POLITECHNIKA WROCŁAWSKA
WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI

PROJEKT Z BAZ DANYCH

Bazodanowy system zarządzania oraz rezerwacji usług autokarowych

AUTOR:
Seweryn Pastuch

Indeks: 280897
E-mail: 280897@student.pwr.edu.pl

PROWADZĄCY ZAJĘCIA:
Dr inż. Robert Wójcik, K30W04D03

OCENA PRACY:

Wrocław, 2026

Spis treści

Spis rysunków

1. Wstęp

1.1. Cel projektu

Celem projektu jest opracowanie oraz implementacja systemu informatycznego wspierającego zadania pracowników firmy transportowej w zakresie obsługi rezerwacji przejazdów autokarowych, zarządzania flotą pojazdów, planowania grafików pracowników oraz obsługi klientów dokonujących rezerwacji. System umożliwia zarówno obsługę administracyjną (zarządzanie pracownikami i autobusami), jak i klientom – rezerwację oraz śledzenie stanu ich przejazdów.

Zasoby ludzkie:

Użytkownikami oprogramowania są kierowcy firmy oraz potencjalni klienci

1. Kierowca- w tej roli występuje każdy pracownik firmy.
2. Klient- wchodząc na stronę systemu
3. Kierownik firmy- jest tylko jeden

Celem inżynierskim jest stworzenie bezpiecznej, relacyjnej bazy danych, która nie tylko przechowuje informacje, ale aktywnie wspiera proces decyzyjny (np. poprzez walidację dostępności zasobów w czasie rzeczywistym). System ma za zadanie zminimalizować ryzyko błędów ludzkich, takich jak podwójna rezerwacja pojazdu.

1.2. Zakres projektu

Dokumentacja obejmuje pełen proces wytwórczy – od analizy wymagań, przez projektowanie architektury, aż po implementację i testy. Zakres funkcjonalny podzielono na trzy kluczowe moduły:

1. **Moduł Administracyjny (Kierownik):** Zarządzanie zasobami ludzkimi i sprzętowymi, wycena usług, nadzór nad globalnym grafikiem.
2. **Moduł Operacyjny (Kierowca):** Interfejs do podglądu grafiku

3. **Moduł Sprzedażowy (Klient):** System zapytań ofertowych i rezerwacji online.

2. Analiza wymagań

2.1. Opis działania i schemat logiczny systemu

System realizuje kompleksowe zarządzanie firmą transportową w oparciu o relacyjną bazę danych (tabele przechowujące dane o autokarach, pracownikach, klientach, zapytaniach i rezerwacjach). Z poziomu przeglądarki internetowej użytkownicy mogą wykonywać różne operacje w zależności od swojej roli w systemie:

- **Kierownik** firmy - ma pełne uprawnienia administracyjne: zarządza danymi o pracownikach, autobusach, zatwierdza dni wolne, wycenia przejazdy i akceptuje rezerwacje klientów.
- **Pracownik** - przegląda przydzielone przejazdy, wprowadza swoje dni wolne oraz śledzi harmonogram pracy.
- **Klient** - rejestruje się w systemie, wysyła zapytania o przejazdy autokarowe, otrzymuje automatyczną informację o dostępności terminów, akceptuje wyceny i potwierdza rezerwacje.

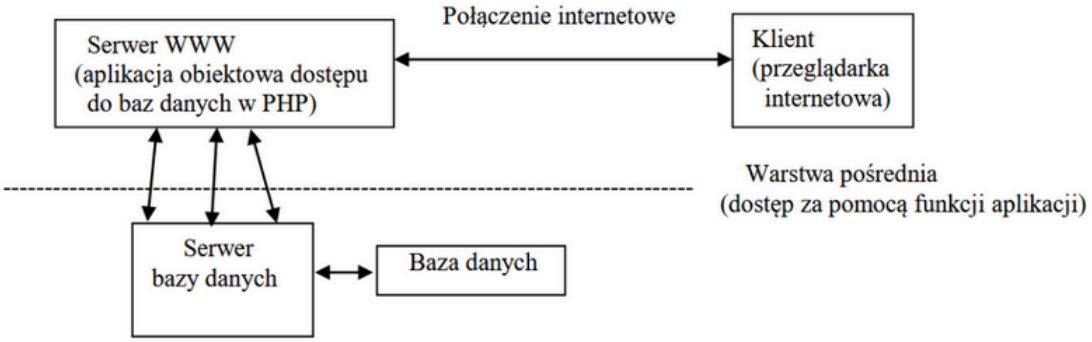
System umożliwia automatyczne sprawdzanie dostępności pojazdów i kierowców w wybranym terminie oraz częściowo automatyzuje proces odpowiedzi na zapytania klientów. Dodatkowo zapewnia kontrolę poprawności danych przy dodawaniu lub usuwaniu pracowników (np. niemożliwe je]st usunięcie pracownika z aktywnymi przejazdami).

3. Założenia architektoniczne przyjęte podczas realizacji systemu

System zostanie zaprojektowany zgodnie z trójwarstwowym modelem architektury klient–serwer, obejmującym:

- Warstwę prezentacji (frontend) - interfejs użytkownika dostępny z poziomu przeglądarki internetowej
- Warstwę logiki aplikacji (backend)
- Warstwę danych (baza danych)
- Aplikacja webowa komunikuje się z serwerem bazy danych za pomocą zapytań SQL

natomiast przetwarzanie logiki i operacji biznesowych jest realizowane po stronie serwera. Docelowo system będzie uruchamiany lokalnie (na localhost-cie), przy wykorzystaniu narzędzia phpMyAdmin do zarządzania strukturą bazy danych.



RYSUNEK 1. SCHEMAT KOMUNIKACJI, STRUKTURA SYSTEMU

2.2. Wymagania funkcjonalne

Aby sprecyzować wymagania funkcjonalne systemu przypomnijmy, że
Użytkownikami oprogramowania są kierowcyfirmy oraz potencjalni klienci

1) Kierowca- w tej roli występuje każdy pracownik firmy.

2) Klient- wchodząc na stronę systemu

3) Kierownik firmy - któy jest jednocześnie jej pracownikiem, administrator systemu.

Przepisy i strategie:

1) Osobą zarządzającą i administratorem danych osobowych pracownikpw i uzyt.jest Kierownik firmy.

2) System częściowo lub całkowicie (jeśli możliwe) automatyzuje operacje wykonywane przez jego użytkowników.

3) Warunkiem usunięcia pracownika jest nie posiadanie przez niego żadnych nie zrealizowanych przejazdów mssql server

Wymagania Funkcjonalne:

WF01) System przechowuje i przetwarza dane o:

- przejazdach, oraz o stanie ich realizacji, wycenie, datach oraz ich rozmiarach.

- pracownikach firmy, ich dniach wolnych, pracujących

- pojazdach, ich rozmiarze, marce, rejestracji oraz dostępności\

- klientach, ich rezerwacjach

WF02) Kierownik firmy zatwierdza przejazdy oraz wycenia je dla klienta.

WF03) Kierownik firmy dodaje, oraz usuwa pracowników.

WF04) Kierownik może ręcznie zarejestrować przejazd, bez ingerencji użytkownika.

WF05) Klient wysyła do systemu zapytania o możliwośc przejazdu z miejsca na miejsce konkretnym autokarem.

WF06) System wysyła do klienta automatyczną informację zwrotną, o dostępności autokaru i pomyślnego wysłania zapytania w proponowanej datcie przejazdu.

WF07) Klient tworzy konto, lub loguje się do niego.

WF08) Klient sprawdza wycenę w spodziewanej dacie(zaleznie od godziny wyslania zapytania).

WF9) Klient akceptuje proponowaną cenę.

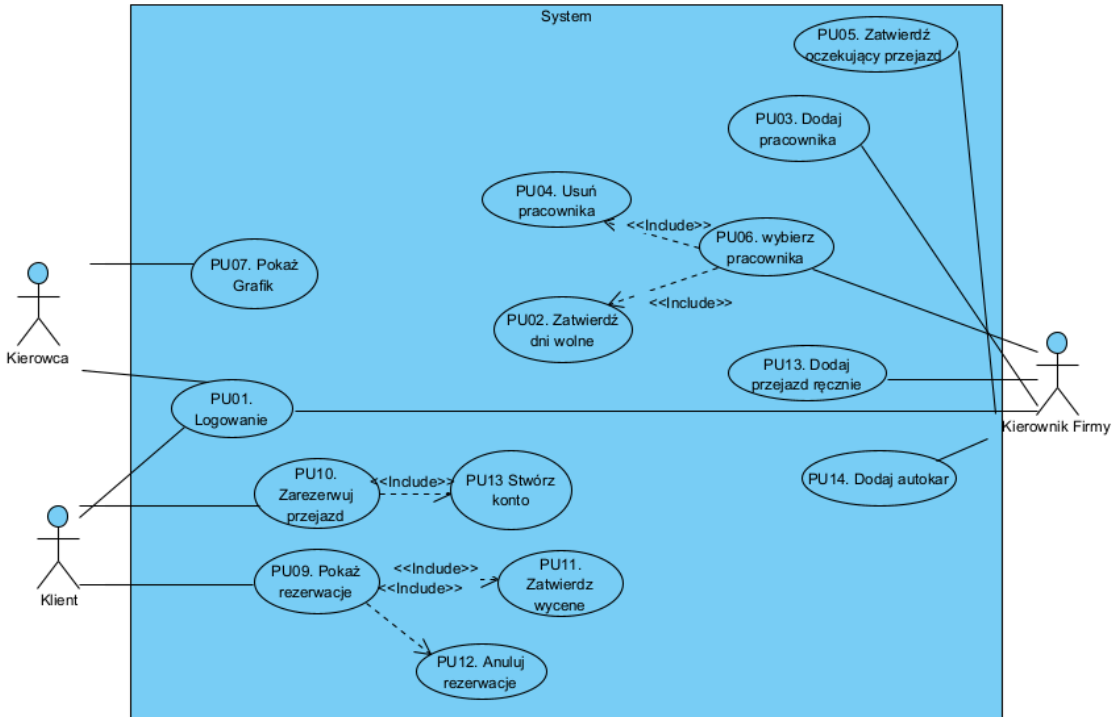
WF10) Klient widzi swoje rezerwacje.

WF11) Klient może anulować rezerwacje.

WF12) Kierowca widzi swój grafik.

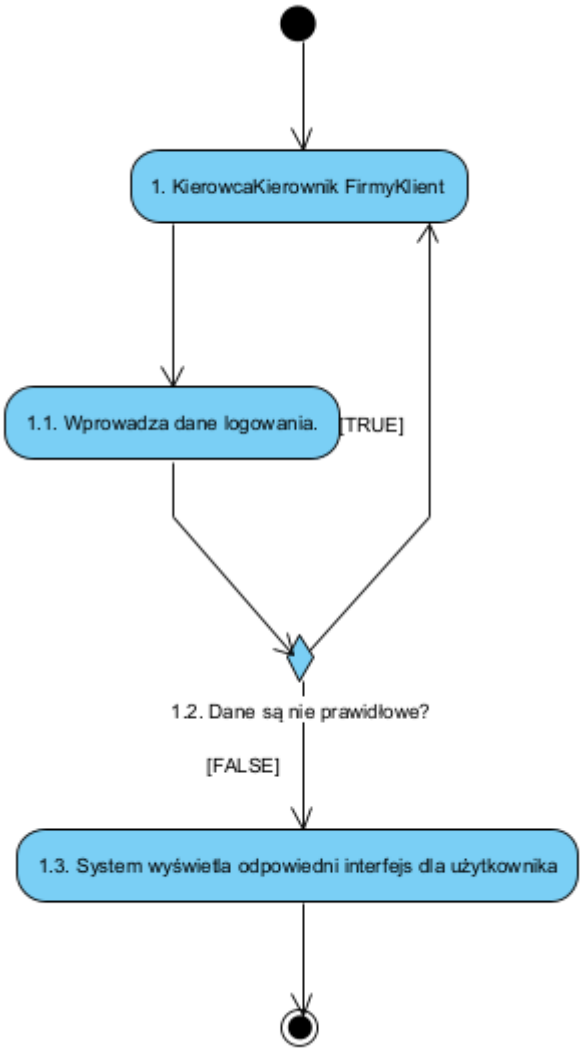
2.2.1. Diagram przypadków użycia

W wyniku Specyfikacji wymagań oraz analizy problemu stworzony został diagram przypadków użycia:

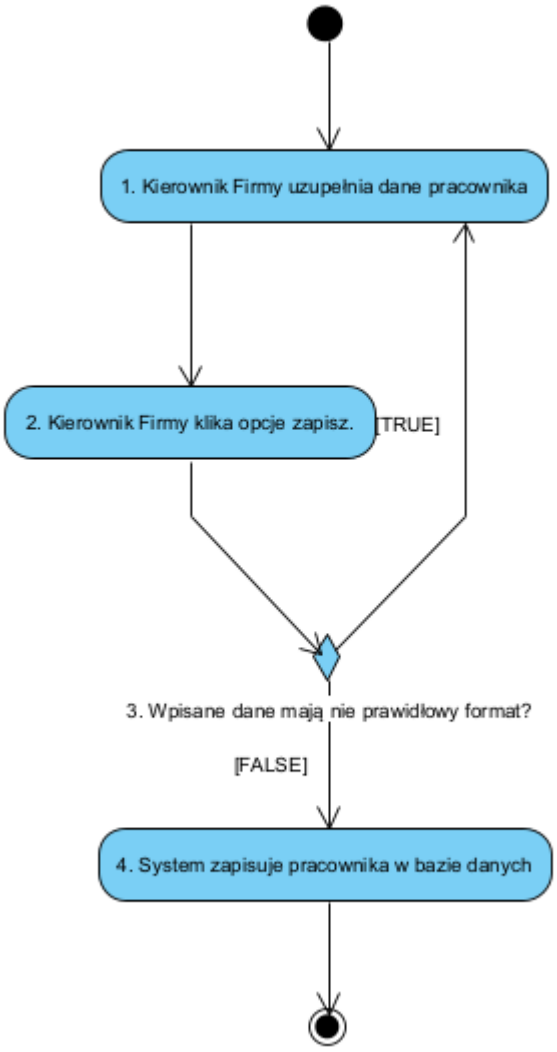


RYSUNEK 2. DIAGRAM PRZYPADKÓW UŻYCIA

2.2.2. Scenariusze wybranych przypadków użycia

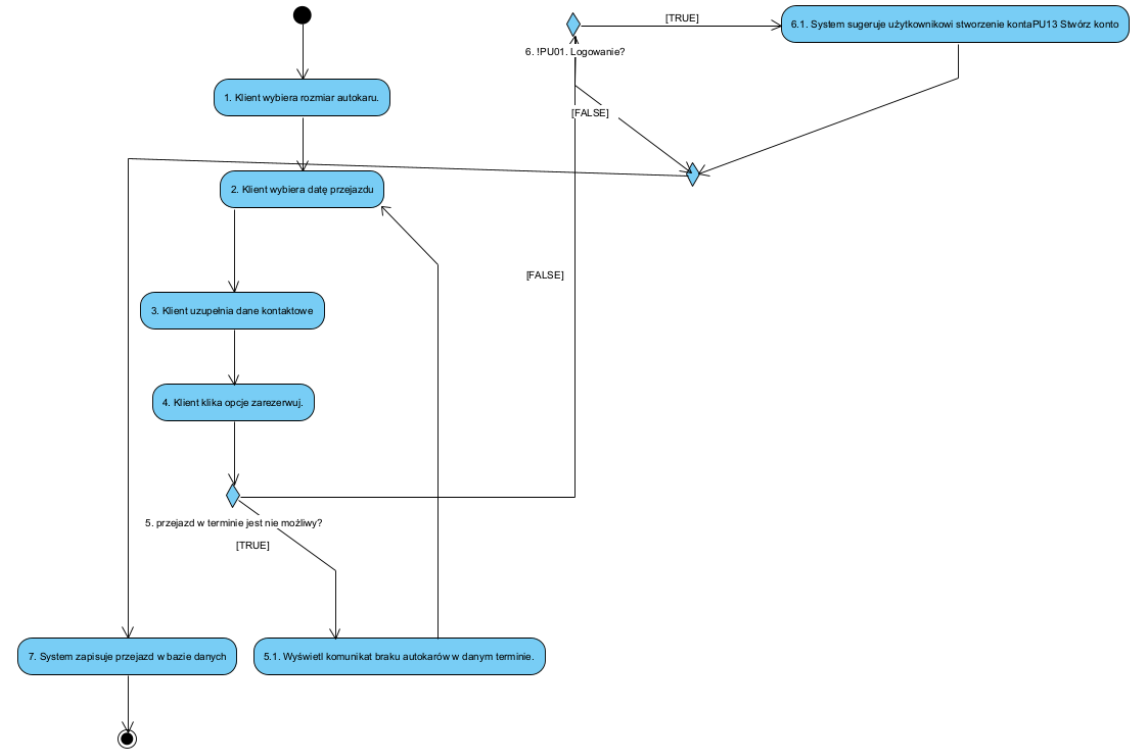


RYSUNEK 3. PU01 LOGOWANIE

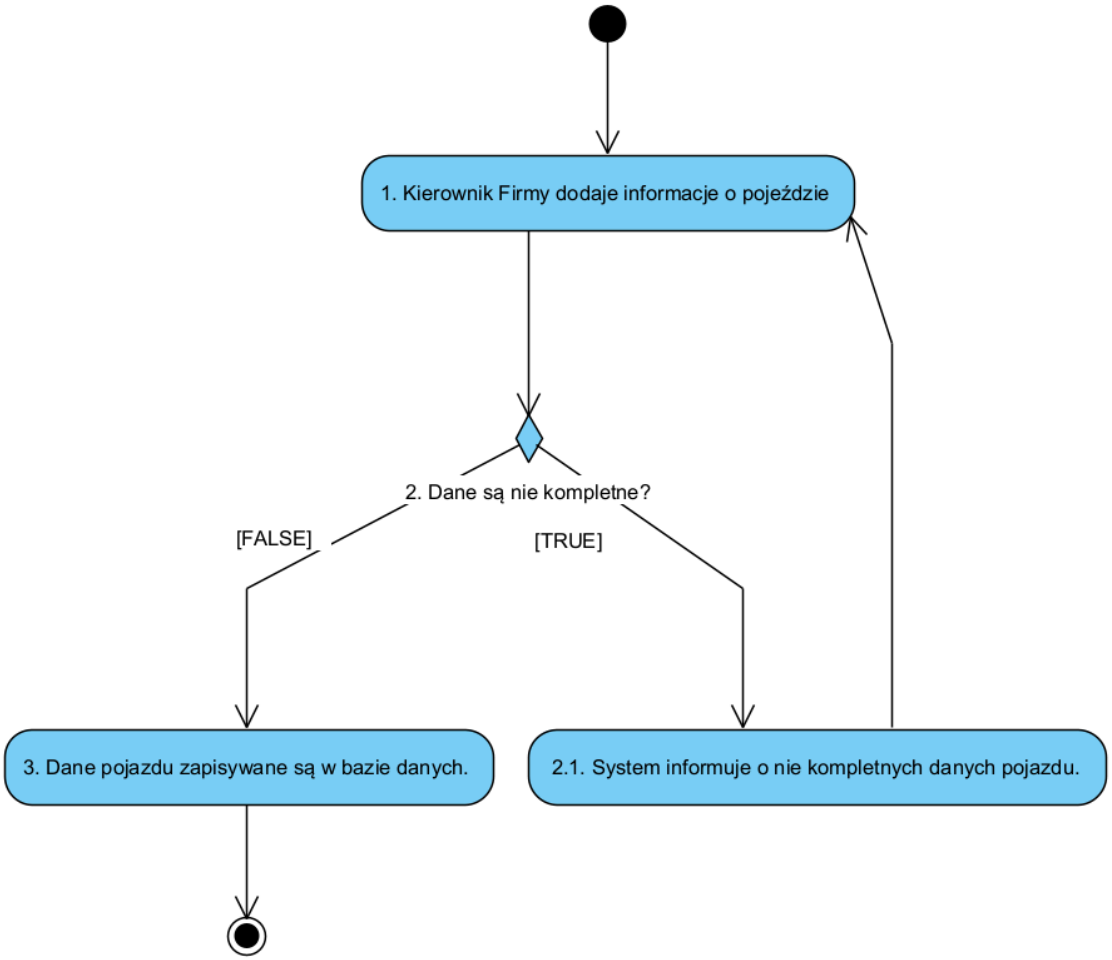


pu03 zatwierdz

Rysunek 4. Pu03 Dodaj pracownika



Rysunek 5. Pu10 Zarezerwuj Przejazd



RYSUNEK 6. PU 14 DODAJ AUTOKAR

2.3. Wymagania niefunkcjonalne

WNF1) System ma zapewniać bezpieczne przechowywanie haseł oraz danych użytkowników(szyfrowanie).

WNF2) System ma zapobiegać działaniom typu SQL injection

2.3.1. Wykorzystywane technologie i narzędzia

System został zrealizowany w oparciu o architekturę trójwarstwową, obejmującą warstwę prezentacji, logiki aplikacji oraz warstwę danych.

Warstwa prezentacji (frontend) została zaimplementowana przy użyciu technologii HTML oraz CSS i jest dostępna dla użytkownika za pośrednictwem przeglądarki internetowej Mozilla Firefox.

Warstwa logiki aplikacji (backend) została zrealizowana w postaci skryptów napisanych w języku PHP, uruchamianych po stronie serwera WWW Apache. Skrypty te odpowiadają za przetwarzanie danych wejściowych, realizację logiki biznesowej systemu oraz komunikację z bazą danych.

Warstwa danych została oparta na relacyjnej bazie danych SQL, zarządzanej za pomocą narzędzia phpMyAdmin. Serwer WWW oraz serwer bazy danych zostały uruchomione w

lokalnym środowisku testowym przy użyciu pakietu XAMPP na systemie operacyjnym Windows.

2.3.2. Wymagania dotyczące rozmiaru bazy danych

Na etapie projektowania systemu nie zostały zdefiniowane szczegółowe wymagania dotyczące maksymalnego rozmiaru bazy danych ani przewidywanej liczby przechowywanych rekordów. Projektowany system ma charakter demonstracyjno-edukacyjny i jest przeznaczony do pracy w lokalnym środowisku testowym, w związku z czym nie zakładano obsługi dużych wolumenów danych.

Zastosowana relacyjna baza danych umożliwia elastyczne skalowanie wraz ze wzrostem ilości przechowywanych informacji, bez konieczności wprowadzania zmian w strukturze logicznej systemu. Przyjęto założenie, że pojemność bazy danych będzie wystarczająca do poprawnego działania aplikacji w typowych scenariuszach użytkowania.

W przypadku wdrożenia systemu w środowisku produkcyjnym możliwe byłoby doprecyzowanie wymagań dotyczących rozmiaru bazy danych, w tym określenie maksymalnej liczby rekordów, objętości danych oraz mechanizmów archiwizacji lub optymalizacji przechowywania informacji.

2.3.3. Wymagania dotyczące bezpieczeństwa systemu

W projekcie systemu uwzględniono podstawowe wymagania dotyczące bezpieczeństwa przetwarzanych danych oraz komunikacji z bazą danych.

Komunikacja pomiędzy warstwą logiki aplikacji a bazą danych realizowana jest z wykorzystaniem interfejsu **PDO (PHP Data Objects)**, co umożliwia stosowanie zapytań parametryzowanych i ogranicza ryzyko wystąpienia ataków typu SQL Injection.

Dane uwierzytelniające użytkowników, w szczególności hasła, nie są przechowywane w postaci jawnej. Hasła są zabezpieczane przy użyciu mechanizmów kryptograficznych dostępnych w języku PHP, takich jak funkcje haszujące, co zapewnia ochronę danych nawet w przypadku nieautoryzowanego dostępu do bazy danych.

Dodatkowo system zakłada ograniczenie dostępu do zasobów aplikacji wyłącznie dla użytkowników posiadających odpowiednie uprawnienia, a jego działanie odbywa się w kontrolowanym środowisku lokalnym, co minimalizuje ryzyko zagrożeń zewnętrznych.

2.4. Przyjęte założenia projektowe

Na potrzeby realizacji projektu przyjęto zestaw założeń projektowych, które określają sposób działania systemu, zakres jego funkcjonalności oraz środowisko uruchomieniowe. Założenia te wynikają z edukacyjnego charakteru projektu oraz jego realizacji w ramach przedmiotu „Bazy danych”.

Przyjęto, że system będzie działał jako aplikacja webowa w architekturze trójwarstwowej klient–serwer, obejmującej warstwę prezentacji, warstwę logiki aplikacji oraz warstwę danych. Komunikacja pomiędzy poszczególnymi warstwami odbywa się za pośrednictwem zapytań SQL generowanych przez warstwę backendową.

Zakłada się, że system będzie uruchamiany w lokalnym środowisku testowym (localhost), bez wdrożenia produkcyjnego oraz bez obsługi jednoczesnej pracy dużej liczby użytkowników. W związku z tym nie uwzględniono mechanizmów równoważenia obciążenia ani zaawansowanych rozwiązań wysokiej dostępności.

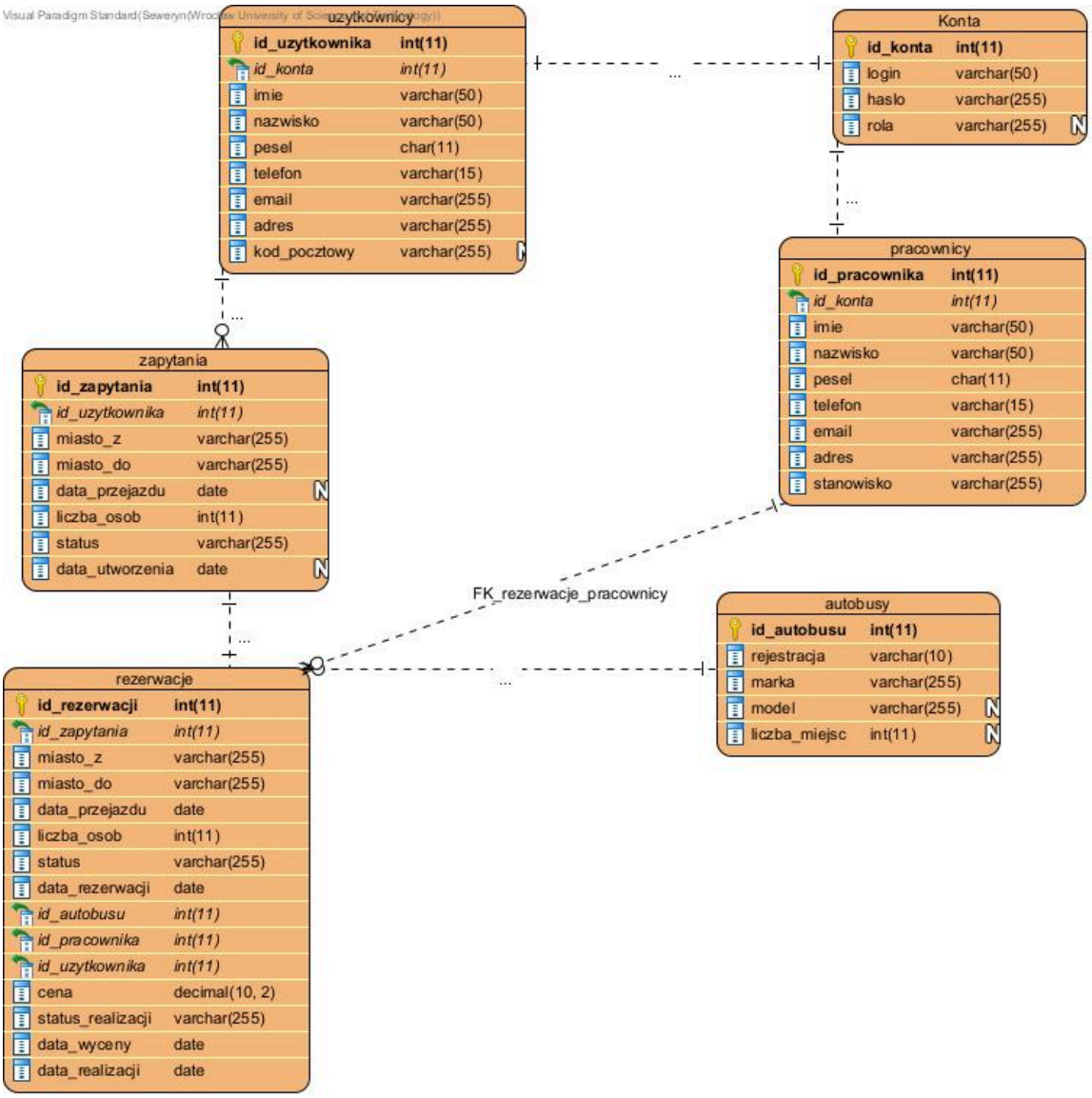
Przyjęto, że użytkownicy systemu posiadają podstawową znajomość obsługi aplikacji internetowych, a interfejs użytkownika nie musi spełniać rygorystycznych wymagań dostępności.

Założono również, że system będzie wykorzystywany w kontrolowanym środowisku, co pozwala ograniczyć zakres implementowanych mechanizmów bezpieczeństwa do podstawowych rozwiązań, takich jak uwierzytelnianie użytkowników, kontrola dostępu oraz zabezpieczenie komunikacji z bazą danych przed atakami typu SQL Injection.

Struktura bazy danych została zaprojektowana z myślą o poprawności logicznej, integralności danych oraz czytelności projektu, bez optymalizacji pod kątem bardzo dużych wolumenów danych. Ewentualna rozbudowa systemu w przyszłości wymagałaby ponownej analizy wymagań niefunkcjonalnych.

3. Projekt systemu

3.1. Projekt bazy danych



RYSUNEK 7. DIAGRAM ERD BAZY DANYCH.

3.1.1. Analiza rzeczywistości i uproszczony model konceptualny

Analiza rzeczywistości została przeprowadzona w oparciu o procesy zachodzące w firmie transportowej realizującej przewozy autokarowe na zlecenie klientów. Podstawowym celem systemu jest ewidencjonowanie rezerwacji przejazdów, zarządzanie flotą pojazdów oraz obsługa użytkowników systemu o różnych rolach.

W analizowanym obszarze wyróżniono następujące główne obiekty rzeczywiste:

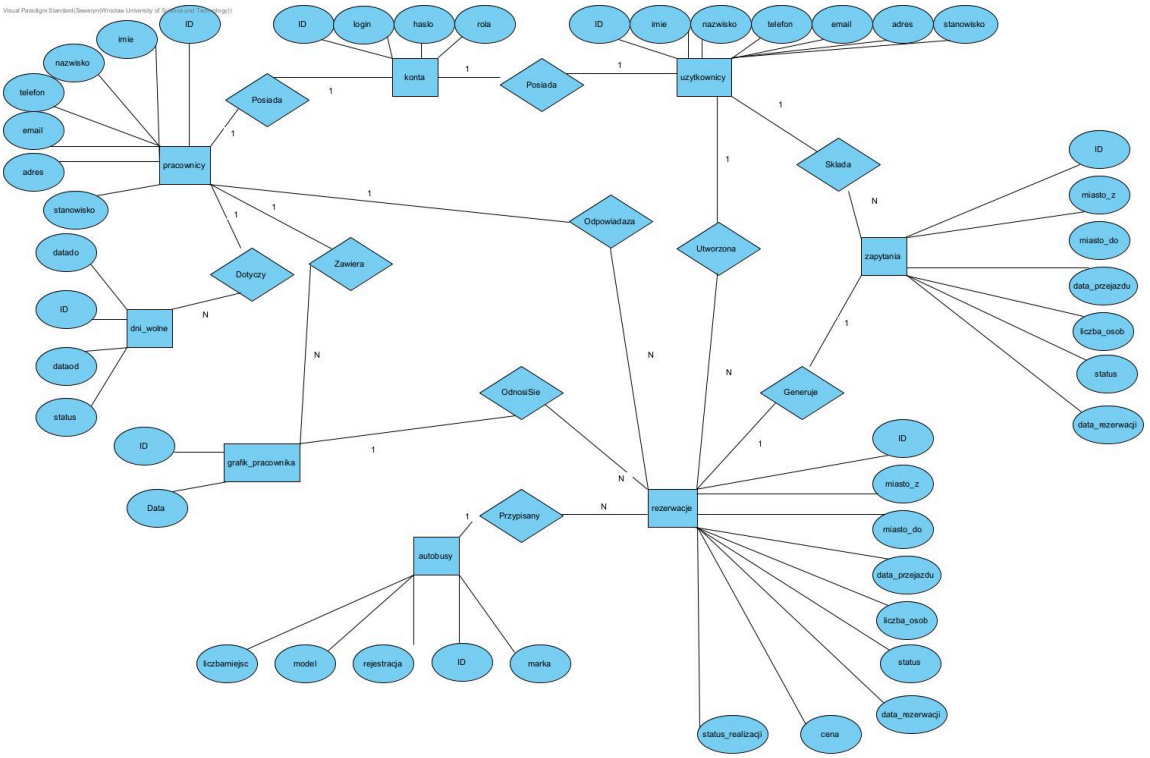
- **Autobus**, będący środkiem transportu wykorzystywanym do realizacji przejazdów,

- **Rezerwacja**, opisująca zlecenie przejazdu w określonym terminie,
- **Konto**, reprezentujące dane uwierzytelniające użytkowników systemu,
- **Pracownik**, realizujący przejazdy oraz pełniący funkcje administracyjne,
- **Użytkownik (klient)**, dokonujący rezerwacji przejazdów.

Pomiędzy wyróżnionymi obiektami zachodzą jednoznaczne relacje:

- rezerwacja jest powiązana z dokładnie jednym autobusem,
- rezerwacja może być przypisana do jednego pracownika,
- każda rezerwacja jest tworzona przez użytkownika posiadającego konto w systemie,
- zarówno pracownicy, jak i użytkownicy systemu są powiązani z kontami służącymi do logowania.

Na podstawie tej analizy opracowano uproszczony model konceptualny w Notacji chena, w którym uwzględniono wyłącznie encje oraz relacje pomiędzy nimi, pomijając szczegóły implementacyjne takie jak typy danych czy klucze techniczne.



RYSUNEK 8. MODEL KONCEPTUALNY BAZY DANYCH NOTACJI CHENA

3.1.2. Model logiczny i normalizacja

Na podstawie modelu konceptualnego opracowano model logiczny bazy danych, w którym zdefiniowano strukturę tabel, ich atrybuty oraz relacje pomiędzy nimi. Model logiczny obejmuje następujące tabele: `autobusy`, `konta`, `pracownicy`, `uzytkownicy` oraz `rezerwacje`.

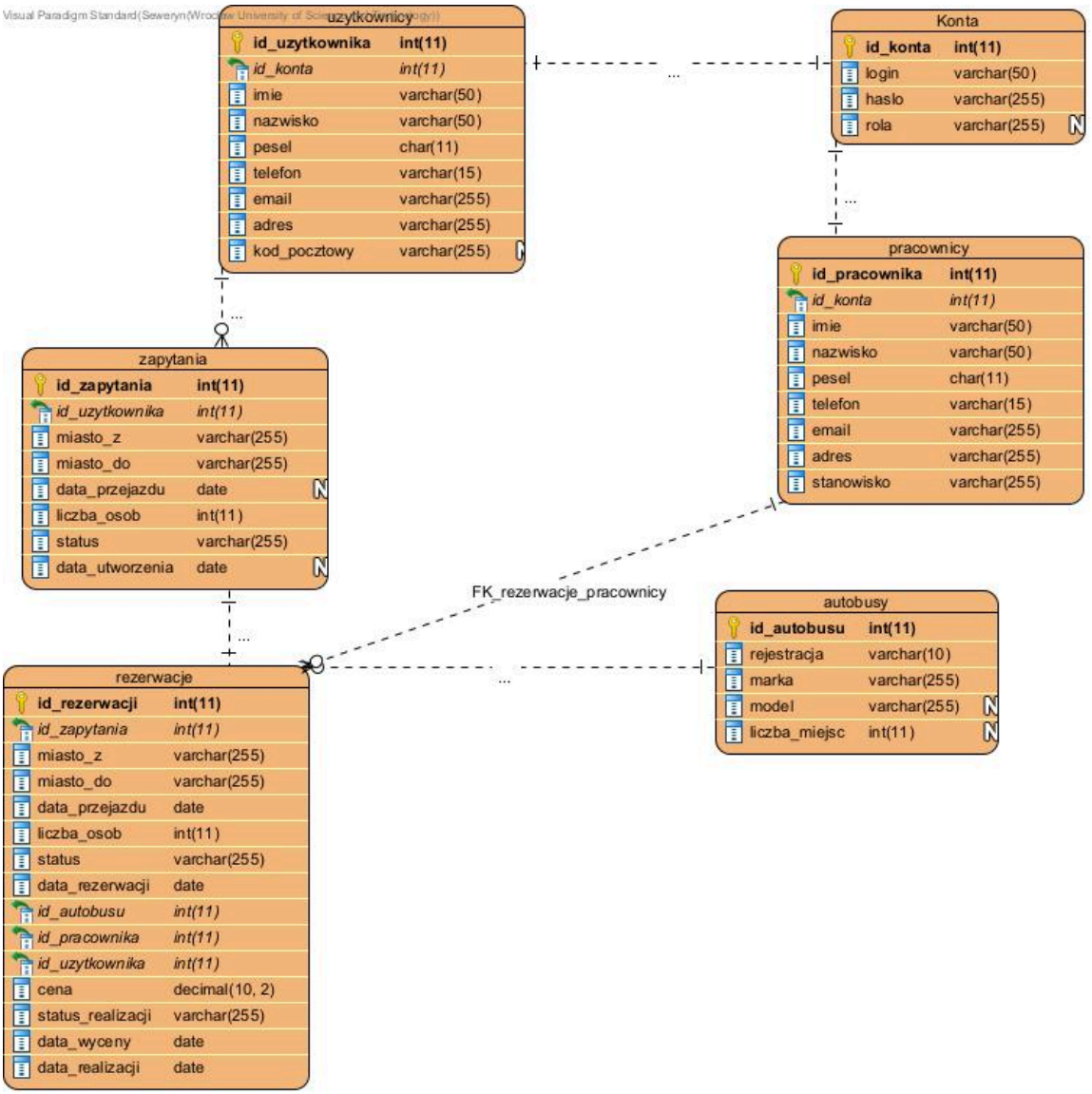
Każda z tabel posiada jednoznaczny klucz główny, a relacje pomiędzy encjami zostały odwzorowane przy użyciu kluczy obcych. W szczególności:

- tabela `pracownicy` oraz `uzytkownicy` zawiera klucz obcy `id_konta`, wskazujący na tabelę `konta`,
- tabela `rezerwacje` zawiera klucze obce wskazujące na tabele `konta`, `autobusy` oraz `pracownicy`.

Model logiczny został zaprojektowany zgodnie z zasadami normalizacji relacyjnych baz danych. Dane zostały doprowadzone do trzeciej postaci normalnej (3NF), co oznacza, że:

- każda tabela przechowuje dane dotyczące jednego typu obiektu,
- wszystkie atrybuty są funkcjonalnie zależne od klucza głównego,
- wyeliminowano redundancję danych oraz zależności przechodnie.

Dzięki normalizacji uzyskano spójną, czytelną strukturę bazy danych, ułatwiającą dalszą rozbudowę systemu oraz zapewniającą integralność logiczną danych.



Rysunek 9. Diagram loficzny ERD bazy danych (Crows foot notation)

3.1.3. Model fizyczny i ograniczenia integralności danych

Model fizyczny bazy danych został zaimplementowany w relacyjnym systemie zarządzania bazą danych i odwzorowuje bezpośrednio model logiczny w postaci konkretnych tabel, kolumn oraz typów danych. W projekcie zastosowano typy danych adekwatne do charakteru przechowywanych informacji, takie jak `INT`, `VARCHAR`, `DATE`, `DATETIME`, `TIME` oraz `DECIMAL`.

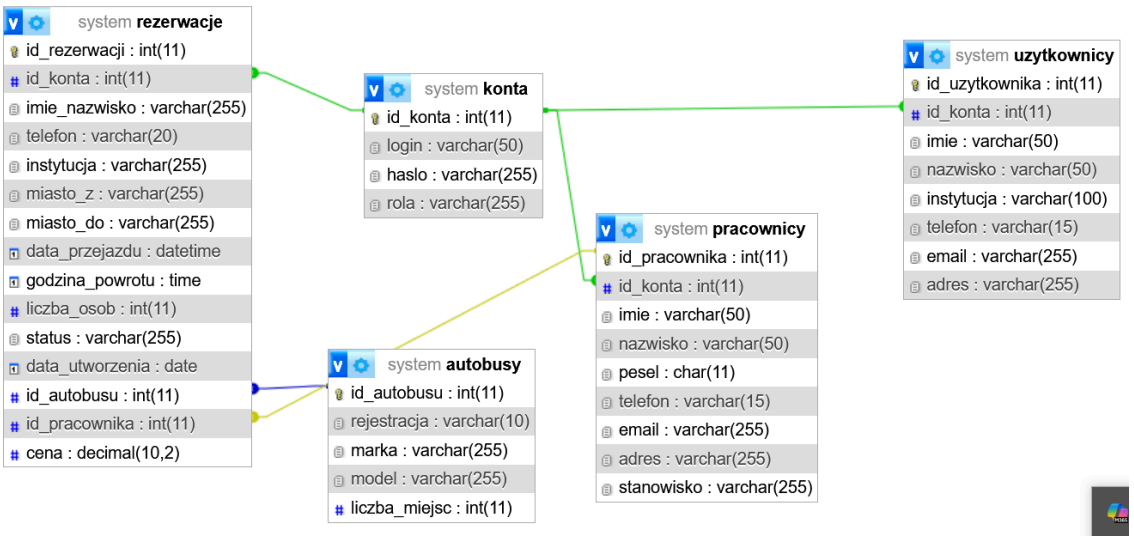
Integralność danych zapewniono poprzez zastosowanie następujących mechanizmów:

- klucze główne (PRIMARY KEY), jednoznacznie identyfikujące rekordy w każdej tabeli,
- klucze obce (FOREIGN KEY), wymuszające poprawność relacji pomiędzy tabelami,
- ograniczenia NOT NULL dla pól wymaganych,
- mechanizm AUTO_INCREMENT dla kluczy głównych.

Zdefiniowane ograniczenia integralności zapobiegają m.in.:

- dodaniu rezerwacji dla nieistniejącego autobusu lub użytkownika,
- przypisaniu rezerwacji do nieistniejącego pracownika,
- istnieniu rekordów pracowników lub użytkowników bez powiązanego konta.

Dzięki zastosowaniu powyższych mechanizmów baza danych zapewnia spójność referencyjną oraz poprawność przechowywanych informacji, co ma kluczowe znaczenie dla prawidłowego działania całego systemu.



RYSUNEK 10.MODEL FIZYCZNY BAZY DANYCH

3.1.4. Inne elementy schematu – mechanizmy przetwarzania danych

W ramach projektu bazy danych nie zastosowano zaawansowanych mechanizmów przetwarzania danych na poziomie serwera bazy danych, takich jak procedury składowane, funkcje użytkownika czy wyzwalacze (triggery).

Przetwarzanie danych oraz realizacja logiki biznesowej systemu odbywa się po stronie aplikacji, w warstwie logiki aplikacyjnej, zaimplementowanej w języku PHP. Takie podejście umożliwia większą elastyczność w modyfikacji logiki systemu oraz uproszczenie struktury bazy danych.

Na poziomie bazy danych realizowane są wyłącznie podstawowe operacje manipulacji danymi (INSERT, UPDATE, DELETE), inicjowane przez aplikację webową. Kontrola poprawności danych opiera się głównie na ograniczeniach integralności zdefiniowanych w modelu fizycznym.

3.1.5. Projekt mechanizmów bezpieczeństwa na poziomie bazy danych

Bezpieczeństwo danych w projekcie zostało zapewnione głównie poprzez zastosowanie mechanizmów integralności oraz kontroli dostępu na poziomie aplikacji. Na poziomie bazy danych zaimplementowano podstawowe mechanizmy bezpieczeństwa.

Dostęp do bazy danych realizowany jest wyłącznie poprzez dedykowane konto użytkownika bazy danych, wykorzystywane przez aplikację. Użytkownicy końcowi nie posiadają bezpośredniego dostępu do struktury bazy danych ani do danych w niej przechowywanych.

Dodatkowo zastosowano:

- klucze obce zapewniające spójność referencyjną,
- ograniczenia `NOT NULL` zapobiegające zapisywaniu niekompletnych danych,
- separację danych uwierzytelniających (tabela `konta`) od danych osobowych użytkowników.

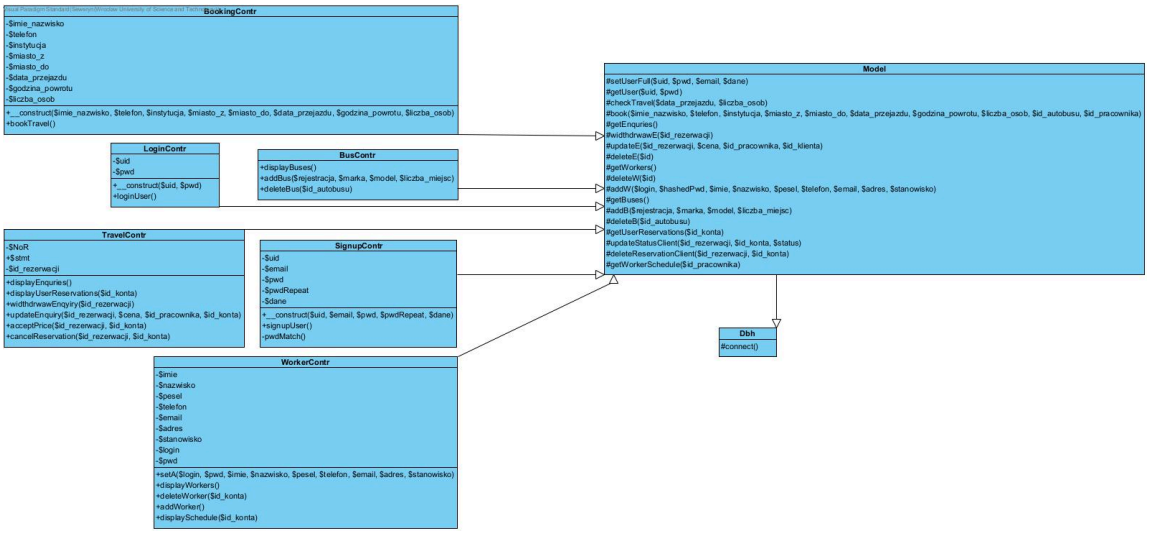
Zabezpieczenie przed nieautoryzowanym dostępem oraz atakami typu SQL Injection realizowane jest w warstwie aplikacyjnej poprzez zastosowanie zapytań parametryzowanych przy użyciu interfejsu PDO.

3.2. Projekt aplikacji użytkownika

Projekt aplikacji użytkownika obejmuje strukturę logiczną warstwy aplikacyjnej systemu oraz sposób organizacji kodu odpowiedzialnego za obsługę logiki biznesowej i komunikację z bazą danych. Aplikacja została zaprojektowana jako aplikacja webowa działająca w architekturze klient–serwer, w której użytkownik komunikuje się z systemem za pośrednictwem przeglądarki internetowej.

Logika aplikacji została zaimplementowana w języku PHP i opiera się na podziale funkcjonalnym na klasy odpowiedzialne za obsługę poszczególnych obszarów systemu, takich jak logowanie i rejestracja użytkowników, zarządzanie rezerwacjami przejazdów, obsługa danych pracowników oraz zarządzanie flotą pojazdów. Struktura projektu została oparta na podejściu architektonicznym zbliżonym do wzorca MVC (Model–View–Controller), co zapewnia czytelny podział odpowiedzialności pomiędzy warstwę prezentacji, logikę aplikacyjną oraz warstwę dostępu do danych. Modularna organizacja kodu ułatwia jego utrzymanie, rozwój oraz ewentualną rozbudowę funkcjonalności systemu.

3.2.1. Architektura aplikacji i diagramy projektowe



RYSUNEK 11. DIAGRAM KLAS SYSTEMU

Architektura aplikacji użytkownika została zaprojektowana w sposób modularny, z wykorzystaniem klas pełniących rolę kontrolerów, które delegują operacje do wspólnej warstwy dostępu do danych. Centralnym elementem architektury jest klasa bazowa `Model`, z której dziedziczą wszystkie pozostałe klasy aplikacji.

Relacje pomiędzy klasami zostały przedstawione na diagramie klas UML i mają charakter **generalizacji (dziedziczenia)**. Klasa `Model` udostępnia wspólne mechanizmy komunikacji z bazą danych, takie jak wykonywanie zapytań SQL oraz obsługa połączenia z bazą danych. Klasy

pochodne, takie jak kontrolery odpowiedzialne za rezerwacje, użytkowników, pracowników czy pojazdy, rozszerzają funkcjonalność klasy bazowej o logikę specyficzną dla danego obszaru systemu.

Zastosowanie relacji generalizacji pozwoliło na:

- eliminację powielania kodu,
- centralizację mechanizmów dostępu do danych,
- ujednolicenie sposobu komunikacji z bazą danych,
- uproszczenie struktury aplikacji.

W ramach zaprojektowanej architektury aplikacji użytkownika wyróżniono następujące klasy wraz z ich rolami w systemie:

- **Model** – klasa bazowa, stanowiąca wspólną warstwę dostępu do danych dla całej aplikacji. Odpowiada za nawiązywanie połączenia z bazą danych, wykonywanie zapytań SQL oraz obsługę mechanizmów PDO. Klasa ta nie realizuje logiki biznesowej, lecz udostępnia funkcjonalności wykorzystywane przez klasy pochodne.
- **Dbh** – klasa odpowiedzialna za konfigurację oraz inicjalizację połączenia z bazą danych. Zapewnia jednolity sposób dostępu do zasobów bazy danych i stanowi techniczne zaplecze dla klasy `Model`.
- **LoginContr** – klasa kontrolera odpowiedzialna za obsługę procesu logowania użytkowników do systemu. Realizuje weryfikację danych uwierzytelniających, inicjalizację sesji użytkownika oraz kontrolę poprawności logowania.
- **SignupContr** – klasa kontrolera obsługująca proces rejestracji nowych użytkowników systemu. Odpowiada za walidację danych rejestracyjnych oraz tworzenie nowych kont w bazie danych.
- **TravelContr** – klasa kontrolera odpowiedzialna za obsługę rezerwacji przejazdów. Realizuje operacje związane z przeglądaniem rezerwacji, ich aktualizacją,

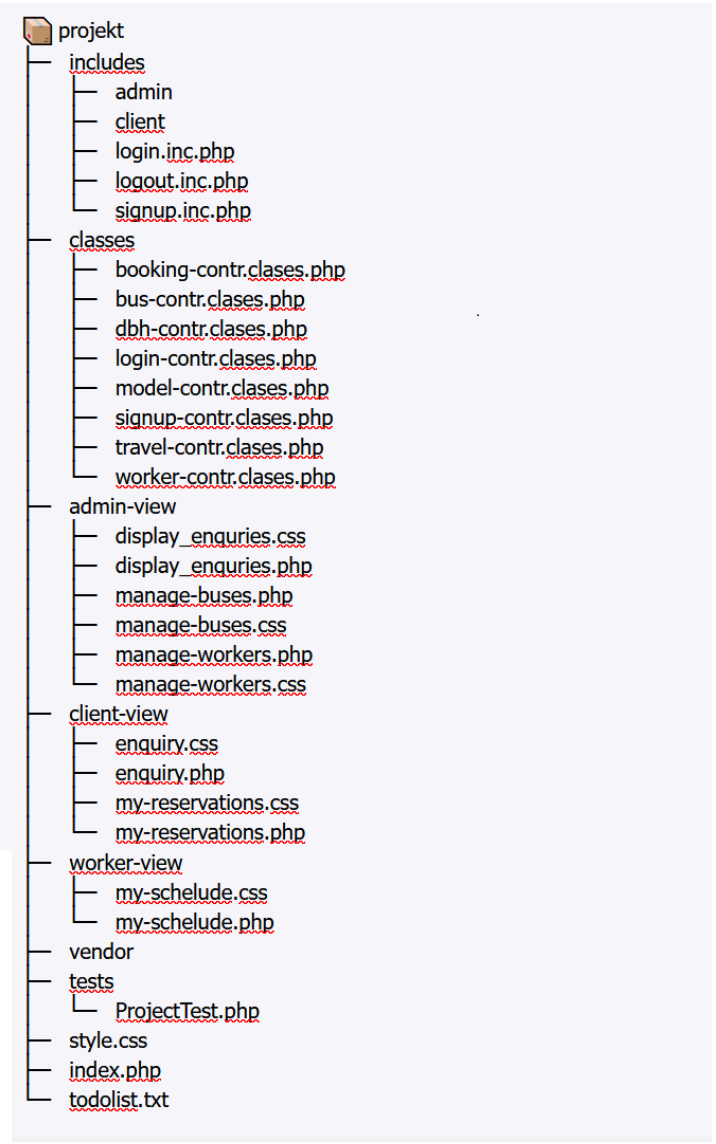
zatwierdzaniem cen oraz anulowaniem rezerwacji przez użytkowników lub administratora.

- **BookingContr** – klasa kontrolera wspierająca proces tworzenia oraz zarządzania zgłoszeniami rezerwacyjnymi. Odpowiada za logikę związaną z dodawaniem nowych rezerwacji oraz ich powiązaniem z użytkownikami i zasobami systemu.

- **WorkerContr** – klasa kontrolera odpowiedzialna za obsługę danych pracowników systemu. Umożliwia zarządzanie informacjami o pracownikach oraz ich przypisaniem do realizowanych przejazdów.

- **BusContr** – klasa kontrolera odpowiedzialna za zarządzanie flotą pojazdów. Obsługuje operacje dodawania, edycji oraz pobierania danych dotyczących autobusów wykorzystywanych w systemie.

W celu uzupełnienia diagramu klas oraz lepszego zobrazowania organizacji kodu źródłowego aplikacji, przygotowano dodatkowy rysunek przedstawiający strukturę plików projektu. Zaprezentowany podział odzwierciedla przyjętą architekturę aplikacji oraz sposób separacji warstw odpowiedzialnych za logikę biznesową, obsługę danych oraz prezentację interfejsu użytkownika.



Rysunek 12. Struktura Plików Projektu

1. Folder główny (`projekt/`)

Stanowi fundament aplikacji i zawiera pliki konfiguracyjne oraz główny punkt wejścia do systemu:

- **`index.php`**: Główny plik aplikacji pełniący rolę dynamicznego panelu startowego. Wyświetla on formularze logowania i rejestracji dla gości oraz spersonalizowane pulpity nawigacyjne (dashboardy) dla zalogowanych użytkowników w zależności od ich roli: kierownika, pracownika lub klienta.
- **`style.css`**: Główny arkusz stylów definiujący wspólny wygląd dla całego systemu.

- **`composer.json`** oraz **`composer.lock`**: Pliki konfiguracyjne menedżera pakietów Composer, definiujące m.in. zależności dla testów jednostkowych (PHPUnit).
- **`todolist.txt`**: Dokumentacja wewnętrzna zawierająca listę funkcjonalności do zaimplementowania oraz założenia projektowe.

2. Folder `classes/`

Zawiera rdzeń logiczny aplikacji oparty na programowaniu obiektowym. Architektura opiera się na dziedziczeniu z centralnej klasy `Model`:

- **`dbh.classes.php`**: Klasa techniczna odpowiedzialna za nawiązywanie i konfigurację połączenia z bazą danych przy użyciu interfejsu PDO.
- **`model-classes.php`**: Centralna klasa bazy danych, z której dziedziczą wszystkie kontrolery. Zawiera chronione metody wykonujące bezpośrednie operacje na tabelach SQL (np. `setUserFull`, `getEnquries`, `updateE`).
- **Kontrolery (`*-contr.classes.php`)**: Specjalistyczne klasy rozszerzające `Model`, które odpowiadają za konkretną logikę biznesową systemu: obsługę logowania, rejestracji, zarządzanie rezerwacjami, flotą pojazdów oraz grafikami pracowników.

3. Folder `includes/`

Zawiera skrypty procesowe, które nie generują bezpośrednio widoku, ale pośredniczą w przesyłaniu danych z formularzy do odpowiednich kontrolerów:

- **`login.inc.php`** i **`signup.inc.php`**: Obsługa logowania i rejestracji użytkowników.
- **Podfolder `admin/`**: Skrypty do zarządzania zasobami (dodawanie pojazdów, pracowników, aktualizacja rezerwacji przez kierownika).
- **Podfolder `client/`**: Skrypty obsługujące akcje klienta (wysyłanie zapytań o przejazd, akceptacja wyceny, odwoływanie rezerwacji).

4. Foldery widoków (`*-view/`)

Podzielone zgodnie z uprawnieniami użytkowników, zawierają pliki `.php` odpowiedzialne za warstwę prezentacji (HTML) oraz dedykowane im pliki `.css`:

- **`admin-view/`**: Panele przeznaczone dla kierownika, umożliwiające zarządzanie całą flotą (`manage-buses.php`), kadrą (`manage-workers.php`) oraz listą wszystkich zapytań i rezerwacji (`display_enquries.php`).
- **`client-view/`**: Interfejs dla klienta, zawierający formularz zapytania o przejazd (`enquiry.php`) oraz listę własnych rezerwacji wraz z ich statusami i wycenami (`my-reservations.php`).
- **`worker-view/`**: Widok dla kierowców, prezentujący ich indywidualny grafik zatwierdzonych przejazdów (`my-schelude.php`).

5. Folder `tests/`

- **`ProjectTest.php`**: Zawiera testy jednostkowe przygotowane w środowisku PHPUnit, służące do weryfikacji poprawności działania poszczególnych modułów systemu.

3.2.2. Interfejs graficzny i struktura menu

Interfejs graficzny Systemu Rezerwacji Przejazdów został zaprojektowany z naciskiem na przejrzystość, intuicyjność oraz responsywność, co pozwala na wygodną obsługę systemu zarówno na urządzeniach stacjonarnych, jak i mobilnych. Warstwa wizualna opiera się na spójnej palecie barw i nowoczesnych elementach UI (takich jak karty panelu użytkownika czy tabele typu agenda), co buduje profesjonalny wizerunek aplikacji.

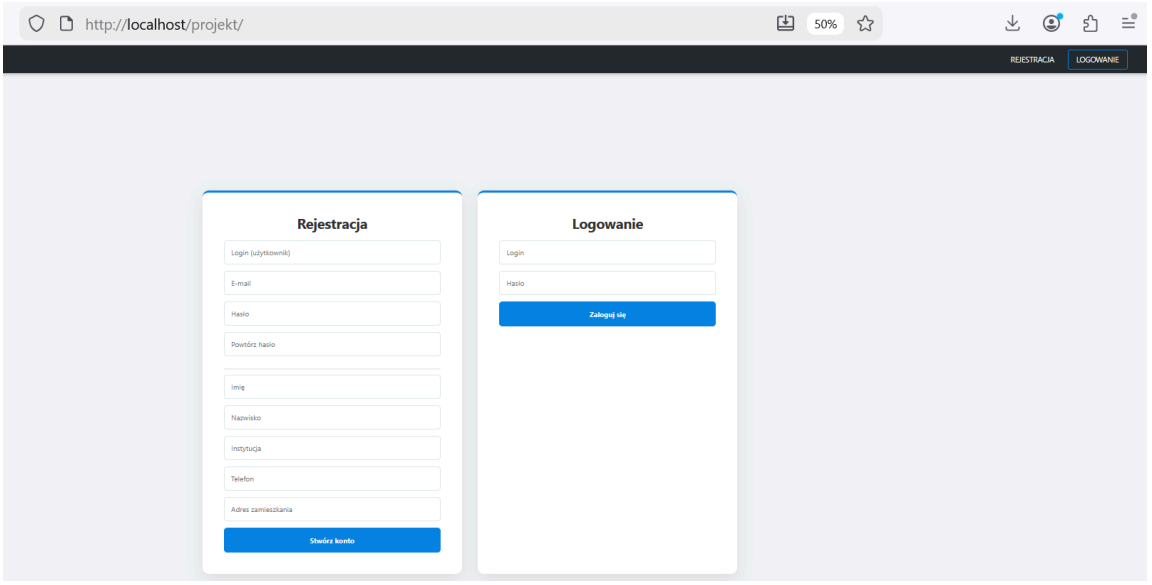
Struktura menu z pliku **index.php** oraz dostępna zawartość interfejsu są ściśle uzależnione od uprawnień zalogowanego użytkownika (Role-Based Access Control), co zapewnia bezpieczeństwo i personalizację doświadczenia użytkownika:

- **Gość (niezalogowany):** Posiada dostęp do strony głównej, na której prezentowane są wyłącznie formularze rejestracji oraz logowania. Interfejs prowadzi użytkownika krok po kroku przez proces zakładania konta, wymagając podania niezbędnych danych kontaktowych i instytucjonalnych.
- **Klient:** Po zalogowaniu otrzymuje dostęp do panelu użytkownika, który umożliwia sprawne tworzenie nowych zapytań o przejazd oraz monitorowanie statusu i wyceny już istniejących rezerwacji. Menu klienta jest uproszczone, aby maksymalnie skrócić ścieżkę od pomysłu do wysłania formularza.
- **Pracownik (Kierowca):** Widok ten jest zorientowany na zadania i terminy. Interfejs ogranicza się do przejrzystego grafiku, który w formie kart (job-cards) prezentuje

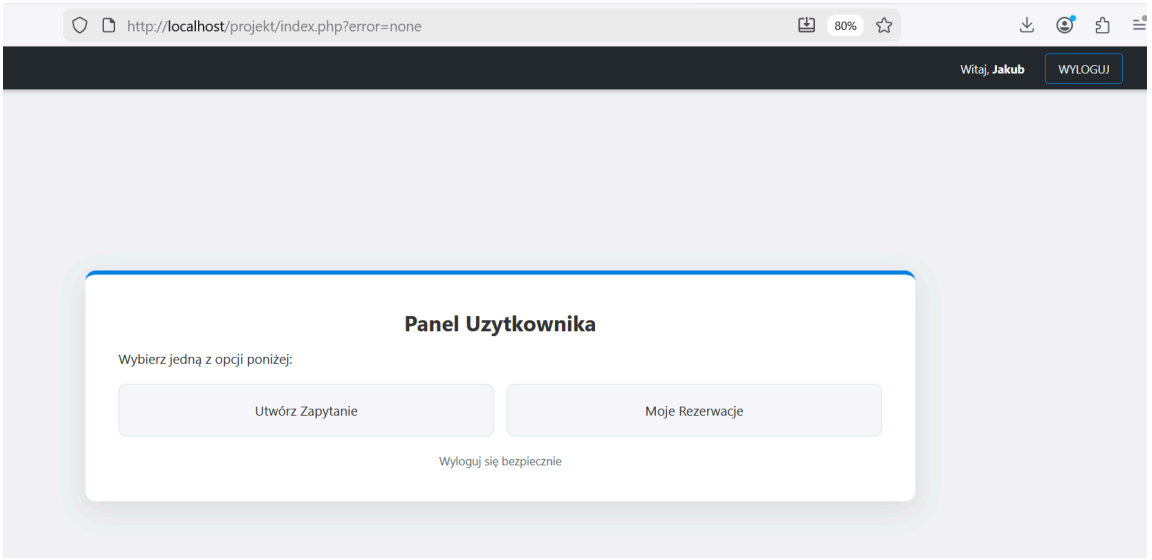
szczegóły przypisanych tras, dane kontaktowe klienta oraz informacje o przydzielonym pojeździe.

- **Kierownik (Administrator):** Dysponuje najbardziej rozbudowanym interfejsem, podzielonym na moduły tematyczne: zarządzanie flotą, kadrą oraz operacyjne zarządzanie rezerwacjami. System nawigacji pozwala na szybkie przełączanie się między listą pracowników a agendą przejazdów, gdzie administrator może dokonywać wycen i przypisywać zasoby.

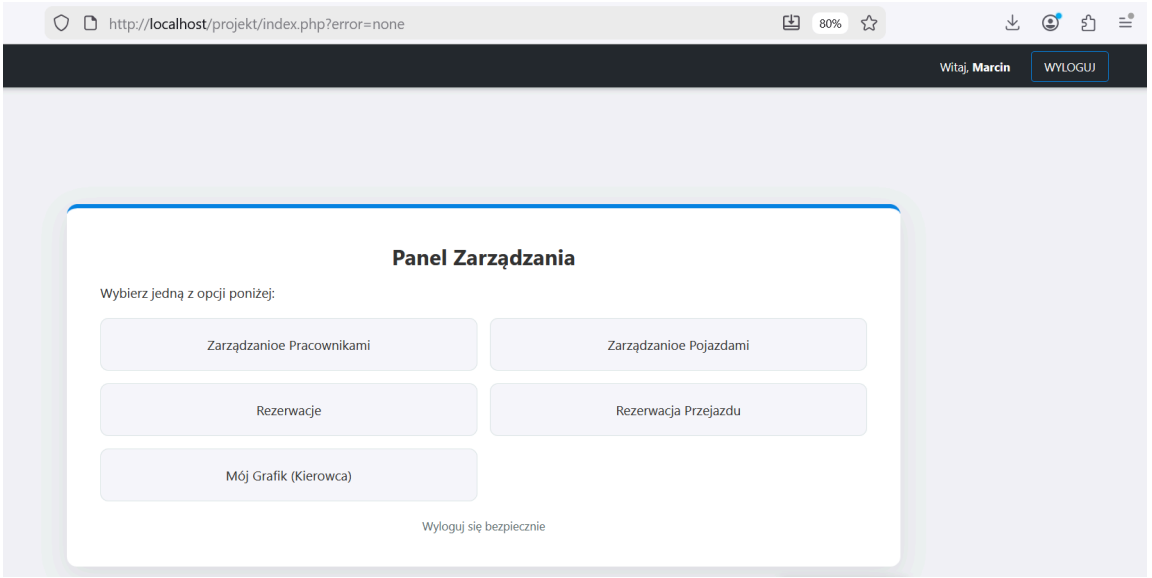
Wszystkie widoki łączy wspólny nagłówek (header) zawierający logikę powitalną oraz przycisk bezpiecznego wylogowania, co ujednolica nawigację w obrębie całego systemu. Poniżej przedstawiono szczegółową analizę oraz zrzuty ekranu poszczególnych widoków systemu.



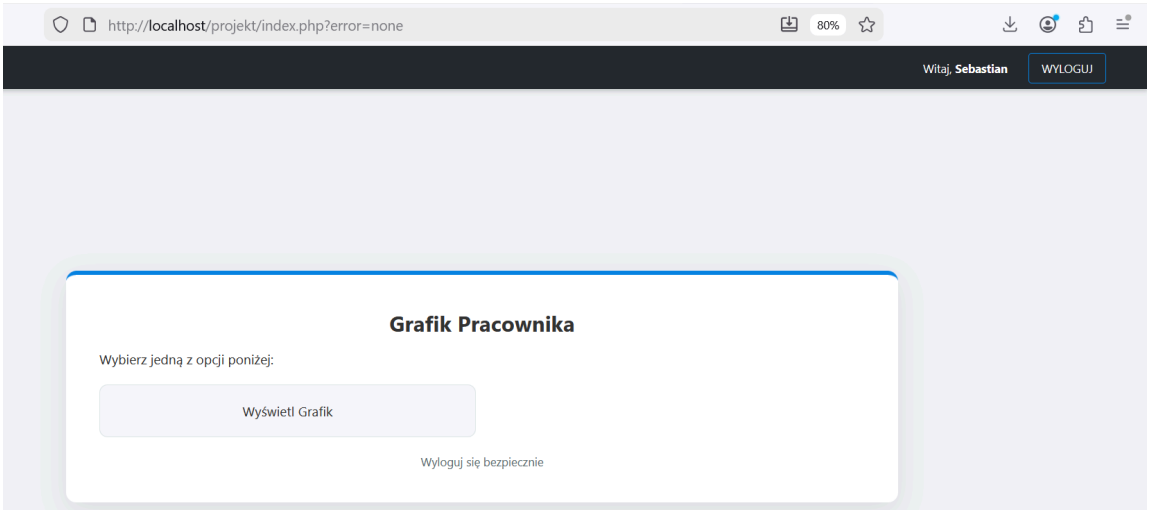
RYSUNEK 13. WIDOK LOGOWANIA/REJESTRACJI



Rysunek 14. Widok użytkownika



Rysunek 15. Widok administratora

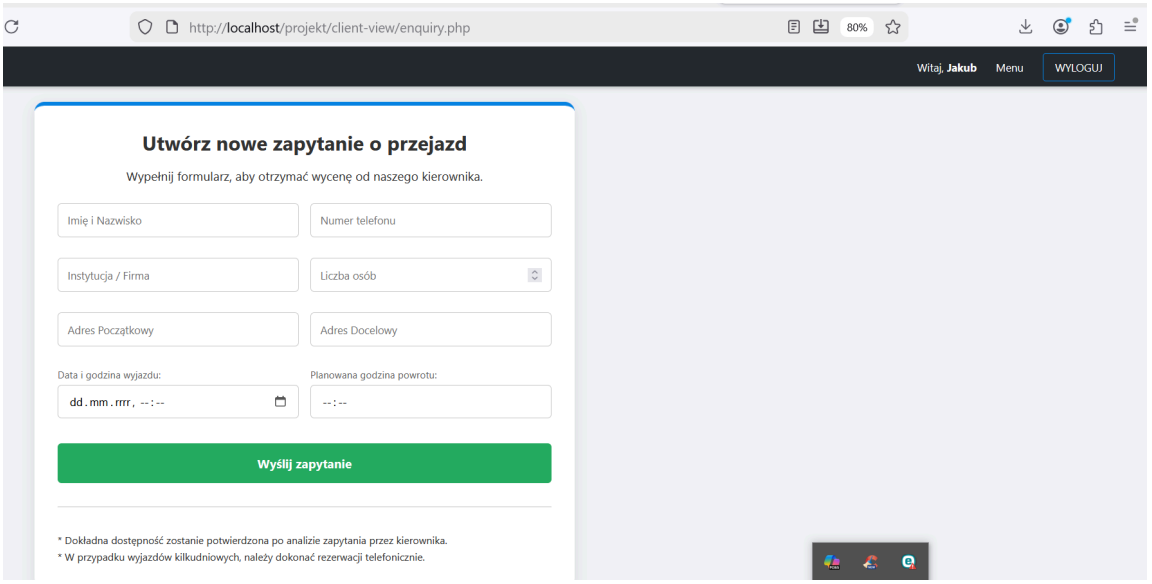


Rysunek 16. Widok pracownika

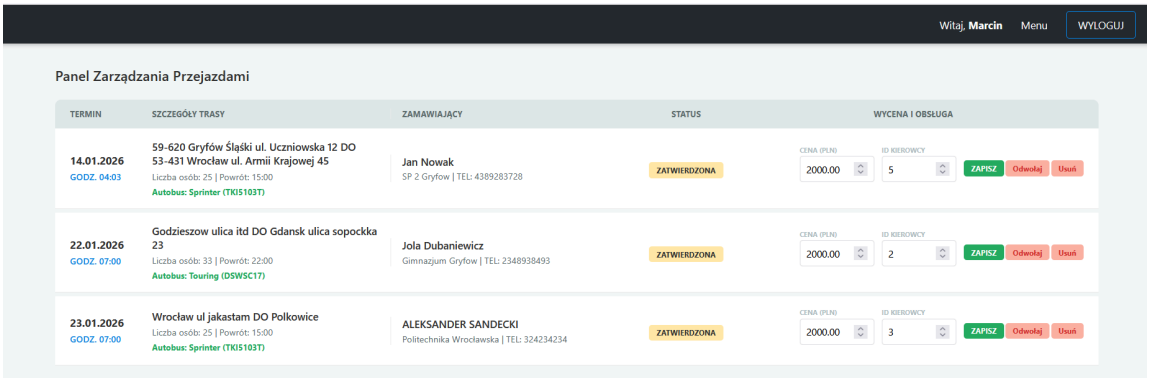
Poza podstawowymi panelami ról, w systemie zaimplementowano dedykowane widoki operacyjne, które stanowią o funkcjonalności aplikacji i pozwalają na pełną obsługę procesu logistycznego. Do najważniejszych z nich należą:

- Widok rezerwacji przejazdu: Jest to interaktywny formularz dla klienta, zaprojektowany z myślą o minimalizacji błędów wprowadzania danych. System automatycznie podpowiada bieżącą datę jako minimalny termin wyjazdu oraz wymusza podanie kluczowych parametrów, takich jak liczba osób czy trasa. Widok ten dynamicznie komunikuje użytkownikowi wynik operacji – od potwierdzenia wysłania zapytania po informację o braku dostępnych pojazdów w danym terminie.
- Widok zarządzania rezerwacjami (Agenda): Centralny punkt operacyjny dla kierownika, prezentujący wszystkie aktywne zapytania w formie przejrzystej listy. Widok ten integruje dane z wielu tabel, wyświetlając jednocześnie informacje o kliencie, szczegóły trasy oraz przypisany pojazd. Umożliwia on bezpośrednią interakcję, taką jak szybka wycena, przypisanie kierowcy do zlecenia czy usunięcie rezerwacji z systemu.
- Widok zarządzania pojazdami: Moduł ten pozwala na pełną kontrolę nad flotą autobusową. Przedstawiony w formie tabelarycznej, umożliwia kierownikowi nie tylko podgląd parametrów technicznych pojazdów (liczba miejsc, marka, model), ale także dynamiczne rozbudowywanie zasobów poprzez formularz dodawania nowych jednostek oraz usuwanie wycofanych z eksploatacji pojazdów.

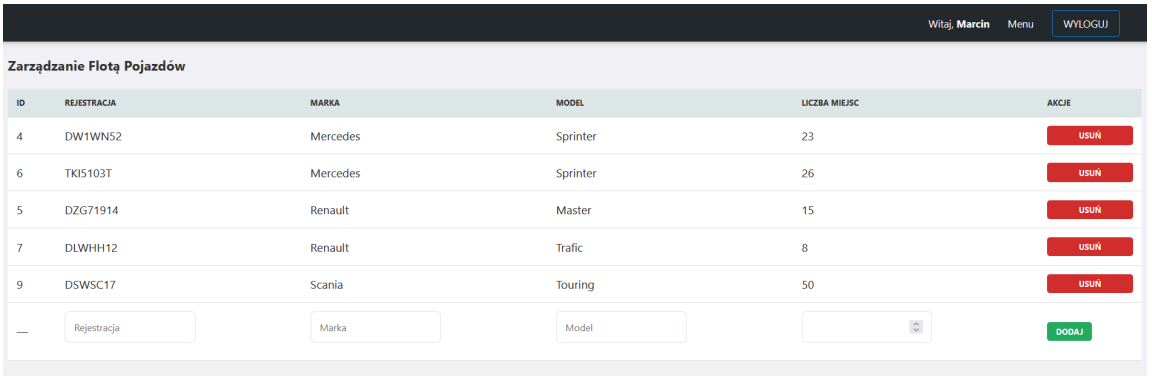
Wszystkie te widoki zostały zoptymalizowane pod kątem czytelności: zastosowano w nich intuicyjne oznaczenia kolorystyczne statusów (np. statusy rezerwacji) oraz przyciski akcji oparte na standardowych wzorcach UX, co znacząco skraca czas potrzebny na przeszkolenie personelu do obsługi systemu.



RYSUNEK 17. WIDOK TWORZENIA REZERWACJI



Rysunek 18. Widok Zarządzania Przejazdami



Rysunek 19. Widok Zarządzania Pojazdami

3.2.3. Projekt wybranych funkcji systemu

Logika biznesowa systemu opiera się na automatyzacji procesu dopasowania zasobów do potrzeb klienta oraz wielostopniowym obiegu dokumentacji rezerwacyjnej. Za realizację tych procesów odpowiadają wyspecjalizowane obiekty kontrolerów, które pośredniczą między warstwą prezentacji a bazą danych.

- Proces rezerwacji i wyceny: Głównym mechanizmem jest metoda `checkTravel` w klasie `Model`. System analizuje datę przejazdu oraz liczbę pasażerów, aby zweryfikować dostępność floty. Za ten proces po stronie logiki odpowiada obiekt `BookingContr`, który przyjmuje dane od klienta i inicjuje sprawdzenie dostępności. Każda rezerwacja przechodzi przez cykl życia zdefiniowany statusami:
    - "Oczekuje na wycenę": Status nadawany automatycznie przez obiekt `BookingContr` po pomyślnym wysłaniu zapytania przez klienta.
    - "Wyceniono": Stan po wprowadzeniu kwoty przez kierownika. Operację tę wykonuje obiekt `TravelContr`, który aktualizuje dane rezerwacji.

    - "Zatwierdzona": Finalny status po akceptacji ceny przez klienta (zarządzany przez `TravelContr`) lub nadawany automatycznie dla wybranych grup użytkowników.
- Automatyzacja wyboru zasobów: System dąży do optymalizacji kosztów poprzez automatyczne przypisanie najmniejszego dostępnego autobusu, który pomieści wskazaną liczbę osób. Realizuje to zapytanie SQL wewnątrz klasy `Model` z klauzulą `ORDER BY a.liczba_miejsc ASC LIMIT 1`. Obiektem wykonawczym wywołującym tę logikę podczas tworzenia rezerwacji jest `BookingContr`.
- Zarządzanie kadrą i flotą: Procesy te są zarządzane przez obiekty `WorkerContr` oraz `BusContr` i są zabezpieczone przed naruszeniem integralności danych. Przykładowo, metoda `deleteB` w klasie `Model`, wywoływana przez obiekt `BusContr`, wykorzystuje blok `try-catch`, aby zablokować usunięcie pojazdu przypisanego do istniejących rezerwacji, co pozwala na obsługę błędu klucza obcego o kodzie 23000. Podobnie obiekt `WorkerContr` zarządza usuwaniem pracowników, dbając o to, by operacja nie naruszyła powiązań z grafikami przejazdów.
- Zarządzanie dostępem i tożsamością: Obiekty `LoginContr` oraz `SignupContr` stanowią fundament bezpieczeństwa systemu. Odpowiadają one za prawidłowe tworzenie kont użytkowników (metoda `setUserFull`) oraz weryfikację haseł, co umożliwia późniejszą autoryzację działań w systemie na podstawie przypisanych ról.

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

Komunikacja z warstwą danych została odizolowana w dedykowanej klasie technicznej, co zapewnia wysoką reużywalność kodu.

- Wykorzystanie PDO (PHP Data Objects): Aplikacja wykorzystuje sterownik PDO, co pozwala na bezpieczne i wydajne zarządzanie połączeniami z bazą danych.
- Parametry połączenia: Konfiguracja środowiska (host: localhost, dbname: system, user: root) jest zaszyta w metodzie `connect()` klasy `Dbh`.
- Obsługa błędów: W celu ochrony informacji o infrastrukturze serwera, połączenie realizowane jest w bloku przechwytującym wyjątki:

  } catch (PDOException $e)

  {

  print "Błąd połączenia: " . $e->getMessage() . "
";

  die();

  }

3.2.5. Projekt zabezpieczeń na poziomie aplikacji

Mechanizmy bezpieczeństwa systemu

Bezpieczeństwo systemu zostało zrealizowane na kilku uzupełniających się poziomach, obejmujących ochronę przed atakami zewnętrznymi, zabezpieczenie danych użytkowników oraz kontrolę dostępu do zasobów aplikacji.

Ochrona przed atakami typu SQL Injection

Podstawowym mechanizmem zabezpieczającym aplikację przed atakami typu SQL Injection jest stosowanie zapytań przygotowanych (prepared statements) z wykorzystaniem interfejsu PDO. Zamiast dynamicznego wstawiania danych użytkownika bezpośrednio do zapytań SQL, wykorzystywane są symbole zastępcze, a wartości przekazywane są jako parametry, co eliminuje możliwość modyfikacji struktury zapytania.

Przykładowe użycie zapytania przygotowanego przedstawiono poniżej:

```
$stmt = $this->connect()->prepare(
    'SELECT * FROM rezerwacje WHERE id_rezerwacji = ?;'
);
```

```
$stmt->execute([$id]);
```

Takie podejście zapewnia poprawne rozdzielenie logiki zapytania od danych wejściowych użytkownika.

Bezpieczne przechowywanie haseł

System nie przechowuje haseł użytkowników w postaci jawnej. Podczas procesu rejestracji hasła są przetwarzane przy użyciu funkcji `password_hash()` z domyślnym algorytmem `PASSWORD_DEFAULT`. W trakcie logowania poprawność hasła jest weryfikowana za pomocą funkcji `password_verify()`. Zastosowanie mechanizmu haszowania zapewnia ochronę danych uwierzytelniających nawet w przypadku nieautoryzowanego dostępu do bazy danych.

Zarządzanie sesją i autoryzacja użytkowników

Dostęp do zasobów systemu oraz poszczególnych widoków aplikacji jest kontrolowany poprzez mechanizm sesji. Na początku każdego pliku procesowego oraz widoku wykonywana jest weryfikacja tożsamości użytkownika oraz jego roli w systemie. Brak odpowiednich uprawnień skutkuje przerwaniem dalszego przetwarzania żądania i przekierowaniem użytkownika na stronę główną aplikacji.

Przykładowa kontrola dostępu została przedstawiona poniżej:

```
if (!isset($_SESSION["userid"]) || $_SESSION["userrole"] !== "kierownik") {
    header("location: ../index.php?error=brak_autoryzacji");
    exit();
}
```

Walidacja danych wejściowych

W systemie zastosowano wielopoziomową walidację danych wejściowych. Na poziomie interfejsu użytkownika wykorzystano mechanizmy walidacji oferowane przez HTML5, takie jak atrybuty `required`, `type="email"` oraz ograniczenia wartości liczbowe (`min`). Dodatkowo przewidziano rozbudowę walidacji po stronie serwera, obejmującą m.in. sprawdzanie formatów numerów telefonów oraz danych osobowych, co zostało uwzględnione w dokumentacji projektowej.

4. Implementacja systemu

Etap implementacji polegał na przekształceniu założeń projektowych w działające rozwiązanie programistyczne. System został zrealizowany w architekturze opartej na języku PHP oraz relacyjnej bazie danych MySQL, co pozwoliło na stworzenie skalowalnego i bezpiecznego środowiska do zarządzania rezerwacjami.

4.1. Realizacja bazy danych

4.1. Realizacja bazy danych

Warstwa danych stanowi fundament aplikacji, odpowiadając za trwałe przechowywanie informacji o użytkownikach, pojazdach oraz procesach logistycznych. Wykorzystano silnik **InnoDB**, który zapewnia wsparcie dla transakcyjności oraz integralności referencyjnej poprzez klucze obce.

4.1.1. Tworzenie tabel i definiowanie ograniczeń

Struktura bazy danych składa się z pięciu powiązanych ze sobą tabel, z których każda pełni określoną rolę w systemie:

- **`konta`**: Tabela centralna dla modułu autentykacji, przechowująca loginy, zahashowane hasła oraz role użytkowników (`kierownik`, `pracownik`, `uzytkownik`).
- **`uzytkownicy`** i **`pracownicy`**: Tabele rozszerzające dane kont o informacje personalne i kontaktowe. Zastosowano tu relację jeden-do-jednego z tabelą `konta`, wymuszoną przez klucz obcy `id_konta`.
- **`autobusy`**: Katalog floty pojazdów, zawierający parametry techniczne takie jak numer rejestracyjny, marka, model oraz liczba dostępnych miejsc.
- **`rezerwacje`**: Najbardziej rozbudowana tabela, stanowiąca punkt styku wszystkich modułów. Przechowuje szczegóły trasy, daty, liczbę osób, statusy oraz wycenę.

Ograniczenia i integralność danych:

- Każda tabela posiada klucz podstawowy (`PRIMARY KEY`) z atrybutem `AUTO_INCREMENT`, co gwarantuje unikalność rekordów.
- Zdefiniowano wiązania `FOREIGN KEY`, które zapobiegają m.in. usunięciu konta, do którego przypisane są dane profilowe lub aktywne rezerwacje.
- Pola krytyczne, takie jak `login`, `email` czy `data_przejazdu`, posiadają restrykcję `NOT NULL`, co wymusza kompletność danych na poziomie bazy.

4.1.2. Implementacja mechanizmów przetwarzania danych

Przetwarzanie danych odbywa się za pośrednictwem klasy `Model`, która implementuje logikę CRUD (Create, Read, Update, Delete). Kluczowe mechanizmy obejmują:

- **Wyszukiwanie zasobów**: Zastosowano zaawansowane zapytania `SELECT` z podzapytaniami, np. w metodzie `checkTravel`, która weryfikuje dostępność autobusów poprzez wykluczenie pojazdów już zajętych w danym terminie.
- **Agregacja danych**: Widoki administracyjne korzystają ze złączeń tabel (`LEFT JOIN`), co pozwala na wyświetlenie w jednej liście danych rezerwacji wraz z informacjami o przypisanym autobusie i pracowniku.
- **Automatyzacja statusów**: Logika biznesowa w PHP steruje zmianami statusów w bazie, przechodząc od "Oczekuje na wycenę" do "Zatwierdzona" na podstawie interakcji użytkowników.

4.1.3. Implementacja uprawnień i innych zabezpieczeń

Zabezpieczenia na poziomie bazy danych i jej integracji z aplikacją zrealizowano poprzez:

- **Izolację dostępu (PDO)**: Połączenie z bazą odbywa się przez klasę `Dbh`, która wykorzystuje mechanizm `PDO`. Dzięki temu wszystkie parametry przesyłane do bazy są traktowane jako dane, co całkowicie eliminuje ryzyko ataków typu SQL Injection.
- **Integralność referencyjną**: Zdefiniowane w schemacie ograniczenia `ON DELETE RESTRICT` (domyślne dla kluczy obcych) chronią przed niespójnością danych. Przykładem jest blokada usunięcia pracownika lub autobusu, który figuruje w zatwierdzonym grafiku przejazdów.

- **Zarządzanie sesją**: Uprawnienia do wykonywania konkretnych operacji (np. `UPDATE` ceny) są weryfikowane po stronie aplikacji na podstawie roli zapisanej w tabeli `konta` podczas logowania, co zapobiega nieautoryzowanym zmianom w bazie danych.

4.2. Realizacja elementów aplikacji

W tej sekcji opisano techniczne szczegóły wdrożenia mechanizmów odpowiedzialnych za bezpieczeństwo danych oraz architekturę komunikacji między logiką aplikacji a serwerem bazy danych.

4.2.1. Walidacja i filtracja

System stosuje dwupoziomowy model weryfikacji danych, łączący walidację po stronie klienta z mechanizmami bezpieczeństwa po stronie serwera:

- **Walidacja po stronie interfejsu (Front-end):** Wykorzystuje wbudowane atrybuty HTML5 w formularzach (np. w pliku `projekt/index.php` oraz `projekt/client-view/enquiry.php`), takie jak `required` dla pól obowiązkowych, `type="email"` do weryfikacji formatu adresu poczty elektronicznej oraz `type="tel"` dla numerów kontaktowych. W formularzu rezerwacji zastosowano również atrybut `min` dla daty przejazdu, który dynamicznie blokuje możliwość wybrania daty wstecznej względem bieżącego czasu serwera.
- **Filtracja danych wejściowych:** Dane przesyłane metodą POST są przechwytywane przez skrypty w folderze `projekt/includes/` (np. `signup.inc.php`), a następnie przekazywane do odpowiednich kontrolerów.
- **Zabezpieczenie przed atakami SQL Injection:** Jest to kluczowy element filtracji na poziomie bazy danych. Zamiast bezpośredniego umieszczania zmiennych w zapytaniach SQL, system w klasie `Model` wykorzystuje parametryzowane zapytania (tzw. *placeholders*), co sprawia, że silnik bazy danych traktuje wprowadzone dane wyłącznie jako tekst, a nie jako komendy wykonywalne.
- **Hashowanie danych wrażliwych:** Hasła użytkowników są filtrowane przez algorytm `bcrypt` przy użyciu funkcji `password_hash()` w pliku `projekt/classes/model-classes.php`, co zapewnia, że nawet w przypadku wycieku bazy danych, oryginalne hasła pozostają niemożliwe do odczytania.

4.2.2. Implementacja interfejsu dostępu do bazy danych

Dostęp do bazy danych został zrealizowany w modelu obiektowym, z wyraźnym podziałem na warstwę techniczną i logiczną:

- Klasa łączności `Dbh`: Znajduje się w pliku `projekt/classes/dbh.classes.php` i pełni rolę interfejsu technicznego. Wykorzystuje bibliotekę PDO (PHP Data Objects) do nawiązywania bezpiecznych połączeń z bazą danych. Zastosowanie PDO pozwala na łatwą zmianę silnika bazy danych w przyszłości bez konieczności modyfikowania całej logiki aplikacji.
- Warstwa Modelu: Klasa `Model` (plik `projekt/classes/model-classes.php`) dziedziczy po klasie `Dbh`, dzięki czemu posiada bezpośredni dostęp do metody `connect()`. Implementuje ona konkretne zapytania SQL pogrupowane w chronione metody (np. `getUser`, `book`, `getEnquries`), które są następnie wykorzystywane przez kontrolery.
- Przykład implementacji zapytania:

```
protected function getEnquries() {

  // Przykład złączenia tabel (JOIN) w celu pobrania kompletnych danych

  $stmt = $this->connect()->prepare('SELECT r.*, a.rejestracja, a.model

      FROM rezerwacje r LEFT JOIN autobusy a ON r.id_autobusu =
a.id_autobusu

      WHERE r.status != "Odrzucono" ORDER BY r.data_przejazdu ASC;');

  $stmt->execute();

  return $stmt->fetchAll(PDO::FETCH_ASSOC);

}
```

Zalety rozwiązania: Taka architektura (Generalizacja/Dziedziczenie) eliminuje powielanie kodu odpowiedzialnego za połączenie, centralizuje mechanizmy obsługi błędów połączenia w jednym miejscu i pozwala na bezpieczne zarządzanie zasobami poprzez automatyczne zamykanie kursorów po wykonaniu operacji.

4.2.3. Implementacja wybranych funkcjonalności systemu

W tej sekcji przedstawiono sposób technicznej realizacji kluczowych procesów biznesowych, ze szczególnym uwzględnieniem automatyzacji i obiegu informacji.

- **Automatyczne dopasowanie zasobów (Algorytm wyboru pojazdu):** Sercem systemu rezerwacji jest mechanizm weryfikacji dostępności floty zaimplementowany w metodzie `checkTravel`. System wykonuje zapytanie SQL, które filtruje tabelę `autobusy`, poszukując pojazdów o odpowiedniej liczbie miejsc, a następnie wyklucza te jednostki, które posiadają już przypisane i nieodrzucone rezerwacje w wybranym dniu. Wynik jest sortowany rosnąco według liczby miejsc, co pozwala na automatyczny wybór najmniejszego, a tym samym najbardziej ekonomicznego pojazdu dla danego zlecenia.
- **Zarządzanie cyklem życia rezerwacji:** Implementacja opiera się na stanowej zmianie statusów w tabeli `rezerwacje`. Obiekt `BookingContr` inicjuje proces, ustawiając status „Oczekuje na wycenę”. Administrator, korzystając z widoku `display_enquries.php`, przesyła dane do skryptu `update-reservation.inc.php`, który za pośrednictwem metody `updateE` zmienia status na „Wyceniono” oraz przypisuje konkretną cenę i identyfikator kierowcy. Ostatnim etapem jest akceptacja klienta, realizowana przez skrypt `ack-price.inc.php`, który finalizuje rezerwację.
- **Dynamiczny interfejs użytkownika:** System implementuje logikę wyświetlania treści w zależności od roli zalogowanego użytkownika w pliku `index.php`. Wykorzystując instrukcje warunkowe PHP sprawdzające zmienną `$_SESSION["userrole"]`, aplikacja renderuje dedykowane panele: zarządzania dla kierownika, panelu zapytań dla klienta lub grafiku dla pracownika.

4.2.4. Implementacja mechanizmów bezpieczeństwa

Bezpieczeństwo aplikacji zostało zintegrowane bezpośrednio z architekturą dostępu do danych oraz mechanizmami obsługi sesji.

- **Ochrona przed SQL Injection (Prepared Statements):** Najważniejszym mechanizmem obronnym jest konsekwentne stosowanie zapytań przygotowanych we wszystkich metodach klasy `Model`. Dzięki użyciu metody `prepare()` i przekazywaniu zmiennych w tablicy do metody `execute()`, dane użytkownika są oddzielone od instrukcji sterujących SQL, co uniemożliwia manipulację zapytaniami.
- **Autoryzacja i kontrola dostępu do zasobów:** Bezpieczeństwo na poziomie plików widoków zrealizowano poprzez rygorystyczne sprawdzanie sesji na początku każdego skryptu. Przykładem jest plik `manage-buses.php`, który weryfikuje nie tylko fakt bycia zalogowanym, ale również konkretną rolę „kierownik". W przypadku braku uprawnień następuje natychmiastowe przerwanie skryptu i przekierowanie użytkownika.
- **Szyfrowanie haseł:** W procesie tworzenia konta (metoda `setUserFull`), hasło wprowadzone przez użytkownika jest poddawane jednostronnemu haszowaniu przy użyciu funkcji `password_hash` z algorytmem `PASSWORD_DEFAULT`. Podczas logowania, system korzysta z `password_verify`, aby porównać wpisany tekst z hashem zapisanym w bazie danych, co chroni dane dostępowe nawet w razie nieautoryzowanego wglądu w tabelę `konta`.
- **Ochrona integralności referencyjnej:** Aplikacja implementuje zabezpieczenia przed błędami logicznymi w danych. Metoda `deleteW` (usuwanie pracownika) oraz `deleteB` (usuwanie autobusu) są objęte blokami `try-catch`. Mechanizm ten przechwytuje wyjątki PDO rzucane przez bazę danych w sytuacji naruszenia kluczy obcych (np. próba usunięcia pojazdu przypisanego do trasy), co zapobiega powstawaniu tzw. „sierocych rekordów" i wyświetla czytelny komunikat o błędzie zamiast błędu systemowego.

5. Testowanie systemu

Proces testowania miał na celu weryfikację poprawności działania wszystkich zaimplementowanych modułów, sprawdzenie stabilności połączenia z bazą danych oraz potwierdzenie skuteczności mechanizmów bezpieczeństwa.

5.1. Instalacja i konfigurowanie systemu

Instalacja systemu wymaga przygotowania środowiska serwerowego opartego na stosie technologicznym Apache, MySQL/MariaDB oraz PHP (XAMPP).

- **Import bazy danych**: Proces rozpoczyna się od utworzenia bazy danych o nazwie `system` i zaimportowania schematu SQL.
- **Konfiguracja połączenia**: Należy upewnić się, że parametry w klasie Dbh odpowiadają lokalnym ustawieniom serwera (domyślnie: localhost, root, brak hasła).
- **Zależności**: System wykorzystuje menedżera pakietów Composer do obsługi biblioteki PHPUnit, co wymaga wykonania komendy `composer install` w celu pobrania niezbędnych zależności do testów.

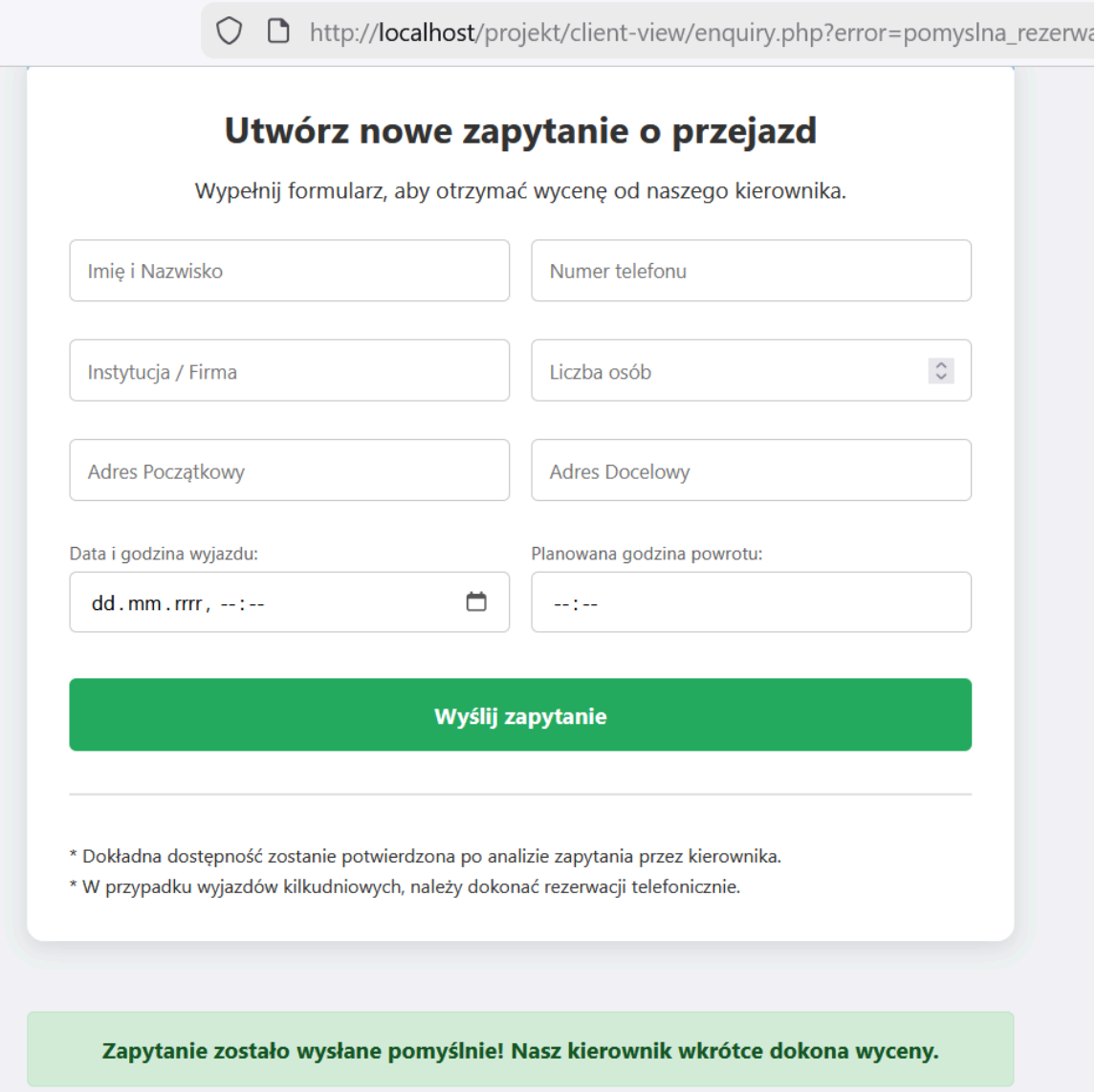
5.2. Testowanie opracowanych funkcji systemu

Testy funkcjonalne przeprowadzono metodą czarnoszynową, weryfikując reakcje systemu na poprawne oraz błędne dane wejściowe.

5.2.1. Testowanie funkcji: Rezerwacja przejazdu (klient)

- **Scenariusz**: Użytkownik wypełnia formularz zapytania o przejazd, podając liczbę osób i termin.

- **Oczekiwany wynik**: System automatycznie sprawdza dostępność pojazdu metodą `checkTravel`. Jeśli pojazd o odpowiedniej liczbie miejsc jest wolny, rezerwacja zostaje dodana ze statusem „Oczekuje na wycenę”.
- **Rezultat**: Test zakończony pomyślnie.

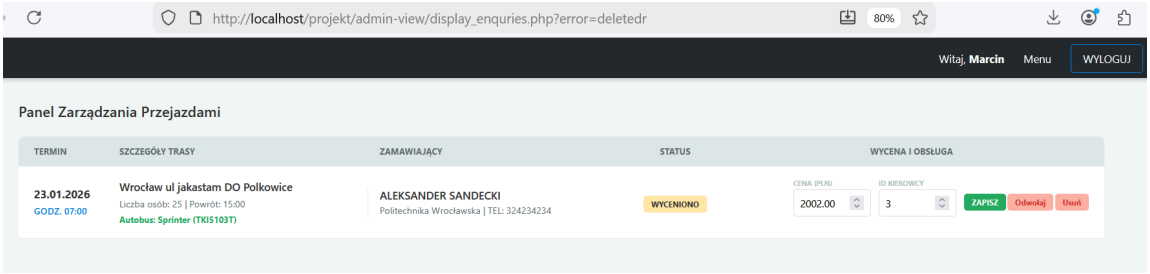


Rysunek 19. Widok potwierdzenia wysłania zapytania

5.2.2. Testowanie funkcji: Zarządzanie agendą (administrator)

- **Scenariusz**: Kierownik dokonuje wyceny rezerwacji i przypisuje do niej pracownika.
- **Oczekiwany wynik**: Metoda `updateE` zmienia status rekordu w bazie danych na „Wyceniono”, a klient widzi nową cenę w swoim panelu.

- **Rezultat**: Dane są poprawnie aktualizowane i wyświetlane w widoku agendy.

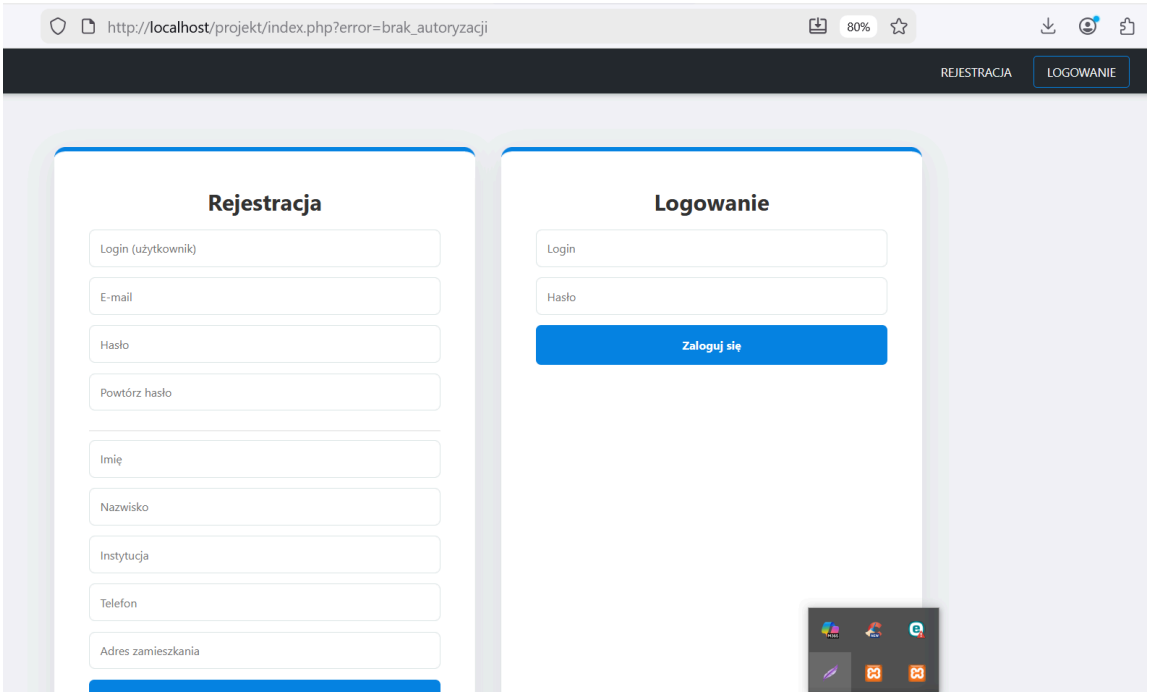


Rysunek 20. Panel administratora po dokonaniu wyceny i przypisaniu kierowcy

5.3. Testowanie mechanizmów bezpieczeństwa

Weryfikacja bezpieczeństwa skupiła się na trzech kluczowych aspektach:

- **Autoryzacja ról**: Próba wejścia na adres `admin-view/manage-workers.php` przez niezalogowanego użytkownika kończy się natychmiastowym przekierowaniem do strony głównej z błędem `brak_autoryzacji`.



Rysunek 21. Komunikat o braku uprawnień przy próbie nieautoryzowanego dostępu

5.4. Inne testy

W ramach testów automatycznych przygotowano plik `ProjectTest.php` wykorzystujący framework PHPUnit. Testy te pozwalają na szybką weryfikację logiczną metod kontrolerów bez konieczności manualnego wypełniania formularzy w przeglądarce. Dodatkowo sprawdzono poprawność wyświetlania grafiku kierowcy, upewniając się, że pracownik widzi tylko te przejazdy, które zostały mu przypisane i mają status „Zatwierdzona".

```
PS C:\xampp\htdocs\projekt> ./vendor/bin/phpunit tests/ProjectTest.php
PHPUnit 11.5.46 by Sebastian Bergmann and contributors.

Runtime:       PHP 8.2.12

...............                                                 15 / 15 (100%)

Time: 00:00.029, Memory: 8.00 MB

OK (15 tests, 16 assertions)
PS C:\xampp\htdocs\projekt>
```

Rysunek 22. Wynik wykonania testów jednostkowych w konsoli systemowej

5.5. Wnioski z testów

Przeprowadzone testy wykazały, że system jest stabilny i poprawnie realizuje założone cele biznesowe. Wszystkie kluczowe funkcjonalności, takie jak automatyczne dopasowanie autobusu, obieg statusów rezerwacji oraz segregacja uprawnień, działają zgodnie z projektem. Mechanizmy bezpieczeństwa skutecznie blokują próby nieautoryzowanego dostępu oraz chronią bazę danych przed wprowadzeniem błędnych powiązań. System jest gotowy do wdrożenia w środowisku produkcyjnym.

6. Podsumowanie

Celem niniejszego projektu było zaprojektowanie oraz implementacja systemu informatycznego wspierającego obsługę rezerwacji przejazdów autokarowych, z wykorzystaniem relacyjnej bazy danych oraz aplikacji webowej działającej w architekturze klient–serwer. Założony cel został osiągnięty poprzez realizację kompletnego procesu projektowego, obejmującego analizę wymagań, projekt bazy danych, projekt aplikacji użytkownika, implementację oraz testowanie systemu.

W ramach projektu opracowano spójną i znormalizowaną strukturę bazy danych, zapewniającą integralność referencyjną oraz poprawność przechowywanych informacji. Zastosowanie kluczy głównych i obcych, odpowiednich typów danych oraz ograniczeń integralności pozwoliło na wyeliminowanie redundancji danych oraz zabezpieczenie systemu przed niespójnościami logicznymi.

Równolegle zaprojektowano i zaimplementowano aplikację webową w języku PHP, opartą na modularnej architekturze zbliżonej do wzorca MVC. Wyraźny podział odpowiedzialności pomiędzy warstwę prezentacji, logikę aplikacyjną oraz warstwę dostępu do danych umożliwił czytelną organizację kodu oraz ułatwił dalszy rozwój systemu. Zastosowanie relacji generalizacji w diagramie klas pozwoliło na centralizację mechanizmów komunikacji z bazą danych i ograniczenie powielania kodu.

Szczególną uwagę poświęcono aspektom bezpieczeństwa systemu. Zaimplementowano mechanizmy zabezpieczające przed atakami typu SQL Injection poprzez stosowanie zapytań parametryzowanych, bezpieczne przechowywanie haseł użytkowników z wykorzystaniem funkcji haszujących oraz kontrolę dostępu opartą na rolach użytkowników. Dodatkowo integralność danych została wzmocniona zarówno na poziomie bazy danych, jak i warstwy aplikacyjnej.

Przeprowadzone testy funkcjonalne oraz testy mechanizmów bezpieczeństwa potwierdziły poprawność działania systemu oraz zgodność implementacji z założeniami projektowymi. System prawidłowo realizuje proces rezerwacji przejazdów, automatyczne dopasowanie zasobów oraz obsługę różnych ról użytkowników.

Zrealizowany projekt spełnia wymagania stawiane systemom informatycznym o charakterze edukacyjnym i może stanowić solidną podstawę do dalszej rozbudowy, w szczególności o zaawansowane mechanizmy raportowania, optymalizacji wydajności oraz wdrożenia w środowisku produkcyjnym.

Literatura

[1] Mirosław Zelent, Pasja informatyki: Kurs PHP odc. 1-6
https://www.youtube.com/watch?v=tD0Q5QwoQJI
[2] Dani Krossing “How To Create A OOP PHP Login System For Beginners | OOP PHP & PDO | OOP PHP Tutorial” https://youtu.be/BaEm2Qv14oU?si=epbyCVxBD9dRZ0GG