie SQL: [Kopiuj](#)

```
CALL addCar (  
1,  
'Daewoo',  
'Matiz',  
'12345678901234567',  
'30',  
270000
```

MySQL zwrócił komunikat: [?](#)

#1318 - Incorrect number of arguments for PROCEDURE serwis\_aco\_test.addCar; expected 10, got 8

Rysunek 7 Błąd, gdy procedura została wywołana z niewłaściwą ilością argumentów

Jak widać procedura poprawnie nie uruchomiła się, gdy nie dostarczono do niej wystarczającej ilości argumentów. Zabezpiecza to przed naruszeniem spójności danych w bazie.

#### 4.4.3 Test uruchomienia wyzwalacza zaraz po wykonaniu polecenia UPDATE na tabeli cars.

✓ Pokazano wiersze 0 - 0 (1 ogółem, Wykonanie zapytania trwało 0.0016 sekund(y).)

```
SELECT `id`, `mileage` FROM `cars` WHERE `id` = 3
```

☐ Pokaż wszystko | Liczba wierszy: 25 ▼ Filtrowanie wierszy: Prze:

+ Opcje

	id	mileage
<input type="checkbox"/> Edytuj <input type="checkbox"/> Kopiuj <input type="checkbox"/> Usuń	3	270000

Jak widać na powyższym rysunku przebieg dla auta o identyfikatorze równym 3 wynosi 270 000. Zaktualizujmy ten przebieg do wartości 300 000.

✓ Zmodyfikowanych rekordów: 1. (Wykonanie zapytania trwało 0.0078 sekund(y).)

```
UPDATE `cars` SET `mileage` = 300000 WHERE `id` = 3
```

✓ Pokazano wiersze 0 - 0 (1 ogółem, Wykonanie zapytania trwało 0.0016 sekund(y).)

```
SELECT `id`, `mileage` FROM `cars` WHERE `id` = 3
```

☐ Pokaż wszystko | Liczba wierszy: 25 ▼ Filtrowanie wierszy: Prze:

+ Opcje

	id	mileage
<input type="checkbox"/> Edytuj <input type="checkbox"/> Kopiuj <input type="checkbox"/> Usuń	3	300000

Rysunek 8 Zwiększanie wartości przebiegu auta

Przebieg został pomyślnie zaktualizowany do większej wartości niż poprzednia. Sprawdźmy teraz co się stanie, gdy będziemy próbowali wrócić do poprzedniej wartości.

Uruchom zapytanie SQL/zapytania w tabeli serwis\_aco\_test.cars:

```
1 UPDATE `cars` SET `mileage` = 270000 WHERE `id` = 3
```

SELECT \*
SELECT
INSERT
UPDATE
DELETE
Wyczyść
Format

Uzyskaj automatycznie zapisane zapytanie

☐ Parametry wiązania

[ Separator ; ]
☐ Pokaż to zapytanie tutaj ponownie
☐ Zachowaj pole zapytania
☐ Przywróć po zakończeniu

### Błąd

Zapytanie SQL: [Kopiuuj](#)

```
UPDATE `cars` SET `mileage` = 270000 WHERE `id` = 3
```

MySQL zwrócił komunikat:

```
#1644 - New mileage could not be lower than current one
```

Rysunek 9 Błąd podczas zmniejszania wartości przebiegu auta

Tak jak zostało to zaimplementowane w wyzwalaczu o nazwie *check\_mileage* nie można obniżyć przebiegu auta. Ta sytuacja obrazuje poprawne działanie powyższego wyzwalacza.

#### 4.4.4 Dodawanie rekordów które odnoszą się w jakikolwiek sposób do rekordów z innych tabel.

Spróbowano dodać nowe auto jednak jako właściciela podano identyfikator użytkownika, który nie istnieje.

```
1 CALL addCar (
2 4,
3 'Daewoo',
4 'Matiz',
5 '12345678901234568',
6 '30',
7 270000,
8 1.2,
9 'benzyna+gas',
10 'różowy',
11 5
12 )
```

SELECT \*
SELECT
INSERT
UPDATE
DELETE
Wyczyść
Format

Rysunek 10 Próba dodania auta, którego właściciel nie istnieje w bazie danych

Po wykonaniu powyższego polecenia otrzymano poniższy komunikat o błędzie:

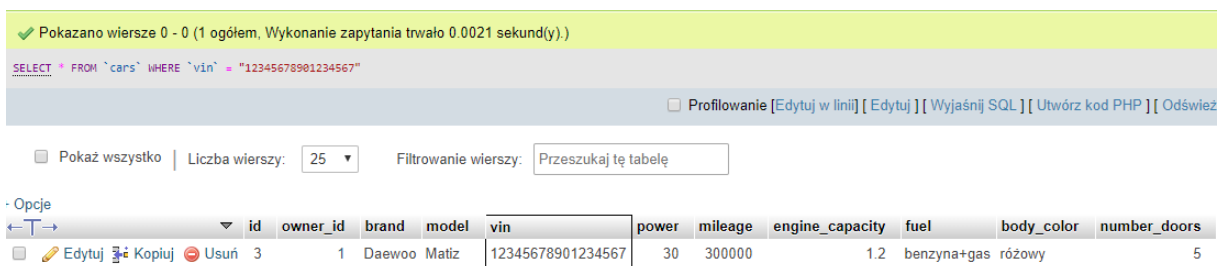


Rysunek 11 Komunikat o błędzie, gdy nie istnieje właściciel dodawanego auta

Taki komunikat oznacza, że relację gwarantującą spójność danych zostały zaimplementowane poprawnie.

#### 4.4.5 Test unikalnych indeksów w tabelach

W bazie danych istniał auto o numerze vin „12345678901234567”



Rysunek 12 Istniejące auto w bazie

Podczas próby dodania innego auta o takim samym numerze vin, otrzymano komunikat



Wykonanie zapytania/zapytań SQL do bazy danych serwis\_aco\_test: ?

1 CALL addCar ( 1,'Opel','Astra','12345678901234567','35',270000,1.2,'diesel','zielony',5)

Wyczyść

Format

Uzyskaj automatycznie zapisane zapytanie

☐ Parametry wiązania ?

[ Separator ; ]

☐ Pokaż to zapytanie tutaj ponownie

☐ Zachowaj pole zapytania

**Błąd**

Zapytanie SQL: [Kopiuj](#)

```
CALL addCar (  
1,  
'Opel',  
'Astra',  
'12345678901234567',  
'35',  
270000
```

**MySQL zwrócił komunikat:** ?

#1062 - Powtórzone wystąpienie '12345678901234567' dla klucza 'vin'

Rysunek 13 Błąd - ponowne wystąpienie wartości w tabeli

Ta sytuacja obrazuje poprawne działanie unikalnych indeksów. System nie pozwoli dodać dwóch aut o takich samych numerach vin.

#### 4.4.6 Test wartości NOT NULL

Jeśli przy tworzeniu tabeli nie została zadeklarowana domyślna wartość pola lub wartość NULL, serwer wymaga, aby przy dodawaniu nowego rekordu, wartość ta została podana. Jeśli nie zostanie podana, powinien zostać zwrócony błąd. Poprawne zachowanie pokazuje poniższy przykład, gdy przy dodawaniu nowego użytkownika nie zostaną podane jego uprawnienia.

Uruchom zapytanie SQL/zapytania w tabeli serwis\_aco\_test.users: ?

1 INSERT INTO `users`(`firstname`, `email`) VALUES ("Jan", "email@przyklad.com")

SELECT \* SELECT INSERT UPDATE DELETE Wyczyść

Uzyskaj automatycznie zapisane zapytanie

☐ Parametry wiązania ?

[ Separator ; ] ☐ Pokaż to zapytanie tutaj ponownie ☐ Zachowaj pole zap

**Błąd**

Zapytanie SQL: [Kopiuj](#)

INSERT INTO `users`(`firstname`, `email`) VALUES ("Jan", "email@przyklad.com")

**MySQL zwrócił komunikat:** ?

#1364 - Field 'permission' doesn't have a default value

Rysunek 14 Błąd, gdy nie zostaną podane wszystkie wymagane informacje

#### 4.4.7 Testowanie uprawnień użytkowników

Utworzono dwóch użytkowników, przy czym zwykły użytkownik ma ograniczone uprawnienia dla tabeli zdarzeń w systemie. Może on dodawać nowe rekordy, jednak nie może robić już nic więcej, nawet ich wyświetlać.

✓ Wstawionych rekordów: 1.  
Id wstawionego wiersza: 33 (Wykonanie zapytania trwało 0.0082 sekund(y).)

INSERT INTO `logs`(`user\_id`, `action\_title`) VALUES (1, 'login')

Rysunek 15 Dodawanie zdarzenia przez zwykłego użytkownika

Wykonanie zapytania/zapytań SQL do bazy danych serwis\_aco: ?

1 SELECT \* FROM `logs`

Wyczyść

Format

Uzyskaj automatycznie zapisane zapytanie

☐ Parametry wiązania ?

[ Separator ; ]

☐ Pokaż to zapytanie tutaj ponownie

☐ Zachowaj pole

**Błąd**

Zapytanie SQL: [Kopiuj](#) ?

```
SELECT * FROM `logs` LIMIT 0, 25
```

MySQL zwrócił komunikat: ?

```
#1142 - SELECT command denied to user 'normal_user'@'localhost' for table 'logs'
```

Rysunek 16 Błąd - brak uprawnień zwykłego użytkownika

Użytkownik administracyjny ma pełny dostęp do wszystkich tabel dlatego powyższy komunikat nie wystąpi.

## 5. Implementacja i testy aplikacji

### 5.1. Instalacja i konfigurowanie systemu

Instalacja aplikacji dostępowej na serwerze klienta jest uproszczona do minimum i sprowadza się do przetransferowania wszystkich uwczesnie przygotowanych przez nas plików na serwer dostępowy poprzez protokół FTP. Jedynym wymogiem który musi spełniać maszyna jest działający serwer WWW i odpowiednio skonfigurowany interpreter PHP.

```
define('DB_HOST', 'host.pl');
define('DB_USER', 'urzytkownik');
define('DB_PASS', 'haslo');
define('DB_NAME', 'nazwa bazy');
```

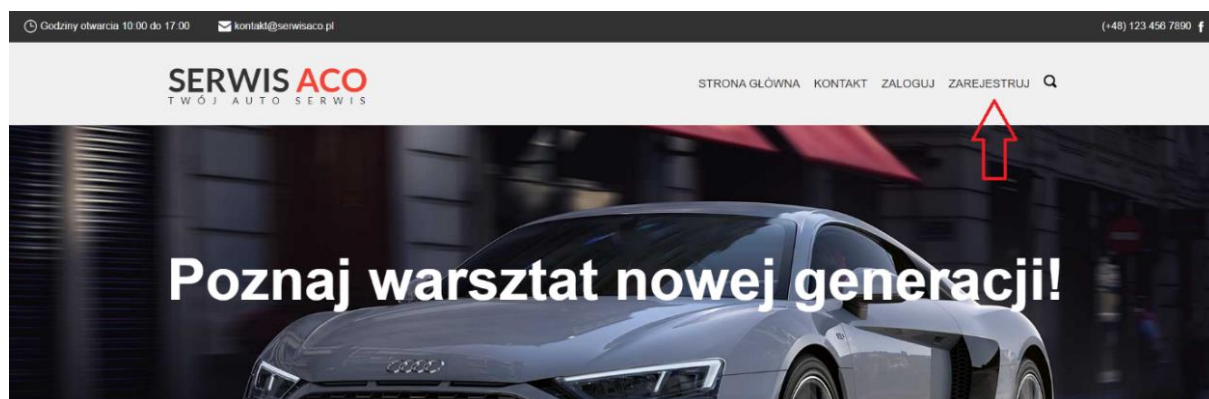
Rysunek 177 - Przykład konfiguracji serwera, plik . setput/db\_config.php

Po przetransferowaniu wszystkich dostarczonych plików należy wykonać prostą konfigurację. Należy edytować plik o ścieżce dostępowej „.setput/db\_config.php” i uzupełnić go o następujące

informacje dotyczące serwera bazy danych. Po poprawnym wprowadzeniu danych serwis powinien być gotowy do pracy.

## 5.2. Instrukcja użytkowania aplikacji

Po wejściu na stronę serwisu ukazuje nam się strona startowa. Nowy użytkownik może znaleźć informacje o godzinach pracy serwisu, służbowym adresie email oraz numerze telefonu do fizycznej placówki. Poniżej znajduje się menu główne dzięki któremu można w łatwy i przejrzysty sposób nawigować po całej witrynie firmy.



Rysunek 18 – Instrukcja rejestracji

Można tam znaleźć odnośnik do strony głównej, zakładkę z szczegółowymi danymi kontaktowymi, a także dwa odnośniki związane z rejestracją nowych kont i logowaniem na uprzednio utworzone konta. Użytkownik nieposiadający własnego konta po zapoznaniu z ofertą warsztatu przedstawioną w dolnej części strony powinien skierować się do zakładki „Zarejestruj”, w celu założenia nowego konta w serwisie. Dzięki niemu będzie w stanie łatwo i szybko stać się pełnoprawnym klientem serwisu.

**Zarejestruj się**

Imię lub nazwa firmy

Jan

**Dane kontaktowe**

Miasto

Wrocław

Nazwisko

Kowalski

Ulica

Norwida

Email

Email

Numer domu/mieszkania

15a

Hasło

Hasło

Kod pocztowy

00-000

Numer kontaktowy

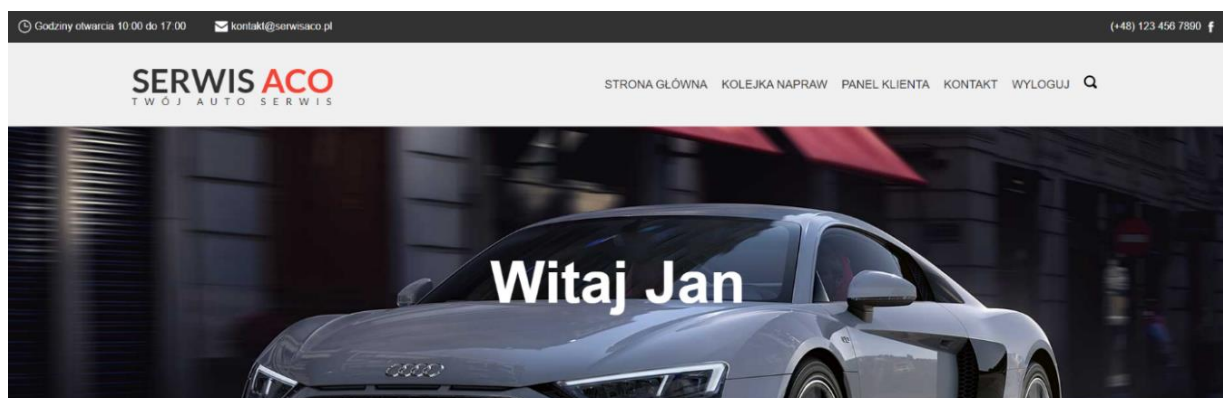
123456789

☐ Zgadzam się z warunkami zawartymi w [regulaminie](#)

**Rejestracja**

Rysunek 19 – Instrukcja rejestracji

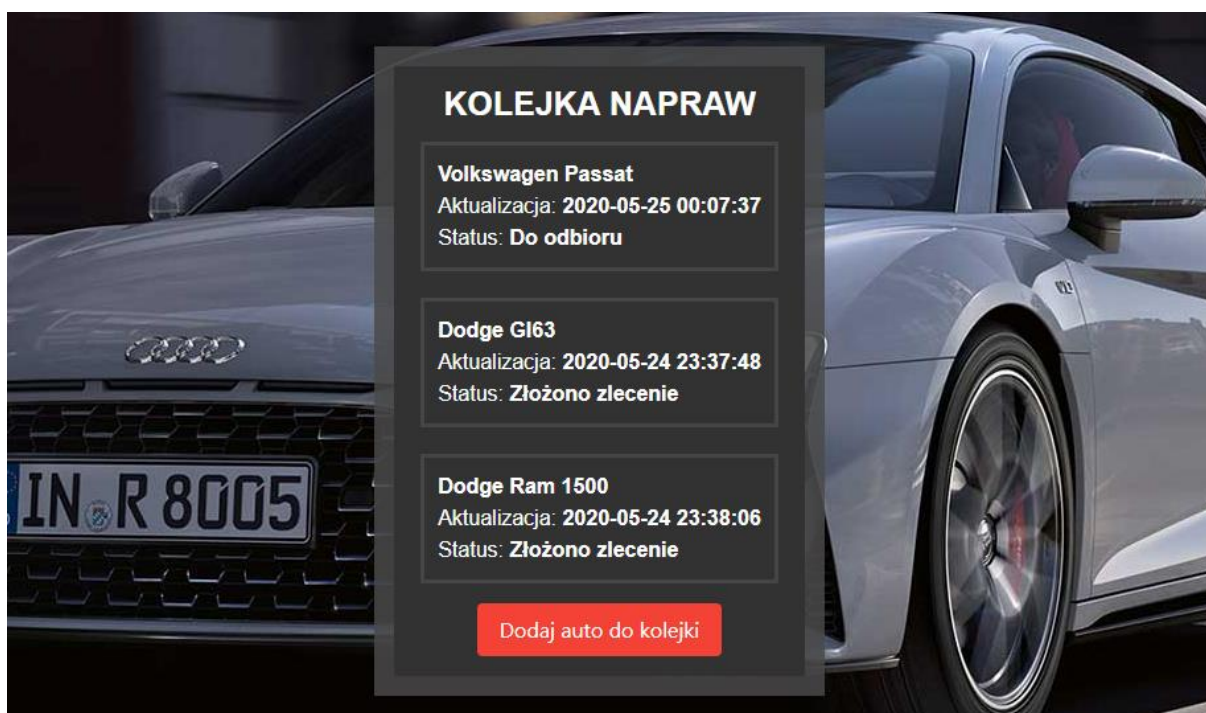
Po kliknięciu w odnośnik portal przekieruje nas do podstrony z formularzem rejestracyjnym. Formularz należy uzupełnić za pomocą poprawnych danych osobowych. Duży nacisk kładziony jest na poprawne wpisanie adresu email, bo to za jego pośrednictwem będzie przebiegać dalsza korespondencja z klientem. Jest on również niezbędny do późniejszego zalogowania się. Przed zatwierdzeniem swoich danych należy bezwzględnie zapoznać się z informacjami zawartymi w regulaminie serwisu. Dopiero po zaznaczeniu zgody na warunki regulaminu strona pozwala utworzyć nowe konto. Jeżeli wszystkie dane są poprawnie wprowadzone serwer automatycznie przekieruje nas na stronę główną. Po utworzeniu konta możemy skorzystać z funkcji logowania do systemu.



Rysunek 20 – Instrukcja logowania

Po poprawnym zalogowaniu baner z zakładkami ulegnie zmianie. Odnośniki do rejestracji i logowania zmieniają się w trzy nowe linki prowadzące do kolejki napraw, panelu klienta oraz funkcji zamykającej sesję z użytkownikiem. Poprawne zalogowanie możemy rozpoznać także poprzez zmianę napisu powitalnego w centralnej części witryny. Powinno ono zawierać teraz nasze imię zgodne z tym które podaliśmy podczas procesu zakładania nowego konta.

Po wybraniu odnośnika do kolejki napraw otrzymujemy wgląd w aktualnie przeprowadzane naprawy. Nie możemy się dowiedzieć kto jest właścicielem wystawianych pojazdów ani jakie są zdiagnozowane usterki, ale mamy możliwość podglądu statusu poszczególnych pozycji w kolejce i pełnej daty i godziny ostatniej zmiany tego statusu. Pozwala nam to śledzić postęp prac w zakładzie pracy oraz względnie oszacować czas wykonania usługi którą możemy zlecić firmie.



Rysunek 21 – Podgląd kolejki

Po wybraniu panelu klienta ukazały nam się dwie odrębne kolumny. Pierwsza z nich przedstawia pojazdy które są przypisane do naszego konta oraz szczegółowe dane na ich temat. Poniżej widnieje przycisk który przekieruje nas na podstronę umożliwiającą dodanie kolejnych pojazdów. Druga z wygenerowanych kolumn odpowiedzialna jest za wystawianie nam naszych ostatnich zgłoszeń do kolejki. Jak można zauważyć zawiera ona zdecydowanie więcej danych aniżeli było prezentowane w zakładce „Kolejka napraw”. Tutaj również znajduje się analogiczny przycisk który umożliwi nam przekierowanie do podstrony odpowiedzialnej za dodawanie nowych zgłoszeń napraw do kolejki.



## TWOJE AUTA

Marka: Volkswagen  
Model: Passat  
Nr VIN: KWDMDWODKMWDDD  
Moc: 290000KM  
Przebieg: 101km  
Pojemność silnika: 2137cm<sup>3</sup>  
Paliwo: Diesel  
Kolor: Biały  
Ilość drzwi: 5

Dodaj auto

## TWOJE ZGŁOSZENIA NAPRAWY

Pojazd: Volkswagen Passat  
Usterki: auto strasznie szarpie  
Status: Złożono zlecenie  
Ostatnia status z: 2020-05-24 23:35:44  
Rozpoczęto naprawę:

Dodaj auto do kolejki

Rysunek 22 – Panel klienta

W przypadku konta oznaczonego jako Pracownik zakłada ta zmienia swoją nazwę na „Panel Pracownika”. Z jej poziomu mamy możliwość zarządzaniem stanem kolejki. Dla każdego aktywnego zlecenia mamy cztery przyciski umożliwiające nam szybką zmianę statusu zlecenia. Aktualny status zlecenia jest podkreślony kolorem czerwonym. Jest to szybkie i bardzo wygodne narzędzie dla osób zajmujących się obsługą kolejki napraw w serwisie.

### KOLEJKA NAPRAW

Pojazd: Volkswagen Passat  
Nr. VIN: KWDMDWODKMWDDD  
Usterki: auto strasznie szarpie  
Status: Do odbioru  
Ostatnia status z: 2020-05-25 00:07:37  

Status "przyjęto"
Status "w naprawie"
Status "do odbioru"
Status "zakończono"

Pojazd: Dodge Gl63  
Nr. VIN: VF7DCXFXF76026643  
Usterki: podczas jazdy ściąga w prawo  
Status: Złożono zlecenie  
Ostatnia status z: 2020-05-24 23:37:48  

Status "przyjęto"
Status "w naprawie"
Status "do odbioru"
Status "zakończono"

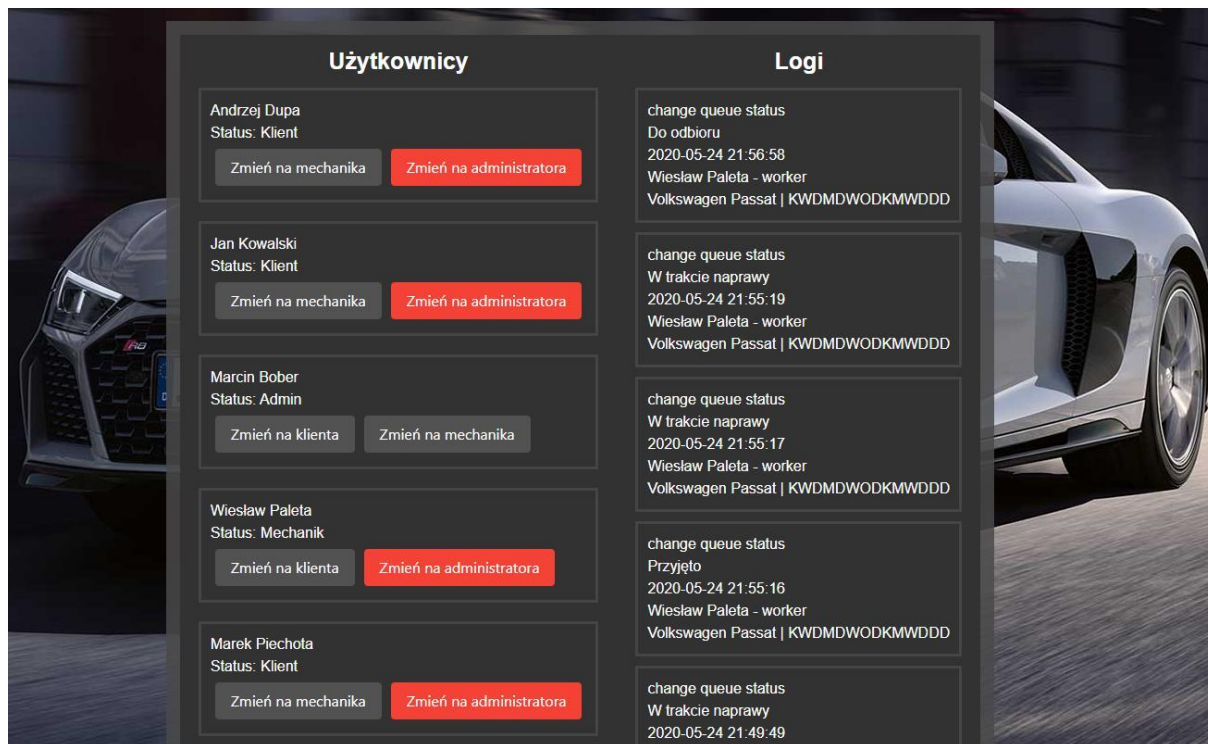
Pojazd: Dodge Ram 1500  
Nr. VIN: NUNJNLJNJNJDNAS  
Usterki: pije olej  
Status: Złożono zlecenie  
Ostatnia status z: 2020-05-24 23:38:06  

Status "przyjęto"
Status "w naprawie"
Status "do odbioru"
Status "zakończono"

Rysunek 23 – Panel pracownika



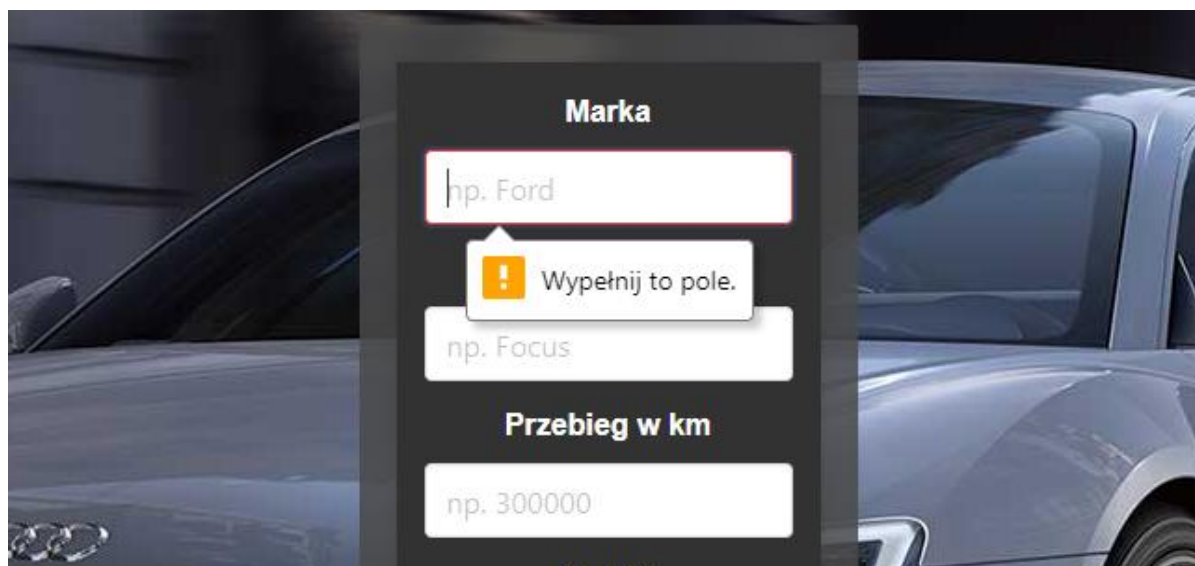
Jeżeli natomiast nasze konto będzie posiadać uprawnienia administratorskie zakładka ta przyjmie postać „Panelu administratora”. Znajdziemy w niej wszystkie niezbędne narzędzia jakie powinien posiadać administrator systemu. Pierwszym z nich jest tablica z użytkownikami w której możemy zmieniać uprawnienia danych kont z łatwy sposób. Drugie narzędzie służy do wyświetlania logów zanotowanych zdarzeń w systemie.



Rysunek 24 – Panel administratora

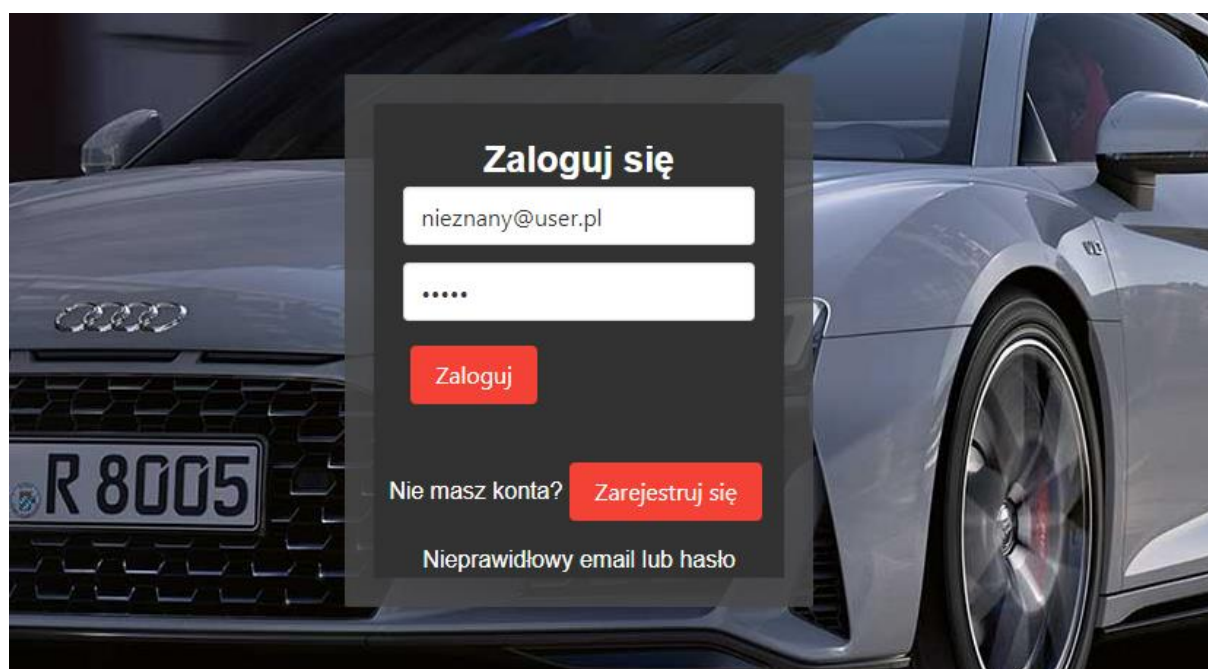
### 5.3. Testowanie opracowanych funkcji systemu

Wszystkie formularze umieszczone w serwisie podlegają walidacji danych zanim zostaną one wysłane do bazy danych. Dzięki temu możemy utrzymywać porządek w bazie oraz na bieżąco informować użytkowników o błędnie wprowadzanych danych bez znaczonego uszczerbku na wydajności całego systemu. Kluczowe informacje takie jak adres email czy inne wartości liczbowe są sprawdzane na poziomie HTML jak i interpretera PHP zanim trafią do bazy. Jeżeli nie wypełnimy jakiegoś pola oznaczonego jako wymagane to pojawi nam się taki komunikat.



Rysunek 25 – Źle wypełniony formularz rejestracji

W celu ujednolicenia niektóre dane mają przeprowadzone odpowiednie przekształcenia dzięki którym nazwy zawsze zaczynają się z wielkiej litery, a reszta znaków jest małymi literami. To sprawia że estetyka danych znacząco wzrosła.



Rysunek 26 – Wpisanie niewłaściwego hasła

Formularz logowania pobiera z bazy jedynie hasła przypisane do emaila podanego przez użytkownika. Jeżeli konto o takim emailu nie zostanie odnalezione lub hasła haseł nie są sobie równe to system wyświetli komunikat „Nieprawidłowy email lub hasło”.

**Zarejestruj się**  
Imię lub nazwa firmy: Adam

**Dane kontaktowe**  
Miasto: Kraków

Nazwisko: Polańczyk

Ulica: Wiejska

Email: adampol@gmail.com

Numer domu/mieszkania: 94a

Hasło: .....

Kod pocztowy: 12345

Numer kontaktowy: 987654321

☒ Zgadzam się z warunkami zawartymi w [regulaminie](#)

**Rejestracja**

Rysunek 27 – Wpisanie nowego klienta do bazy

Show query box

Showing rows 0 - 0 (1 total, Query took 0.0016 seconds.)

```
SELECT * FROM `users` ORDER BY ID DESC LIMIT 1
```

+ Options

	id	firstname	lastname	address_id	email	phone	password	permission	deleted
<input type="checkbox"/>	17	Adam	Polańczyk	17	adampol@gmail.com	987654321	\$2y\$10\$6XUwI06lbKDMAqQuJU0cGuunyQRzxXj6hMAihg29Buo...	0	0

↑ ☐ Check all With selected: ☐ Edit ☐ Copy ☐ Delete ☐ Export

Query results operations

☐ Print ☐ Copy to clipboard ☐ Export ☐ Display chart ☐ Create view

Rysunek 28 – Widok tabeli „users”

Showing rows 0 - 0 (1 total, Query took 0.0009 seconds.)

```
SELECT * FROM `addresses` ORDER BY ID DESC LIMIT 1
```

+ Options

	id	city	postal_code	street	house_number	flat_number
<input type="checkbox"/>	17	Kraków	12345	Wiejska	NULL	94a

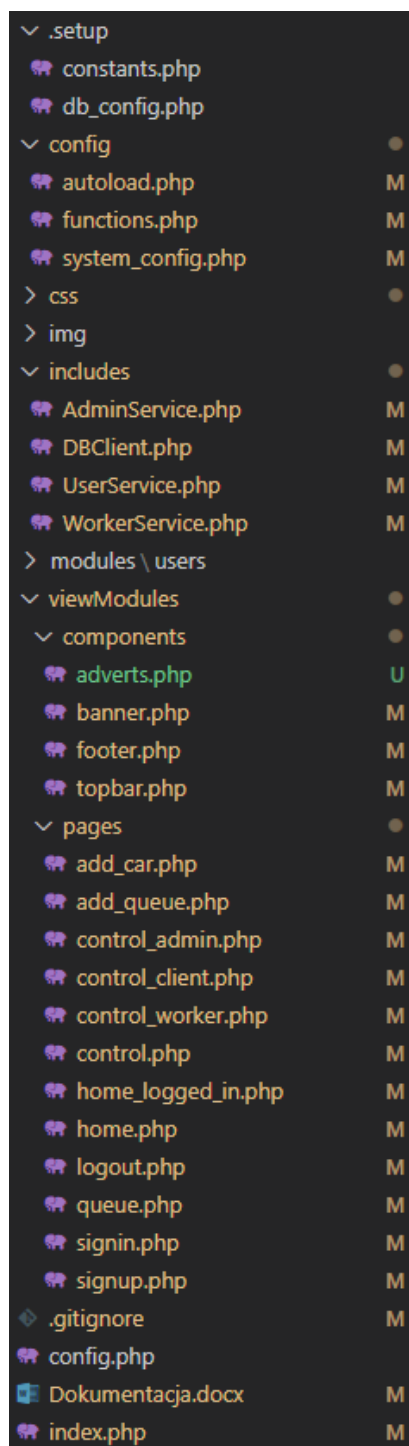
↑ ☐ Check all With selected: ☐ Edit ☐ Copy ☐ Delete ☐ Export

Rysunek 29 – Widok tabeli „addresses”

Na rysunku nr. 27 można zaobserwować proces poprawnego dodawania nowego użytkownika do bazy. Na kolejnym rysunku nr. 28 widzimy efekt dodania nowego użytkownika ze strony bazy danych. Rysunek nr.28 podobnie jak poprzednik pokazuje efekt dodania użytkownika ale tym razem pokazany jest rekord z tabeli z adresami.

#### 5.4. Omówienie wybranych rozwiązań programistycznych

Drzewo najważniejszych plików:



Rysunek 30 – Drzewo plików

#### 5.4.1. Implementacja interfejsu dostępu do bazy danych

Połączenie z bazą danych realizowane jest poprzez rozszerzenie klasy PDO. Jest ono maksymalnie zautomatyzowane dzięki nowemu konstruktorowi klasy DBClient. Dzięki temu zawsze wiemy że istnieje tylko jedno połączenie na klienta i nie musimy się martwić o otwieranie go i zamykanie w odpowiednich miejscach.

```
class DBClient extends PDO {  
  
    public $conn = null;  
    protected $db_host = DB_HOST;  
    protected $db_user = DB_USER;  
    protected $db_password = DB_PASS;  
    protected $db_name = DB_NAME;  
  
    public function __construct() {  
        $this->conn = parent::__construct('mysql:host=' . $this->db_host . ';dbname=' . $this->db_name, $this->db_user, $this->db_password, [  
            PDO::ATTR_EMULATE_PREPARES => false,  
            PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,  
            PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8"  
        ]);  
    }  
  
    public function __destruct() {  
        $this->conn = null;  
    }  
  
    public function connection() {  
        return $this->conn;  
    }  
  
    public function addLog($user, $action_title, $action_description, $origin, $item_id) {  
        $stmt = $this->prepare("INSERT INTO `logs` (`id`, `user_id`, `action_title`, `action_description`, `origin`, `item_id`) VALUES (NULL,  
        $stmt->bindValue(':user', $user, PDO::PARAM_STR);  
        $stmt->bindValue(':action_title', $action_title, PDO::PARAM_STR);  
        $stmt->bindValue(':action_description', $action_description, PDO::PARAM_STR);  
        $stmt->bindValue(':origin', $origin, PDO::PARAM_STR);  
        $stmt->bindValue(':item_id', $item_id, PDO::PARAM_STR);  
        return $stmt->execute();  
    }  
  
    public function getUser($id) {  
        $stmt = $this->prepare("SELECT * FROM `users` WHERE `id` = ?;");  
        $stmt->bindParam(1, $id, PDO::PARAM_STR);  
        $stmt->execute();  
        return $stmt->fetch();  
    }  
}
```

Rysunek 31 – Podgląd pliku includes/DBClient.php

- `__construct()` – konstruktor dla klasy. Przyjmuje wszystkie parametry niezbędne do prawidłowego połączenia z bazą danych.
- `__destruct()` – destruktor dla klasy,
- `connection()` – łączenie z bazą,
- `addLog()` – zwraca informacje dla administratora ostatnich czynnościach podejmowanych przez mechaników i klientów,
- `getUser()` – zwraca informacje o użytkowniku którego numer identyfikacyjny podany jest jako argument

### 5.4.2. Implementacja wybranych funkcjonalności systemu

Cały kod dla interpretera PHP został napisany w sposób obiektowy. Dzięki temu jest on przejrzysty i bardzo łatwo rozszerzalny o kolejne metody oraz funkcje. Sprawia to że tworzenie nowych podstron jest łatwe, szybkie i nie generuje dużo kodu ponieważ nie w kopiujemy w kółko tego samego kodu., bo same metody są wykorzystywane wielokrotnie w różnych podstronach serwisu. Zmniejsza to również koszty utrzymania kodu.

```
public function getCarByOwner($ownerid) {
    $stmt = $this->database->prepare("SELECT * FROM `cars` WHERE `owner_id` = ?;");
    $stmt->bindParam(1, $ownerid, PDO::PARAM_STR);
    $stmt->execute();
    return $stmt->fetchAll();
}

public function checkMyQueue($ownerid) {
    $carIds = array_column($this->getCarByOwner($ownerid), 'id');
    $in = join(',', array_fill(0, count($carIds), '?'));
    if (!$in)
        return [];
    $stmt = $this->database->prepare('SELECT * FROM `queue` WHERE car_id IN (' . $in . ')');
    $stmt->execute($carIds);
    return $stmt->fetchAll();
}
```

Rysunek 32 – Podgląd pliku includes/UserService.php

Mamy tutaj przykład dwóch wybranych funkcji. Pierwsza z nich pobiera z bazy danych informacje na temat pojazdu którego numer identyfikacyjny podawany jest poprzez argument tej metody. Druga funkcja w analogiczny sposób pobiera z bazy wszystkie pojazdy znając jedynie numer identyfikacyjny ich właściciela.

```
public function addAddress($city, $code, $street, $flat_number) {
    $stmt = $this->database->prepare("INSERT INTO `addresses` (`id`, `city`, `postal_code`, `street`, `flat_number`) VALUES (NULL, :city, :code, :street, :flat_number);");
    $stmt->bindValue(':city', $city, PDO::PARAM_STR);
    $stmt->bindValue(':code', $code, PDO::PARAM_STR);
    $stmt->bindValue(':street', $street, PDO::PARAM_STR);
    $stmt->bindValue(':flat_number', $flat_number, PDO::PARAM_STR);
    $stmt->execute();

    $stmt2 = $this->database->prepare("SELECT MAX(id) FROM `addresses`");
    $stmt2->execute();
    return $stmt2->fetch();
}

public function addCar($ownerid, $brand, $model, $vin, $mileage, $power_car, $engine_capacity, $fuel, $body_color, $number_doors) {
    $stmt = $this->database->prepare("INSERT INTO `cars` (`id`, `owner_id`, `brand`, `model`, `vin`, `power`, `mileage`, `engine_capacity`, `fuel`, `body_color`, `number_doors`)");
    $stmt->bindValue(':ownerid', $ownerid, PDO::PARAM_STR);
    $stmt->bindValue(':brand', $brand, PDO::PARAM_STR);
    $stmt->bindValue(':model', $model, PDO::PARAM_STR);
    $stmt->bindValue(':vin', $vin, PDO::PARAM_STR);
    $stmt->bindValue(':mileage', $mileage, PDO::PARAM_STR);
    $stmt->bindValue(':power_car', $power_car, PDO::PARAM_STR);
    $stmt->bindValue(':engine_capacity', $engine_capacity, PDO::PARAM_STR);
    $stmt->bindValue(':fuel', $fuel, PDO::PARAM_STR);
    $stmt->bindValue(':body_color', $body_color, PDO::PARAM_STR);
    $stmt->bindValue(':number_doors', $number_doors, PDO::PARAM_STR);
    return $stmt->execute();
}
```

Rysunek 33 – Podgląd pliku includes/UserService.php

Fragment kodu pokazany na powyższym zdjęciu przedstawia dwie funkcje. Pierwsza z nich ma za zadanie dodawać adresy nowo zarejestrowanych użytkowników do bazy i zwracanie numeru



identyfikacyjnego ostatniego wpisu. Druga z funkcji ma bardzo analogiczne zadanie, wprowadza ona nowe samochody do tabeli „cars”, jednakże ta metoda zwraca już tylko status powodzenia tej operacji.

### 5.4.3. Implementacja mechanizmów bezpieczeństwa

Aplikacja została przygotowana zgodnie z wszystkim wytycznymi i dobrymi nawykami programistycznymi. Co więcej, posiada ona zaimplementowane mechanizmy zabezpieczające przed atakami hackerskimi typu SQL injection. Połączenie z bazą danych jest szyfrowane co sprawia że nawet przejęcie ruchu pomiędzy serwerem z aplikacją dostępową, a serwerem bazy danych nie umożliwi przechwycenia ważnych informacji przez osoby niepożądane. Rdzeń aplikacji jest podzielony na moduły i łączy się z bazą danych poprzez dwa osobne konta z czego tylko jedno ma prawa dostępu i edycji do wszystkich tabeli. Sprawia to że uzyskanie hasła zabezpieczającego do jednego z kont nie musi kończyć się całkowitym wyciekiem bazy. Hasła dostępowe wszystkich użytkowników serwisu są haszowane po stronie aplikacji dostępowej zanim zostaną wysłane do bazy danych. Podczas logowania do systemu porównywane są jedynie hasze haseł dzięki czemu po wycieku danych przestępcy będą mieli znacznie utrudnione zadanie z dekodowaniem pierwotnej postaci hasła.

```
17 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
18     if (!empty($_POST['email']) && !empty($_POST['password']) && !empty($_POST['firstname'])) {
19         try {
20             $email = strtolower(filter_input(INPUT_POST, 'email', FILTER_VALIDATE_EMAIL));
21             if (!empty($email)) {
22                 $user = $service->getUser($email);
23                 if ($user === false) {
24                     $hash = password_hash($_POST['password'], PASSWORD_DEFAULT);
25                     $firstname = ucfirst(strtolower($_POST['firstname']));
26                     $lastname = ucfirst(strtolower($_POST['lastname']));
27                     $city = ucfirst(strtolower($_POST['city']));
28                     $street = ucfirst(strtolower($_POST['street']));
29
30                     $address_id = $service->addAddress($city, $_POST['zip'], $street, $_POST['flat_number']);
31                     $query = $service->signup($email, $hash, $firstname, $lastname, $_POST['phone'], $address_id['MAX(id)']);
32                     header('Location: index.php?site=logout');
33                 } else {
34                     $error = "Taki email już istnieje";
35                 }
36             } else {
37                 $error = "Podaj poprawny adres email";
38             }
39         } catch (Exception $e) {
40             print_r($e);
41         }
42     } else {
43         $error = "Wypełnij wymagane pola";
44     }
45     saveTmpData($_POST['email'], $_POST['password'], $_POST['firstname'], $_POST['lastname'], $_POST['city'], $_POST['street'],
```

Rysunek 34 – Podgląd pliku includes/UserService.php, przykład haszowania hasła i SQL injection.

Jak można zauważyć na powyższym rysunku nr. 32, operacja haszowania hasła oraz ochrony przed SQL Injection odbywa się po stronie aplikacji. W linii 24 można zaobserwować wykorzystanie funkcji która zwraca gotowy hash przygotowany dla hasła podanego jako argument funkcji. Natomiast w linii nr. 20 znajduje się funkcja „filter\_input” której zadanie jest sprawdzenie czy podany przez użytkownika adres email nie zawiera podejrzanych sformułowań, których można by użyć do niepożądanego wydobycia danych z bazy.