WYŻSZA SZKOŁA
INFORMATYKI i ZARZĄDZANIA
z siedzibą w Rzeszowie

KOLEGIUM INFORMATYKI STOSOWANEJ

Kierunek: INFORMATYKA

Specjalność: brak

Paweł Drąg
Nr albumu studenta 72448

System telefoniczny

Promotor: mgr inż. Ewa Żesławska

PRACA ZALICZENIOWA

Rzeszów 2026

.

Spis treści

Wstęp **5**

1 Wymagania i specyfikacja techniczna **6**
1.1 Wymagania funkcjonalne 6
1.2 Wymagania niefunkcjonalne 6
1.3 Charakterystyka wykorzystanych technologii 7
1.4 Wymagania sprzętowe 8

2 Architektura systemu **9**
2.1 Warstwy aplikacji 9
2.2 Struktura klas i hierarchia 9

3 Struktura Bazy Danych **11**
3.1 Diagram związków encji (ERD) 11
3.2 Opis tabel i kolumn 12
3.3 Relacje 15
3.4 Kod definiujący strukturę (DDL) 16
3.5 Weryfikacja operacji na danych (SQL DML) 17
3.6 Podsumowanie wdrożenia bazy danych 19

4 Implementacja kluczowych modułów **20**
4.1 Struktura fizyczna projektu 20
4.2 Implementacja logiki biznesowej 21
4.3 Bezpieczeństwo i Autoryzacja 23
4.4 Testy oprogramowania 24

5 Opis kluczowych funkcjonalności **30**
5.1 Moduł Administratora 30
5.2 Moduł Użytkownika 32

6 Harmonogram realizacji projektu **34**
6.1 Szczegółowy przebieg prac 34

7 Podsumowanie i wnioski **36**

Bibliografia **38**

Spis rysunków **39**

Spis tablic **40**

Streszczenie **40**

.

Wstęp

Celem niniejszej pracy jest zaprojektowanie i implementacja aplikacji webowej o nazwie „SystemTelefonicznyGY”. System ten służy do kompleksowego zarządzania zasobami telekomunikacyjnymi w przedsiębiorstwie. Aplikacja umożliwia ewidencję pracowników, odwzorowanie struktury firmy (działy i stanowiska), zarządzanie urządzeniami (telefony, modemy) oraz zaawansowaną analizę kosztów połączeń telefonicznych na podstawie importowanych bilingów od operatorów telefoni komórkowej jak i stacjonarnej.

Projekt został zrealizowany w architekturze MVC (Model-View-Controller), co zapewnia przejrzystą separację logiki biznesowej od warstwy prezentacji, ułatwiając tym samym dalszy rozwój i utrzymanie oprogramowania.

Założenia projektowe i wymagania

Główne założenia systemu obejmują realizację następujących procesów biznesowych:

- **Zarządzanie użytkownikami:** Rejestracja pracowników, edycja danych osobowych, przydzielanie ról (Administrator/User) oraz zarządzanie hasłami.
- **Zarządzanie zasobami:** Pełna ewidencja telefonów komórkowych, stacjonarnych oraz modemów, wraz z historią przypisań do pracowników.
- **Obsługa bilingów:** Import plików CSV z bilingami dostarczanymi przez operatora telekomunikacyjnego. System automatycznie parsuje pliki i przypisuje koszty do konkretnych pracowników na podstawie numerów telefonów.
- **Raportowanie:** Generowanie raportów kosztowych z możliwością filtrowania danych po zakresie dat, działach, managerach czy numerach faktur. Możliwość eksportu wyników do plików CSV.
- **Panel pracownika:** Dedykowany widok dla zalogowanego użytkownika, umożliwiający podgląd własnych urządzeń i kosztów wygenerowanych za ostatni miesiąc.
- **Widok ogólny:** Umożliwia przeglądanie orza przeszukiwanie książki telefonicznej bez konieczności logowania się do systemu.

.

Rozdział 1

Wymagania i specyfikacja techniczna

Celem niniejszego rozdziału jest zdefiniowanie wymagań funkcjonalnych i niefunkcjonalnych stawianych aplikacji „SystemTelefonicznyGY” oraz charakterystyka środowiska programistycznego i sprzętowego wykorzystanego w procesie wytwórczym.

1.1 Wymagania funkcjonalne

Na podstawie analizy potrzeb przedsiębiorstwa wyodrębniono dwie główne role użytkowników: Administratora oraz Pracownika. Wymagania funkcjonalne systemu zostały podzielone według tych grup.

Panel Administratora

Administrator posiada pełne uprawnienia do zarządzania systemem. Do kluczowych funkcji należą:

- **Autoryzacja i bezpieczeństwo:** Logowanie do panelu administracyjnego chronione sesją.
- **Zarządzanie strukturą:** Edycja działów i stanowisk oraz przypisywanie do nich pracowników.
- **Zarządzanie zasobami:** Ewidencja numerów telefonów (komórkowych i stacjonarnych) oraz urządzeń fizycznych.
- **Obsługa bilingów:** Import plików CSV, automatyczne wykrywanie duplikatów faktur oraz parsowanie niestandardowych formatów danych.
- **Raportowanie:** Generowanie zestawień kosztów i ich eksport do plików CSV.

Panel Użytkownika

Pracownik posiada dostęp do indywidualnego konta, co umożliwia:

- Wgląd w przypisane urządzenia i numery telefonów.
- Weryfikację generowanych kosztów (bilingów) dla własnego numeru.

1.2 Wymagania niefunkcjonalne

Wymagania niefunkcjonalne określają jakość działania systemu oraz ograniczenia techniczne:

- **Bezpieczeństwo i Autoryzacja:**

  - System musi weryfikować tożsamość użytkownika na podstawie unikalnej pary login oraz hasło.
  - Dostęp do funkcji administracyjnych oraz edycji danych musi być chroniony mechanizmem sesji (Session-based Authentication), uniemożliwiającym dostęp osobom nieuprawnionym (np. poprzez bezpośrednie wpisanie adresu URL).
  - Aplikacja powinna implementować zabezpieczenia przed atakami typu SQL Injection poprzez separację danych od zapytań (wykorzystanie parametrów).

- **Wydajność:** System musi być zoptymalizowany pod kątem przetwarzania dużych zbiorów danych tekstowych. Import plików bilingowych (CSV) zawierających do kilku tysięcy rekordów powinien odbywać się w czasie akceptowalnym dla użytkownika (szacunkowo poniżej 30 sekund), nie powodując „zamrożenia" interfejsu przeglądarki.

- **Responsywność i Dostępność:** Interfejs użytkownika (UI) musi być wykonany w technice RWD (Responsive Web Design), co zapewni poprawną czytelność i obsługę systemu na urządzeniach o różnej rozdzielczości ekranu (monitory, laptopy, tablety).

1.3 Charakterystyka wykorzystanych technologii

Projekt został zrealizowany przy użyciu nowoczesnych narzędzi z ekosystemu Microsoft, dobranych w celu zapewnienia stabilności i łatwości utrzymania kodu.

Język programowania i logika aplikacji

Głównym językiem programowania wykorzystanym w projekcie jest **C#** (wersja 7.3/8.0). Logika aplikacji opiera się na wzorcu architektonicznym **ASP.NET MVC 5** (.NET Framework 4.8), który narzuca podział aplikacji na trzy warstwy: Model (dane), Widok (prezentacja) i Kontroler (logika biznesowa). Zastosowanie tego wzorca ułatwia testowanie i rozbudowę systemu.

Baza danych i warstwa dostępu

Do przechowywania danych wykorzystano relacyjną bazę danych **Microsoft SQL Server**. Komunikacja z bazą odbywa się za pośrednictwem technologii **ADO.NET**. W projekcie zaimplementowano autorską klasę `BazaDanych`, która obsługuje zapytania SQL oraz parametryzację (`SqlParameter`), co chroni system przed atakami typu SQL Injection i zapewnia wysoką wydajność przy przetwarzaniu dużych zbiorów danych (import bilingów).

Warstwa prezentacji (Front-end)

Interfejs użytkownika został zbudowany przy użyciu standardowych technologii webowych:

- **HTML5** – do strukturyzacji treści stron.

- **CSS3** oraz framework **Bootstrap 5.2.3** – do stylizacji i zapewnienia responsywności (RWD).

- **Razor** – silnika widoków pozwalającego na dynamiczne osadzanie kodu C# wewnątrz plików .cshtml.

Narzędzia i testowanie

Całość prac implementacyjnych przeprowadzono w środowisku **Microsoft Visual Studio 2022**. Do zarządzania kodem źródłowym wykorzystano system kontroli wersji **Git** (platforma GitHub).

W celu zapewnienia jakości kodu wdrożono testy jednostkowe z wykorzystaniem frameworka **MSTest**. Do izolacji testowanych komponentów (np. kontrolerów) użyto biblioteki **Moq**, co pozwoliło na symulację zachowania bazy danych i sesji użytkownika.

1.4 Wymagania sprzętowe

Aplikacja została zaprojektowana jako rozwiązanie internetowe (Web Application) w architekturze Klient-Serwer.

Wymagania po stronie serwera:

- System operacyjny: Windows Server lub Windows 10/11 (dla środowiska testowego).
- Serwer WWW: IIS (Internet Information Services) z obsługą .NET Framework 4.8.
- Baza danych: SQL Server 2019 Express lub nowszy.

Wymagania po stronie klienta: Użytkownicy końcowi korzystają z systemu za pośrednictwem przeglądarki internetowej (np. Google Chrome, Microsoft Edge, Firefox). Nie jest wymagana instalacja dodatkowego oprogramowania na stacjach roboczych.

.

Rozdział 2

Architektura systemu

Aplikacja została zaprojektowana zgodnie z wzorcem projektowym **MVC (Model-View-Controller)**, z dodatkowym wydzieleniem warstwy logiki biznesowej (Service Layer). Takie podejście pozwala na zachowanie czystości kodu i ułatwia testowanie.

2.1 Warstwy aplikacji

Warstwa Danych (Data Layer) Odpowiada za bezpośrednią komunikację z bazą danych SQL. Jest realizowana poprzez klasę `BazaDanych`, która implementuje interfejs `IBazaDanych`. Warstwa ta ukrywa szczegóły zapytań SQL przed resztą systemu.

Warstwa Logiki Biznesowej (Service Layer) Zawiera całą logikę operacyjną aplikacji, odseparowaną od kontrolerów. Jest to warstwa niezależna od interfejsu użytkownika. Kluczowe serwisy to:

- `PracownikService`: Logika logowania, walidacji i zarządzania personelem.
- `BilingService`: Logika importu plików CSV, przetwarzania danych bilingowych i generowania raportów.
- `ZasobyService`: Zarządzanie sprzętem (przypisywanie do pracownika, stan urządzenia).

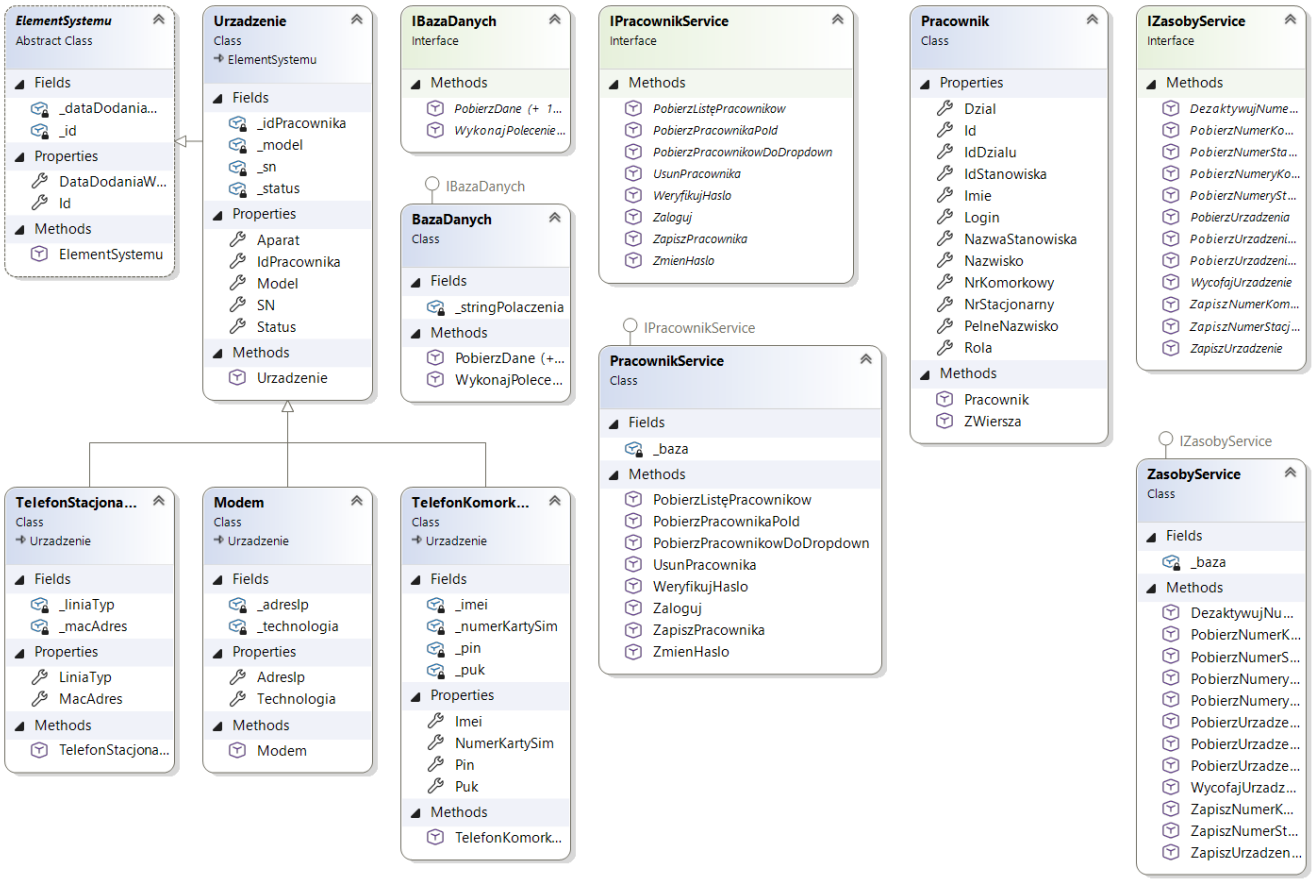
Warstwa Prezentacji (View) Odpowiada za interfejs użytkownika. Została zrealizowana przy użyciu silnika Razor (.cshtml), HTML5 oraz biblioteki Bootstrap. Komunikuje się wyłącznie z Kontrolerami.

Kontrolery (Controllers) Pośredniczy między użytkownikiem a systemem (logiką).Kontrolery (np. `Administ`
odbierają żądania HTTP, wywołują odpowiednie metody z warstwy serwisowej, a następnie zwracają odpowiedni widok.

2.2 Struktura klas i hierarchia

W projekcie zastosowano programowanie obiektowe z wykorzystaniem dziedziczenia i polimorfizmu. Wymóg dotyczący złożoności struktury obiektowej został spełniony – zaprojektowana hierarchia klas składa się z 5 elementów, co pozwoliło na odwzorowanie relacji zachodzących w przedstawionym systemie.

Poniższy diagram klas (Rysunek 2.1) prezentuje relacje między modelami w systemie, w tym dziedziczenie po klasie abstrakcyjnej `ElementSystemu` oraz wykorzystanie interfejsów.



Rysunek 2.1: Diagram klas projektu SystemTelefonicznyGY. Źródło: Opracowanie własne.

Szczegółowa struktura klas prezentuje się następująco::

1. **`ElementSystemu`** (Klasa abstrakcyjna bazowa) – Zawiera wspólne cechy, takie jak unikalne ID oraz data dodania wpisu.

2. **`Urzadzenie`** (Dziedziczy po `ElementSystemu`) – Reprezentuje fizyczny sprzęt. Zawiera pola: Model, SN (Numer Seryjny), Status.

3. **`TelefonKomorkowy`** (Dziedziczy po `Urzadzenie`) – Rozszerza klasę o IMEI, kartę SIM, PIN, PUK.

4. **`TelefonStacjonarny`** (Dziedziczy po `Urzadzenie`) – Rozszerza klasę o adres MAC i typ linii.

5. **`Modem`** (Dziedziczy po `Urzadzenie`) – Klasa reprezentująca urządzenia sieciowe, rozszerzająca model bazowy o atrybuty specyficzne, takie jak adres IP oraz technologia transmisji. Wyodrębnienie tego typu umożliwia w przyszłości implementację dedykowanej logiki sterowania modemami bez ingerencji w ogólną strukturę systemu.

Zastosowana hierarchia klas (Polimorfizm) zapewnia modularność rozwiązania. System został zaprojektowany w sposób otwarty na rozbudowę, co pozwala na łatwe definiowanie nowych typów sprzętu (np. drukarek, routerów) poprzez dodawanie kolejnych klas dziedziczących po `Urzadzenie`, bez konieczności modyfikacji istniejącego kodu obsługującego logikę ogólną.

Dodatkowo zaimplementowano interfejsy (np. `IPracownikService`), co separuje warstwę abstrakcji od implementacji i umożliwia wstrzykiwanie zależności (Dependency Injection) w kontrolerach.

.

Rozdział 3

Struktura Bazy Danych

Baza danych stanowi centralny magazyn informacji dla aplikacji zarządzającej zasobami telekomunikacyjnymi w przedsiębiorstwie. Celem niniejszego projektu było zaprojektowanie i wdrożenie relacyjnej bazy danych dla systemu „SystemTelefonicznyGY”.

Założenia i wymagania

Projekt bazy danych został opracowany w oparciu o następujące założenia:

- **Spójność danych:** Zastosowanie kluczy obcych (Foreign Keys) w celu zapewnienia integralności referencyjnej między pracownikami, działami, a przypisanym sprzętem.
- **Normalizacja i optymalizacja:** Struktura tabel została zaprojektowana w oparciu o zasady normalizacji (dążenie do 3NF). W uzasadnionych przypadkach (tabela `Pracownicy`) zastosowano jednak świadomą **denormalizację** (przechowywanie `ID_Dzialu` bezpośrednio przy pracowniku). Zabieg ten ma na celu uproszczenie zapytań SQL (uniknięcie wielokrotnych złączeń tabel przy podstawowych operacjach) oraz zwiększenie wydajności odczytu danych.
- **Elastyczność modelu sprzętowego:** Zastosowanie strategii *Table-Per-Hierarchy* dla tabeli `Urzadzenia`, co pozwala na przechowywanie różnych typów sprzętu (telefony, modemy) w jednej strukturze z kolumną rozróżniającą (Discriminator).
- **Bezpieczeństwo:** Hasła użytkowników przechowywane są w formacie tekstowym (w wersji deweloperskiej) z możliwością łatwej migracji na skróty kryptograficzne.

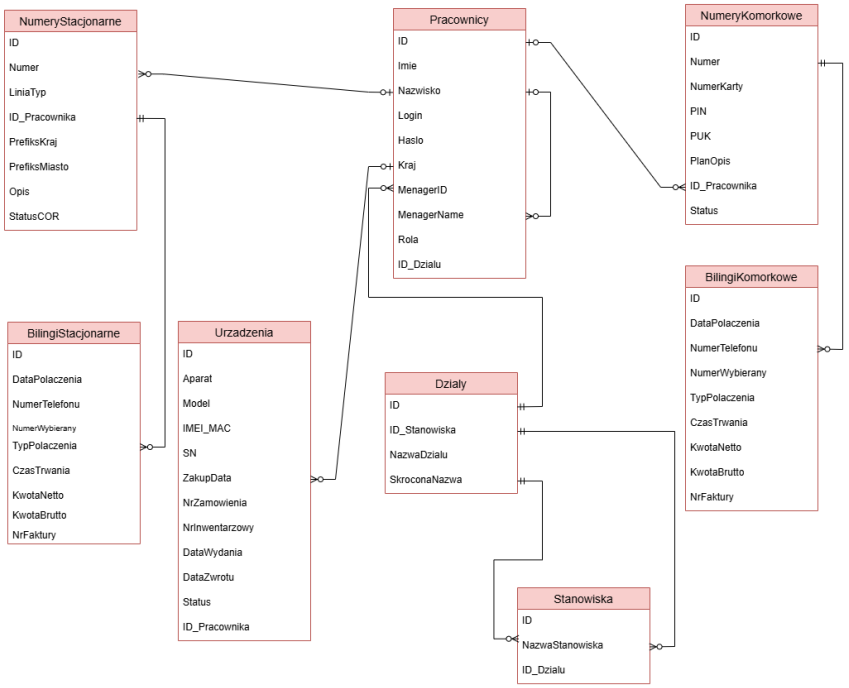
Środowisko implementacji

Baza danych została zaimplementowana w systemie zarządzania bazą danych (RDBMS) **Microsoft SQL Server**. Do komunikacji z bazą wykorzystywany jest język T-SQL (Transact-SQL).

Struktura bazy danych została zaprojektowana w sposób znormalizowany i składa się z 8 powiązanych ze sobą tabel. Rozdzielono logikę obsługi telefonii komórkowej i stacjonarnej, co zwiększa wydajność i przejrzystość systemu.

3.1 Diagram związków encji (ERD)

Poniższy diagram (Rysunek 3.1) prezentuje encje występujące w systemie oraz zachodzące między nimi relacje.



Rysunek 3.1: Diagram ERD bazy danych SystemTelefonicznyGY. Źródło: Opracowanie własne.

3.2 Opis tabel i kolumn

Poniżej przedstawiono szczegółową specyfikację wszystkich tabel wchodzących w skład systemu, wraz z opisem przechowywanych atrybutów oraz ich typów danych.

Słowniki i Struktura Organizacyjna

Tabela Dzialy służy do odwzorowania struktury organizacyjnej przedsiębiorstwa. Jest słownikiem wykorzystywanym do grupowania pracowników i stanowisk, co umożliwia późniejszą agregację kosztów (MPK - Miejsca Powstawania Kosztów).

Tabela 3.1: Tabela: Dzialy

Kolumna	**Typ**	**Opis**
ID	int (PK)	Klucz główny (Identity).
NazwaDzialu	nvarchar(100)	Pełna nazwa departamentu/działu.
SkroconaNazwa	nvarchar(10)	Kod działu (używany w raportach).

Tabela Stanowiska definiuje role pełnione przez pracowników w firmie. Każde stanowisko jest ściśle powiązane z konkretnym działem, co wymusza poprawność struktury podległości.

Tabela 3.2: Tabela: Stanowiska

Kolumna	Typ	Opis
ID	int (PK)	Klucz główny (Identity).
NazwaStanowiska	nvarchar(100)	Nazwa stanowiska służbowego.
ID_Dzialu	int (FK)	Klucz obcy wskazujący na Dział.

Zasoby Ludzkie

Tabela Pracownicy jest centralnym elementem systemu. Przechowuje dane osobowe, dane logowania oraz informacje o miejscu w hierarchii firmy. Zastosowano tu relację zwrotną (rekurencję) do obsługi struktury przełożony-podwładny.

Tabela 3.3: Tabela: Pracownicy

Kolumna	Typ	Opis
ID	int (PK)	Unikalny identyfikator pracownika.
Imie	nvarchar(50)	Imię pracownika.
Nazwisko	nvarchar(50)	Nazwisko pracownika.
Login	nvarchar(50)	Login do systemu (Unikalny).
Haslo	nvarchar(50)	Hasło użytkownika.
Kraj	nvarchar(50)	Lokalizacja pracownika.
MenagerID	int (FK)	ID przełożonego (relacja do tej samej tabeli).
MenagerName	nvarchar(100)	Imię i nazwisko przełożonego (pole pomocnicze).
Rola	nvarchar(20)	Uprawnienia systemowe (np. 'Admin', 'User').
ID_Dzialu	int (FK)	Przypisanie do działu.
ID_Stanowiska	int (FK)	Przypisanie do stanowiska.

Zasoby Sprzętowe i Numery

Tabela Urzadzenia pełni rolę inwentarza sprzętu fizycznego. Przechowuje informacje o telefonach, modemach i innym sprzęcie wydanym pracownikom. Wykorzystuje kolumnę `Discriminator` do obsługi dziedziczenia w kodzie aplikacji.

Tabela 3.4: Tabela: Urzadzenia

Kolumna	Typ	Opis
ID	int (PK)	Klucz główny.
Aparat	nvarchar(50)	Producent urządzenia (np. Samsung, Apple).
Model	nvarchar(50)	Model urządzenia.
IMEI_MAC	nvarchar(50)	Unikalny numer identyfikacyjny (IMEI lub MAC).
SN	nvarchar(50)	Numer seryjny producenta.
ZakupData	date	Data zakupu sprzętu.
NrZamowienia	nvarchar(50)	Numer zamówienia/faktury zakupu.
NrInwentarzowy	nvarchar(50)	Wewnętrzny numer środka trwałego.
DataWydania	date	Data przekazania pracownikowi.
DataZwrotu	date	Data zwrotu do magazynu.
Status	nvarchar(20)	Status (np. 'W użyciu', 'Magazyn').
Discriminator	nvarchar(128)	Rozróżnik typu obiektu (dla ORM).
ID_Pracownika	int (FK)	Aktualny posiadacz urządzenia.

Tabela NumeryKomorkowe ewidencjonuje karty SIM będące w posiadaniu firmy. Zawiera wrażliwe dane (PIN, PUK) oraz parametry taryfowe.

Tabela 3.5: Tabela: NumeryKomorkowe

Kolumna	Typ	Opis
ID	int (PK)	Klucz główny.
Numer	nvarchar(20)	Numer telefonu komórkowego (MSISDN).
NumerKarty	nvarchar(50)	Numer seryjny karty SIM (ICCID).
PIN	nvarchar(10)	Kod zabezpieczający PIN.
PUK	nvarchar(15)	Kod odblokowujący PUK.
PlanOpis	nvarchar(max)	Nazwa taryfy operatorskiej.
Status	nvarchar(20)	Status numeru (np. 'aktywny').
ID_Pracownika	int (FK)	Pracownik używający numeru.

Tabela NumeryStacjonarne służy do zarządzania numeracją stacjonarną (VoIP oraz analogową) w biurach firmy, z uwzględnieniem lokalizacji geograficznej (prefiksy).

Tabela 3.6: Tabela: NumeryStacjonarne

Kolumna	Typ	Opis
ID	int (PK)	Klucz główny.
Numer	nvarchar(20)	Numer stacjonarny lub wewnętrzny.
LiniaTyp	nvarchar(20)	Technologia (np. VoIP, Analog).
PrefiksKraj	nvarchar(5)	Numer kierunkowy kraju.
PrefiksMiasto	nvarchar(5)	Numer kierunkowy strefy.
Opis	nvarchar(200)	Opis lokalizacji (np. 'Recepcja').
StatusCOR	nvarchar(50)	Klasa uprawnień (np. 'międzynarodowe').
ID_Pracownika	int (FK)	Przypisany pracownik (opcjonalnie).

Dane Finansowe (Bilingi)

System przechowuje bilingi w dwóch oddzielnych tabelach (**BilingiKomorkowe** i **BilingiStacjonarne**), co wynika z różnic w strukturze plików dostarczanych przez operatorów oraz różnej specyfiki połączeń. Obie tabele posiadają jednak zbliżony schemat, umożliwiający ujednolicone raportowanie kosztów.

Tabela 3.7: Struktura tabel Bilingowych (Wspólna)

Kolumna	Typ	Opis
ID	int (PK)	Identyfikator pojedynczego połączenia.
DataPolaczenia	datetime	Data i czas rozpoczęcia rozmowy.
NumerTelefonu	nvarchar(20)	Numer, z którego wykonano połączenie (źródłowy).
NumerWybierany	nvarchar(20)	Numer docelowy.
TypPolaczenia	nvarchar(50)	Rodzaj usługi (np. Głosowe, SMS, Roaming).
CzasTrwania	nvarchar(20)	Długość połączenia.
KwotaNetto	decimal(18,2)	Koszt połączenia netto.
KwotaBrutto	decimal(18,2)	Koszt połączenia brutto.
NrFaktury	nvarchar(50)	Numer faktury, z której pochodzi wpis.

3.3 Relacje

W bazie danych zaimplementowano klucze obce (Foreign Keys) zapewniające integralność referencyjną:

- **Pracownicy** są przypisani do **Działów** (`ID_Dzialu`).
- **Stanowiska** są przypisane do **Działów** (`ID_Dzialu`).

- **Urzadzenia, NumeryKomorkowe, NumeryStacjonarne** posiadają klucz obcy `ID_Pracownika`, wskazujący na użytkownika zasobu.

- **Relacja zwrotna (Rekurencja):** W tabeli `Pracownicy` występuje relacja pracownik-przełożony. Kolumna `MenagerID` jest kluczem obcym wskazującym na kolumnę `ID` w tej samej tabeli. Pozwala to na odwzorowanie hierarchii podległości, gdzie jeden pracownik (Manager) może mieć przypisanych wielu podwładnych (relacja Jeden-do-Wielu opcjonalna).

- Tabele bilingowe są powiązane logicznie z tabelami numerów poprzez kolumnę `NumerTelefonu`.

3.4 Kod definiujący strukturę (DDL)

Struktura bazy danych została wdrożona w środowisku Microsoft SQL Server. Pełny kod źródłowy skryptów tworzących tabele, klucze obce oraz procedury składowane nie został umieszczony w niniejszym dokumencie ze względu na swoją objętość.

Kompletne skrypty instalacyjne bazy danych (pliki `.sql`) oraz dane inicjalne (tzw. seed data) dostępne są w publicznym repozytorium projektu:

`https://github.com/pablo42d/C-/tree/main/ProjektZaliczeniowy/DbSystemTelefonicznyGY`

Procedura wdrożenia i dane inicjalne

Poprawne funkcjonowanie systemu „SystemTelefonicznyGY”, w szczególności modułu zarządzania pracownikami, wymaga istnienia w bazie danych co najmniej jednego konta o uprawnieniach administratora.

Aplikacja w obecnej wersji nie posiada mechanizmu automatycznego tworzenia konta „SuperAdmina” przy pierwszym uruchomieniu (tzw. auto-seeding). W celu przygotowania środowiska produkcyjnego należy wprowadzić dane inicjalne bezpośrednio do bazy danych.

Automatyzacja wdrożenia (Skrypt Seed Data)

W repozytorium kodu źródłowego udostępniono dedykowany skrypt SQL (`InsertData.sql`), który automatyzuje proces tworzenia struktury firmy wraz z kontami startowymi. Skrypt ten zachowuje wymaganą kolejność operacji, wynikającą z więzów integralności bazy danych:

Alternatywnie, administrator bazy danych może wprowadzić te rekordy ręcznie, pamiętając o zachowaniu hierarchii zależności (klucze obce).

Po wykonaniu skryptu inicjalizacyjnego, w systemie dostępne jest konto o następujących poświadczeniach Administratora, umożliwiające pierwsze logowanie i dostęp do panelu konfiguracyjnego:

- **Login:** admin

- **Hasło:** admin123

Zaleca się zmianę domyślnego hasła natychmiast po pierwszym zalogowaniu. Dalsze zarządzanie użytkownikami odbywa się już w pełni z poziomu interfejsu graficznego aplikacji.

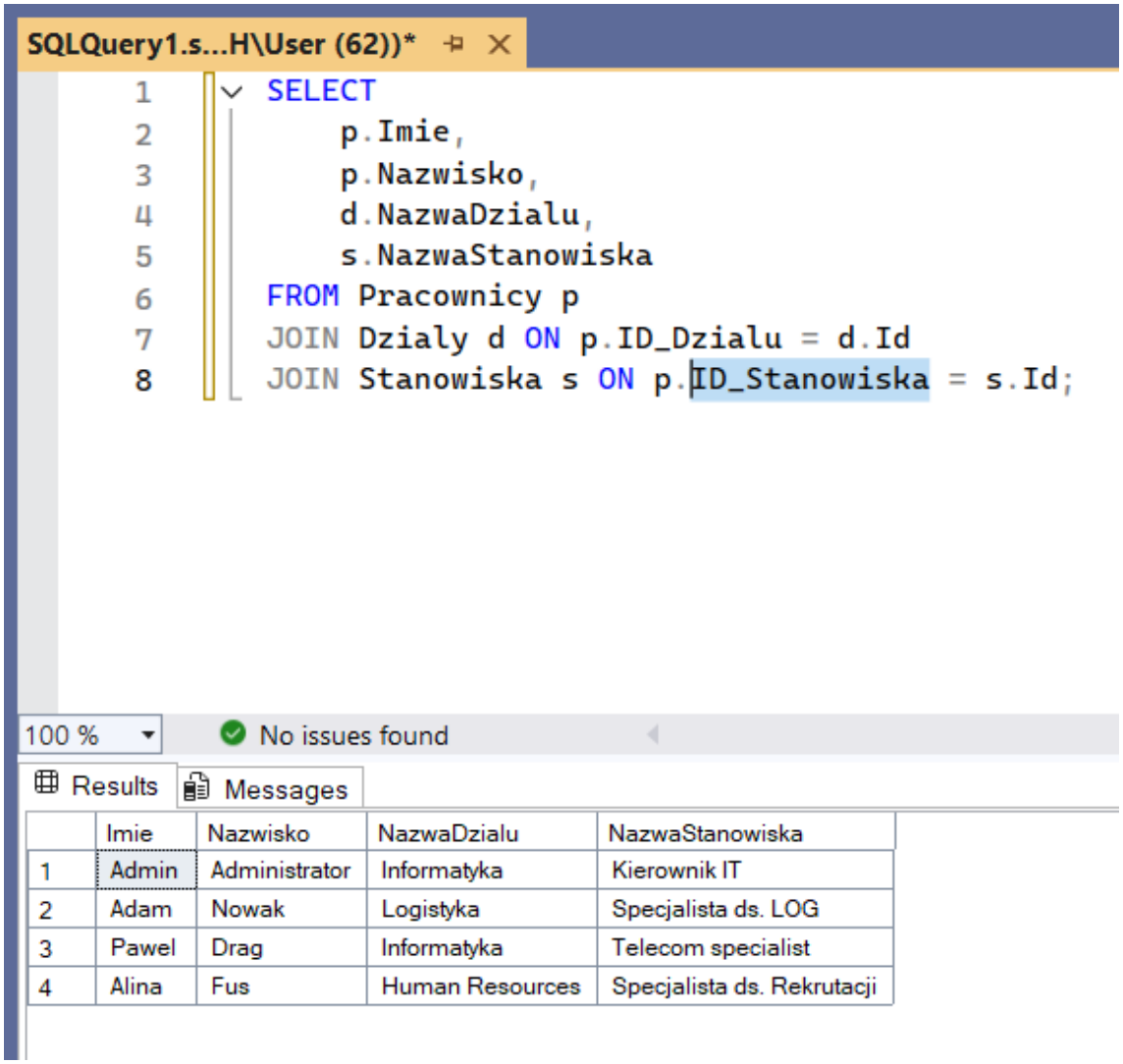
3.5 Weryfikacja operacji na danych (SQL DML)

W celu potwierdzenia poprawności zaprojektowanych relacji oraz przetestowania mechanizmów spójności danych, przygotowano zestaw zapytań testowych realizujących kluczowe operacje systemowe.

Pobieranie danych (Złączenia tabel - JOIN)

Zapytanie łączące trzy tabele w celu wyświetlenia listy pracowników wraz z nazwami ich działów i stanowisk.

```
SELECT
    p.Imie,
    p.Nazwisko,
    d.NazwaDzialu,
    s.NazwaStanowiska
FROM Pracownicy p
JOIN Dzialy d ON p.ID_Dzialu = d.Id
JOIN Stanowiska s ON p.ID_Stanowiska = s.Id;
```



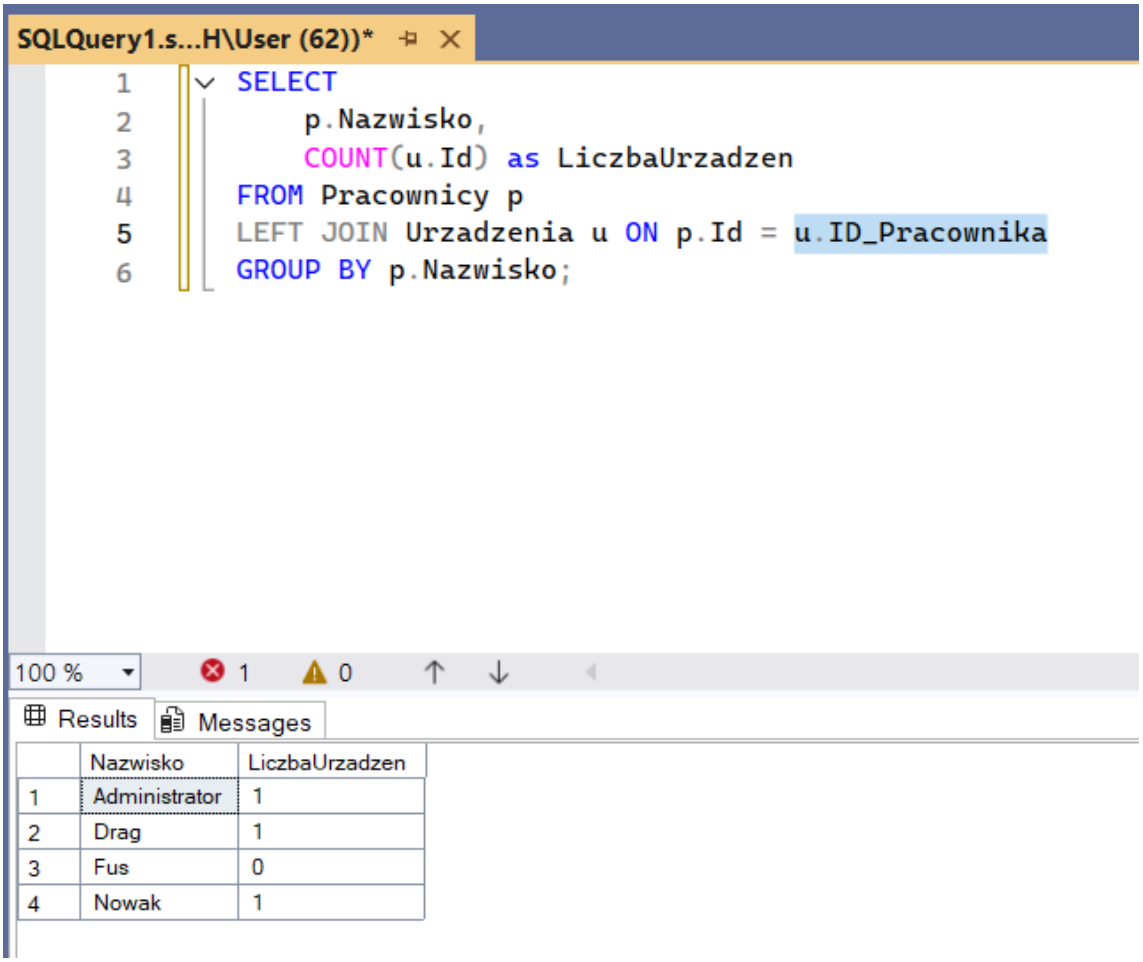
	Imie	Nazwisko	NazwaDzialu	NazwaStanowiska
1	Admin	Administrator	Informatyka	Kierownik IT
2	Adam	Nowak	Logistyka	Specjalista ds. LOG
3	Pawel	Drag	Informatyka	Telecom specialist
4	Alina	Fus	Human Resources	Specjalista ds. Rekrutacji

Rysunek 3.2: Wynik zapytania złączeniowego (JOIN). Źródło: Opracowanie własne.

Agregacja danych (GROUP BY)

Zapytanie zliczające ilość urządzeń przypisanych do każdego pracownika.

```
SELECT
    p.Nazwisko,
    COUNT(u.Id) as LiczbaUrzadzen
FROM Pracownicy p
LEFT JOIN Urzadzenia u ON p.Id = u.ID_Pracownika
GROUP BY p.Nazwisko;
```

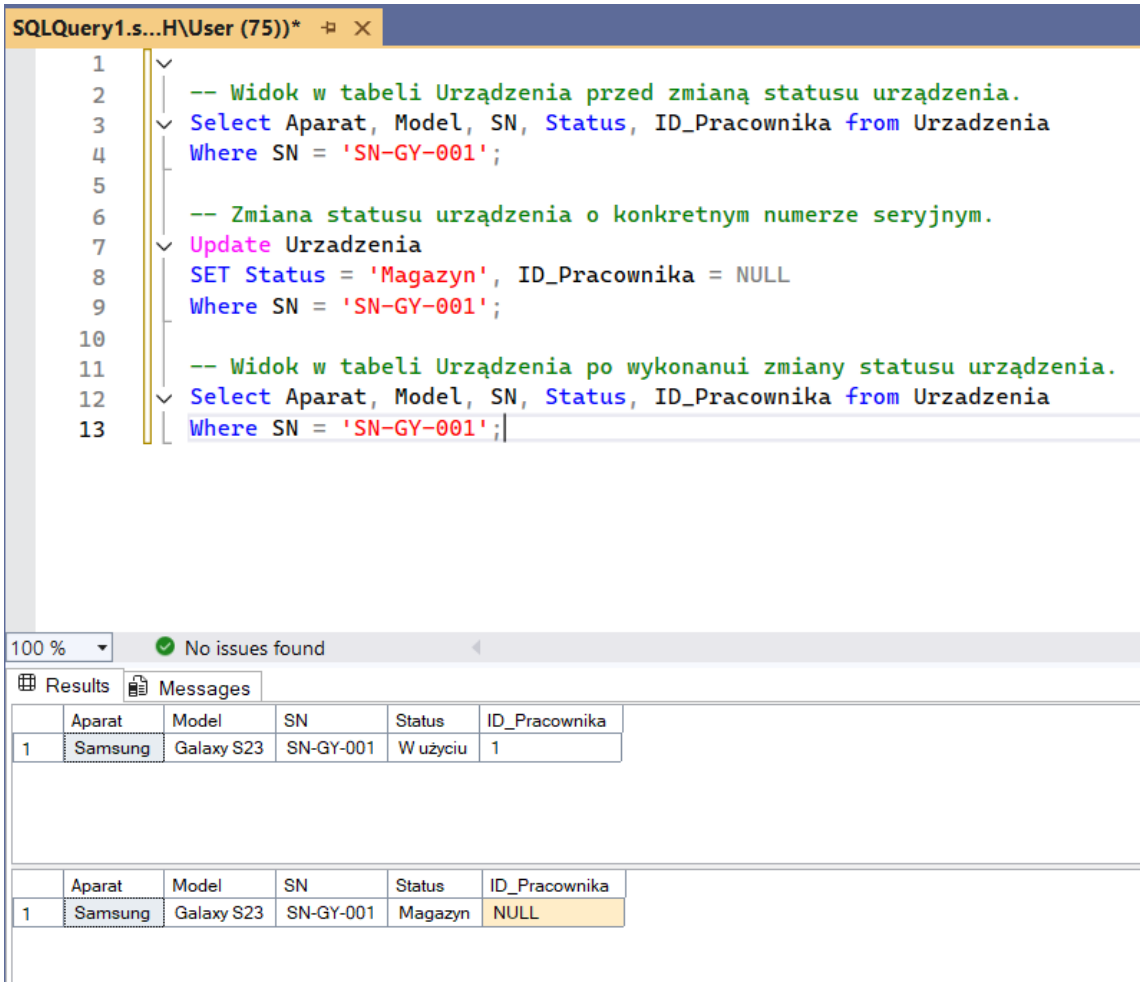


Rysunek 3.3: Wynik zapytania agregującego. Źródło: Opracowanie własne.

Modyfikacja danych (UPDATE)

Prezentacja procesu zwrotu sprzętu przez pracownika. Operacja polega na zmianie statusu urządzenia na 'Magazyn' oraz usunięciu powiązania z pracownikiem (ustawienie klucza obcego na NULL). Następnie wykonywana jest weryfikacja zmiany.

```
-- 1. Aktualizacja statusu i odpięcie pracownika
Update Urzadzenia
SET Status = 'Magazyn', ID_Pracownika = NULL
Where SN = 'SN-GY-001';

-- 2. Widok w tabeli Urządzenia po wykonanui zmiany statusu urządzenia.
Select Aparat, Model, SN, Status, ID_Pracownika from Urzadzenia
Where SN = 'SN-GY-001';
```



Rysunek 3.4: Wynik aktualizacji rekordu (UPDATE) wraz z weryfikacją (SELECT). Źródło: Opracowanie własne.

3.6 Podsumowanie wdrożenia bazy danych

Przeprowadzone testy zapytań potwierdziły, że zaimplementowana baza danych poprawnie realizuje założenia projektowe. Więzy integralności (klucze obce) skutecznie blokują próby wprowadzenia niespójnych danych, a optymalizacja typów danych pozwala na wydajne przetwarzanie informacji o bilingach.

.

Rozdział 4

Implementacja kluczowych modułów

W niniejszym rozdziale przedstawiono szczegóły implementacyjne najważniejszych komponentów systemu, algorytmy przetwarzania danych oraz strukturę projektu.

Kompletna struktura rozwiązania, obejmująca kod źródłowy aplikacji (ASP.NET MVC), zaimplementowane testy jednostkowe, skrypty bazy danych oraz przykładowe pliki bilingowe, została udostępniona w publicznym repozytorium w serwisie GitHub.

Zarówno kod aplikacji, jak i towarzyszące mu artefakty projektowe, są dostępne pod poniższym adresem:

https://github.com/pablo42d/C-/tree/main/ProjektZaliczeniowy

4.1 Struktura fizyczna projektu

Aplikacja została zorganizowana zgodnie ze strukturą katalogów wymaganą przez framework ASP.NET MVC, z dodatkowym wydzieleniem warstwy logiki biznesowej w osobnym katalogu. Poniższe zestawienie prezentuje pełną strukturę plików projektu (widok Solution Explorer).

```
SystemTelefonicznyGY/
|-- App_Data/                # (Pusty) Katalog na lokalne pliki baz danych
|-- App_Start/               # Konfiguracja startowa aplikacji
|   |-- BundleConfig.cs      # Konfiguracja bundli (CSS/JS)
|   |-- FilterConfig.cs      # Rejestracja globalnych filtrów
|   |-- RouteConfig.cs       # Definicje routingu URL
|-- Content/                 # Zasoby statyczne (style)
|   |-- Site.css             # Główny arkusz stylów CSS
|-- Controllers/             # Warstwa kontrolerów (obsługa żądań HTTP)
|   |-- AdministratorController.cs    # Główny kontroler panelu admina
|   |-- HomeController.cs             # Strona startowa i informacyjna
|   |-- KontoController.cs            # Obsługa logowania (Login)
|   |-- PanelUzytkownikaController.cs # Panel dla zwykłego pracownika
|-- Logika/                  # Warstwa logiki biznesowej (BLL)
|   |-- Interfejsy/          # Kontrakty (Dependency Injection)
|   |   |-- IBazaDanych.cs
|   |   |-- IBilingService.cs
|   |   |-- IDzialyService.cs
|   |   |-- IPracownikService.cs
|   |   |-- IZasobyService.cs
|   |-- BazaDanych.cs        # Dostępu do danych(ADO.NETWrapper)
|   |-- BilingService.cs     # Logika importu i przetwarzania bilingów
```

```
|   |-- DzialyService.cs     # Zarządzanie strukturą działów firmy
|   |-- PracownikService.cs  # Logika operacji na użytkownikach
|   |-- ZasobyService.cs     # Logika zarządzania sprzętem i numerami
|-- Models/                  # Modele danych (DTO) i ViewModele
|   |-- ElementSystemu.cs             # Klasa bazowa (abstrakcyjna)
|   |-- LogowanieModel.cs             # Model formularza logowania
|   |-- Modem.cs                      # Klasa urządzenia - Modem
|   |-- PodsumowaniePracownikaModel.cs # Model raportowy
|   |-- Pracownik.cs                  # Reprezentacja pracownika
|   |-- TelefonKomorkowy.cs           # Klasa urządzenia - Komórka
|   |-- TelefonStacjonarny.cs         # Klasa urządzenia - Telefon
|   |-- Urzadzenie.cs                 # Klasa bazowa urządzeń
|-- Views/                   # Warstwa prezentacji (.cshtml)
|   |-- Administrator/       # Widoki dla Administratora
|   |   |-- Dzialy.cshtml, Edytuj.cshtml, Import.cshtml, Index.cshtml,
|   |   |-- Pracownicy.cshtml, Raporty.cshtml, Urzadzenia.cshtml, ...
|   |-- Home/                # Widoki ogólne
|   |   |-- About.cshtml, Contact.cshtml, Index.cshtml
|   |-- Konto/               # Widoki autoryzacji
|   |   |-- Login.cshtml
|   |-- PanelUzytkownika/    # Widoki pracownika
|   |   |-- BilingiKomorkowe.cshtml, BilingiStacjonarne.cshtml, Index...
|   |-- Shared/              # Widoki współdzielone
|       |-- _Layout.cshtml   # Główny szablon strony (menu, stopka)
|       |-- Error.cshtml     # Strona błędu
|-- Global.asax              # Obsługa zdarzeń cyklu życia aplikacji
|-- Web.config               # Główna konfiguracja (Connection String)
```

4.2 Implementacja logiki biznesowej

Kluczowym elementem systemu jest moduł importu danych bilingowych, zaimplementowany w klasie `BilingService`. Klasa ta realizuje interfejs `IBilingService` i odpowiada za przetwarzanie plików CSV dostarczanych przez operatora telekomunikacyjnego.

W przeciwieństwie do prostego mapowania obiektowo-relacyjnego (ORM), ze względu na wydajność oraz specyfikę danych wejściowych, zastosowano bezpośrednie operacje na bazie danych z wykorzystaniem autorskiego wrappera ADO.NET. Poniższy listing (Listing 4.1) prezentuje metodę `ImportujPlik`, która realizuje następujące zadania:

- Odczyt strumieniowy pliku (klasa `StreamReader`) w celu optymalizacji zużycia pamięci.
- Walidacja i parsowanie niestandardowych formatów daty i czasu.
- Zabezpieczenie przed duplikacją danych poprzez usunięcie wcześniejszych rekordów dla tego samego numeru faktury przed rozpoczęciem importu.
- Konwersja danych numerycznych i wstawienie rekordów do odpowiedniej tabeli (`BilingiKomorkowe` lub `BilingiStacjonarne`).

Rysunek 4.1: Implementacja metody importującej dane z CSV w BilingService.cs

```
public WynikImportu ImportujPlik(Stream strumienPliku, string typ)
{
    var wynik = new WynikImportu();
    // Wybór tabeli w zależności od typu
    string tabelaSQL = (typ == "kom")
                       ? "BilingiKomorkowe"
                       : "BilingiStacjonarne";
    int licznik = 0;
    bool czyUsunietoStaraFakture = false;

    using (var reader = new StreamReader(strumienPliku))
    {
        reader.ReadLine(); // Pomijamy nagłówek

        while (!reader.EndOfStream)
        {
            var linia = reader.ReadLine();
            if (string.IsNullOrWhiteSpace(linia)) continue;

            var wartosci = linia.Split(';');
            if (wartosci.Length < 9) continue;

            string nrFaktury = wartosci[8].Trim().Replace("'", "''");

            // Mechanizm zapobiegający duplikatom:
            // usunięcie starych danych dla tej faktury
            if (!czyUsunietoStaraFakture && !string.IsNullOrEmpty(nrFaktury))
            {
                string sqlDel = $"DELETE FROM {tabelaSQL} " +
                                $"WHERE NrFaktury = '{nrFaktury}'";
                _baza.WykonajPolecenie(sqlDel);

                czyUsunietoStaraFakture = true;
                wynik.NumerFaktury = nrFaktury;
            }

            // Parsowanie daty
            string dataDlaSQL;
            if (DateTime.TryParseExact(wartosci[0], "dd.MM.yyyy HH:mm",
                System.Globalization.CultureInfo.InvariantCulture,
                System.Globalization.DateTimeStyles.None, out DateTime dt))
            {
                dataDlaSQL = dt.ToString("yyyy-MM-dd HH:mm:ss");
            }
            else
            {
                DateTime.TryParse(wartosci[0], out dt);
                dataDlaSQL = dt.ToString("yyyy-MM-dd HH:mm:ss");
            }
```

```
            // Przygotowanie danych (zamiana przecinków na kropki dla SQL)
            string numerA = KonwertujNumer(wartosci[1]);
            string numerB = KonwertujNumer(wartosci[2]);
            string typPol = wartosci[3].Replace("'", "''");
            string czas = wartosci[5];
            string netto = wartosci[6].Replace(',', '.');
            string brutto = wartosci[7].Replace(',', '.');

            string sql = $@"INSERT INTO {tabelaSQL}
               (DataPolaczenia, NumerTelefonu, NumerWybierany,
               TypPolaczenia, CzasTrwania, KwotaNetto,
               KwotaBrutto, NrFaktury)
               VALUES ('{dataDlaSQL}', '{numerA}', '{numerB}', '{typPol}',
                       '{czas}', {netto}, {brutto}, '{nrFaktury}')";

            _baza.WykonajPolecenie(sql);
            licznik++;
        }
    }
    wynik.LiczbaRekordow = licznik;
    return wynik;
}
```

4.3 Bezpieczeństwo i Autoryzacja

W obecnej wersji systemu zaimplementowano mechanizm autoryzacji oparty na sesji użytkownika. Kontroler logowania nie łączy się bezpośrednio z bazą danych, lecz wykorzystuje warstwę logiki biznesowej poprzez wstrzykiwanie zależności interfejsu `IPracownikService`.

Proces logowania polega na przekazaniu danych z modelu `LogowanieModel` do metody serwisu, która weryfikuje poświadczenia. W przypadku sukcesu, kluczowe dane pracownika (ID, Imię, Rola) zapisywane są w sesji serwera, co pozwala na ich późniejsze wykorzystanie w innych częściach aplikacji.

Rysunek 4.2: Metoda logowania w KontoController.cs z użyciem serwisu

```
[HttpPost]
public ActionResult Login(LogowanieModel model)
{
    if (ModelState.IsValid)
    {
        // Delegacja logiki do warstwy serwisowej (Separacja kodu)
        // Metoda Zaloguj zwraca wiersz danych lub null
        var pracownikRow = _pracownikService.Zaloguj(
                                model.Login, model.Haslo);

        if (pracownikRow != null)
        {
            // Zapisanie danych użytkownika w Sesji
            Session["IdPracownika"] = pracownikRow["ID"];
```

```
            Session["ImiePracownika"] = pracownikRow["Imie"];
            Session["NazwiskoPracownika"] = pracownikRow["Nazwisko"];
            Session["RolaPracownika"] = pracownikRow["Rola"];

            return RedirectToAction("Index", "Home");
        }

        else
        {
            ModelState.AddModelError("", "Błędny login lub hasło
            pracownika.");
        }
    }
    return View(model);
}
```

4.4 Testy oprogramowania

Proces weryfikacji systemu „SystemTelefonicznyGY" został przeprowadzony w sposób hybrydowy, łącząc automatyczne testy jednostkowe kluczowych komponentów z manualnymi testami funkcjonalnymi interfejsu użytkownika. Głównym celem testów było potwierdzenie zgodności działania aplikacji ze specyfikacją wymagań, weryfikacja mechanizmów bezpieczeństwa oraz zapewnienie integralności danych finansowych.

Automatyczne testy jednostkowe

W celu zapewnienia wysokiej niezawodności warstwy logicznej aplikacji, zaimplementowano testy jednostkowe z wykorzystaniem frameworka **MSTest**. Aby umożliwić testowanie kontrolerów w izolacji od bazy danych i serwera, zastosowano wzorzec Wstrzykiwania Zależności (Dependency Injection) oraz bibliotekę **Moq**.

Kluczowym obszarem poddanym automatycznej weryfikacji był moduł autoryzacji (`KontoController`) oraz panelu administratora (`AdministratorController`). Dzięki zastosowaniu obiektów typu *Mock*, zasymulowano działanie warstwy serwisowej oraz kontekstu HTTP (Sesji), co pozwoliło na sprawdzenie logiki biznesowej bez konieczności uruchamiania całej aplikacji.

Poniższy listing (Listing 4.3) prezentuje przykładowy kod testu weryfikującego poprawność procesu logowania i ustawiania zmiennych sesyjnych.

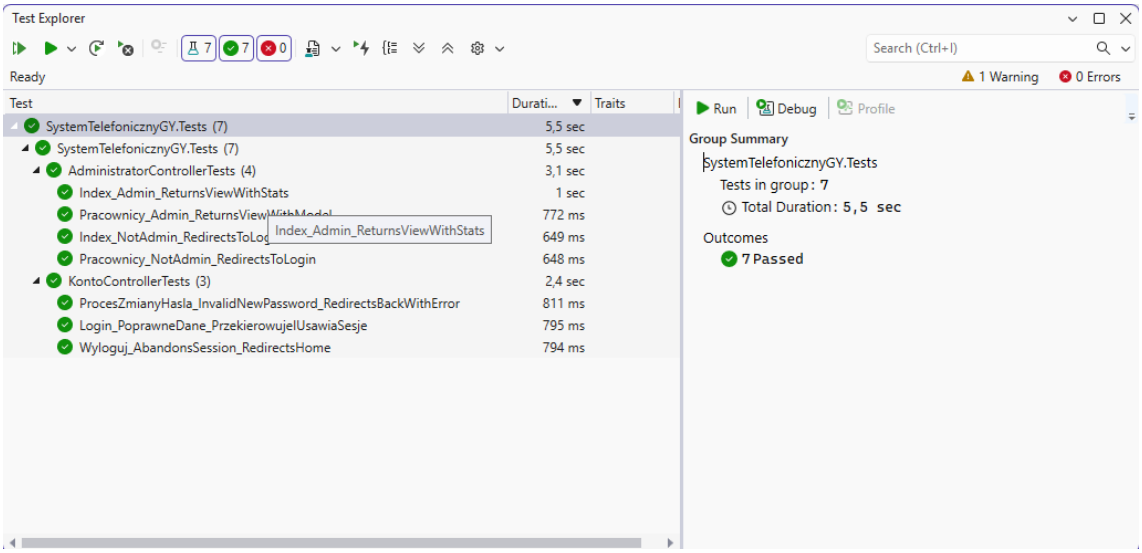
Rysunek 4.3: Test jednostkowy metody Login z wykorzystaniem biblioteki Moq

```
[TestMethod]
public void Login_PoprawneDane_PrzekierowujeIUsawiaSesje()
{
    // 1. ARRANGE (Przygotowanie makiety serwisu)
    var mockService = new Mock<IPracownikService>();

    // Symulacja danych zwracanych przez bazę (bez fizycznego połączenia)
    DataTable dt = new DataTable();
    dt.Columns.Add("ID"); dt.Columns.Add("Imie");
    dt.Columns.Add("Nazwisko"); dt.Columns.Add("Rola");
```

```
    var row = dt.NewRow();
    row["ID"] = 1; row["Imie"] = "Jan"; row["Rola"] = "User";

    // Konfiguracja Mocka: Dla loginu "admin" zwróć przygotowany wiersz
    mockService.Setup(s => s.Zaloguj("admin", "123")).Returns(row);

    var controller = new KontoController(mockService.Object);
    // Wstrzyknięcie fałszywego kontekstu (Mock Session & Request)
    controller.ControllerContext = StworzKontekst(controller);

    // 2. ACT (Wykonanie akcji logowania)
    var result = controller.Login(new LogowanieModel {
        Login = "admin", Haslo = "123"
    }) as RedirectToRouteResult;

    // 3. ASSERT (Weryfikacja wyników)
    Assert.IsNotNull(result);
    // Sprawdzenie czy przekierowano do strony głównej
    Assert.AreEqual("Index", result.RouteValues["action"]);
    // Sprawdzenie czy w sesji zapisano dane pracownika
    Assert.AreEqual("Jan", controller.Session["ImiePracownika"]);
}
```

Wyniki przeprowadzonych testów w środowisku Visual Studio potwierdziły poprawność implementacji. Wszystkie zdefiniowane przypadki testowe zakończyły się wynikiem pozytywnym, co przedstawiono na poniższym zrzucie ekranu.

Rysunek 4.4: Podsumowanie wyników testów jednostkowych w Test Explorer (Visual Studio).



Testy funkcjonalne (Scenariusze i wizualizacja)

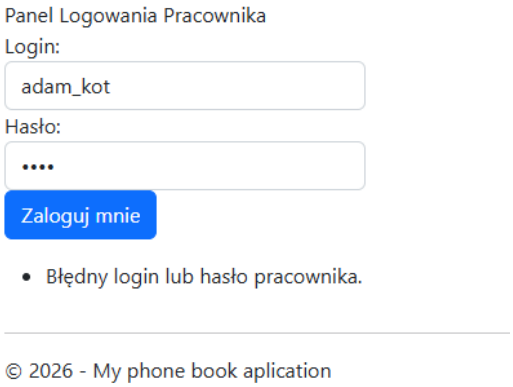
Uzupełnieniem testów automatycznych była seria testów manualnych, mających na celu weryfikację zachowania systemu z punktu widzenia użytkownika końcowego. Poniżej przedstawiono szczegółowy przebieg wybranych scenariuszy testowych wraz z dokumentacją wizualną potwierdzającą ich wynik.

Scenariusz 1: Weryfikacja logowania z błędnymi danymi

Cel: Sprawdzenie, czy system blokuje dostęp nieautoryzowanym użytkownikom.

Przebieg: Wprowadzono nieistniejący login oraz błędne hasło w formularzu logowania.

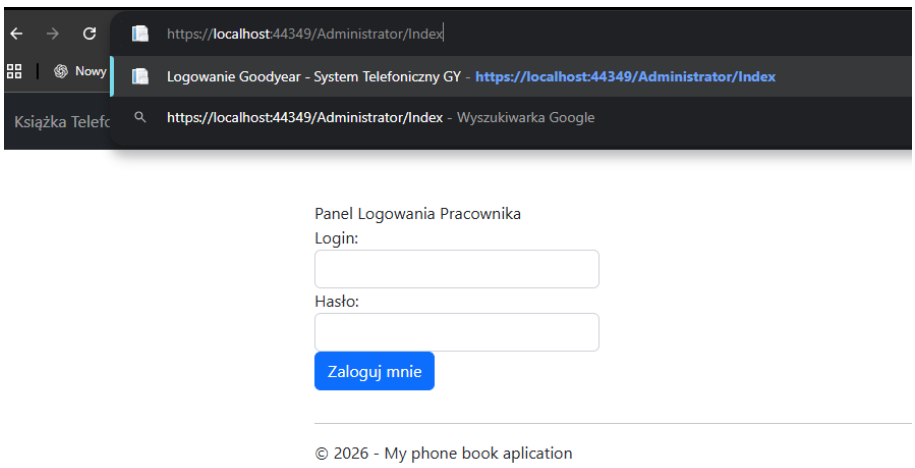
Wynik: Pozytywny. System wyświetlił komunikat walidacyjny „Błędny login lub hasło pracownika” i nie udzielił dostępu.



Rysunek 4.5: Efekt działania Scenariusza 1 – odmowa dostępu.

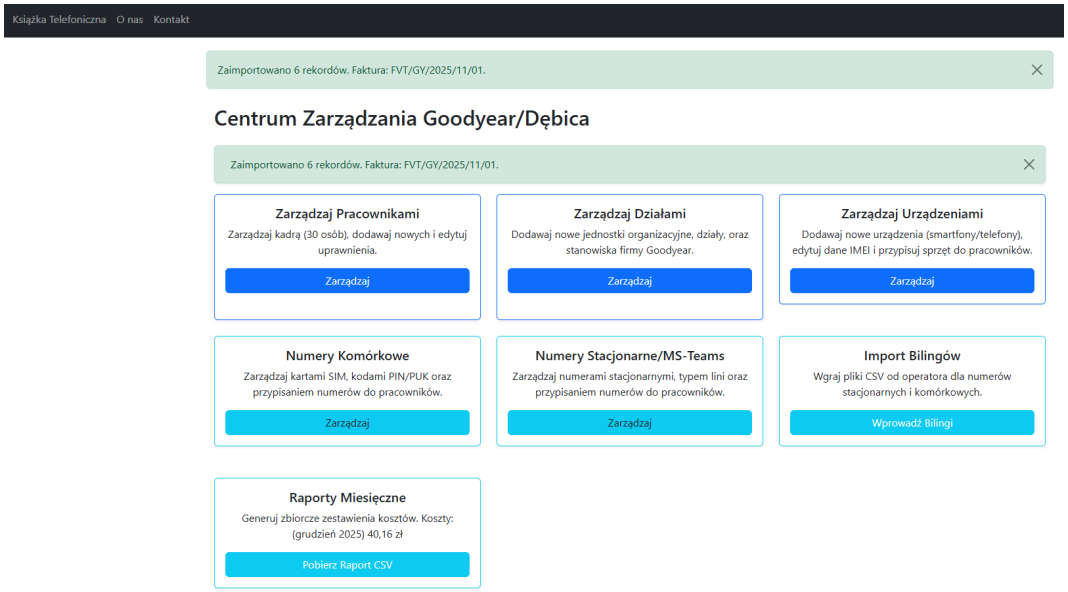
Scenariusz 2: Próba nieautoryzowanego dostępu (Deep Linking)
Cel: Weryfikacja zabezpieczenia kontrolerów przed bezpośrednim wejściem z paska adresu.
Przebieg: Próba wejścia na adres `/Administrator/Index` bez uprzedniego zalogowania.
Wynik: Pozytywny. System automatycznie przekierował żądanie do strony logowania (`/Konto/Login`).

Opcjonalnie zdjęcie (jeśli chcesz pokazać przekierowanie), jeśli nie - usuń figure



Rysunek 4.6: Przekierowanie do logowania przy próbie wejścia bez uprawnień.

Scenariusz 3: Import pliku bilingowego (Happy Path)
Cel: Sprawdzenie poprawności parsowania i zapisu danych z pliku CSV.
Przebieg: Wybrano poprawny plik CSV z bilingami operatora i uruchomiono proces importu.
Wynik: Pozytywny. Dane zostały zapisane w bazie SQL, a system wyświetlił podsumowanie operacji.



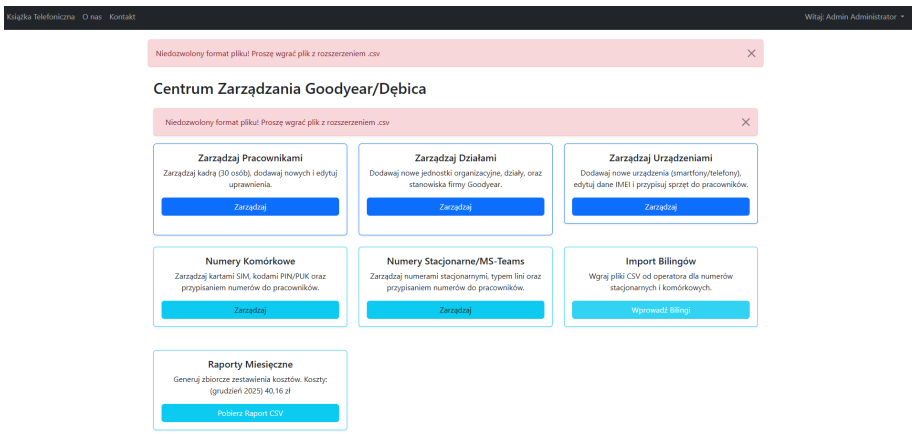
Rysunek 4.7: Poprawny import danych bilingowych (Scenariusz 3).

Scenariusz 4: Walidacja formatu plików wejściowych

Cel: Zabezpieczenie przed wgraniem niewłaściwego typu pliku.

Przebieg: Podjęto próbę wgrania pliku z rozszerzeniem `.pdf` w module importu.

Wynik: Pozytywny. System odrzucił plik i wyświetlił komunikat o błędnym formacie.

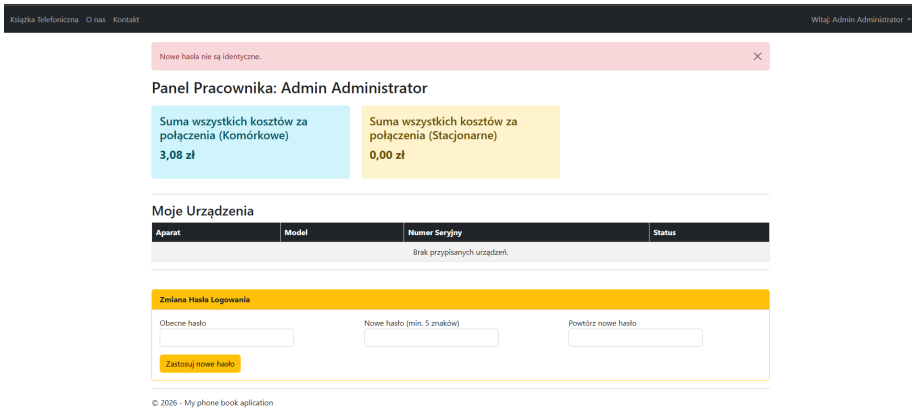


Rysunek 4.8: Komunikat błędu przy próbie wgrania niedozwolonego pliku.

Scenariusz 5: Walidacja formularza zmiany hasła

Cel: Sprawdzenie mechanizmu porównywania haseł (nowe vs powtórz).

Wynik: Pozytywny. Przy wpisaniu dwóch różnych haseł, walidacja modelu (`ModelState`) zablokowała wysyłkę formularza, wyświetlając komunikat „Nowe hasła nie są identyczne”.

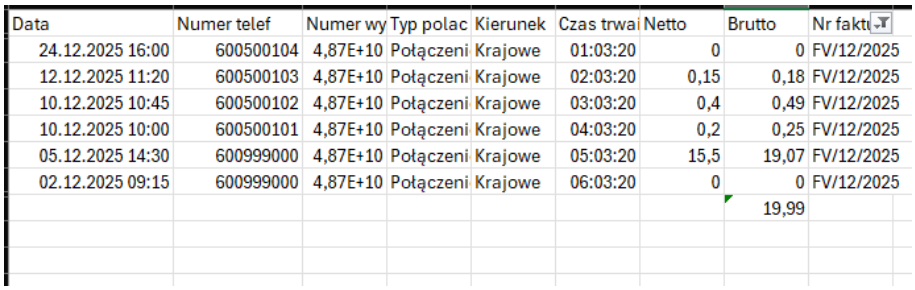


Rysunek 4.9: Walidacja formularza zmiany hasła.

Weryfikacja spójności danych bilingowych

Ze względu na finansowy charakter modułu bilingowego, kluczowym elementem weryfikacji było potwierdzenie poprawności obliczeń matematycznych podczas importu danych. Procedura testowa polegała na:

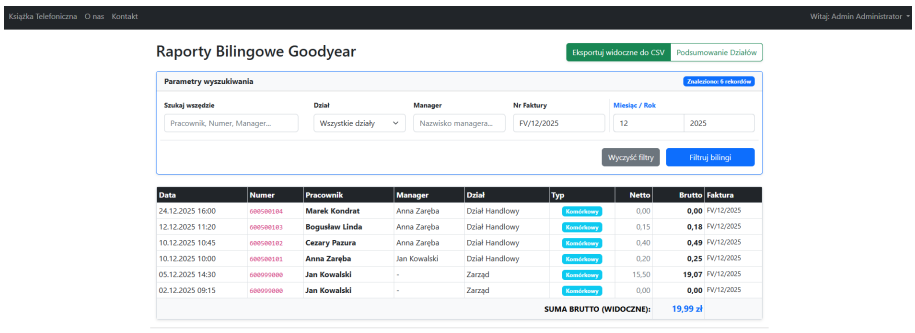
1. Pobraniu sumarycznej kwoty brutto bezpośrednio z pliku źródłowego CSV (za pomocą arkusza kalkulacyjnego Excel).



Data	Numer telef	Numer wy	Typ polac	Kierunek	Czas trwai	Netto	Brutto	Nr fakt
24.12.2025 16:00	600500104	4,87E+10	Połączeni	Krajowe	01:03:20	0	0	FV/12/2025
12.12.2025 11:20	600500103	4,87E+10	Połączeni	Krajowe	02:03:20	0,15	0,18	FV/12/2025
10.12.2025 10:45	600500102	4,87E+10	Połączeni	Krajowe	03:03:20	0,4	0,49	FV/12/2025
10.12.2025 10:00	600500101	4,87E+10	Połączeni	Krajowe	04:03:20	0,2	0,25	FV/12/2025
05.12.2025 14:30	600999000	4,87E+10	Połączeni	Krajowe	05:03:20	15,5	19,07	FV/12/2025
02.12.2025 09:15	600999000	4,87E+10	Połączeni	Krajowe	06:03:20	0	0	FV/12/2025
							19,99	

Rysunek 4.10: Podsumowanie brutto w importowanym pliku.

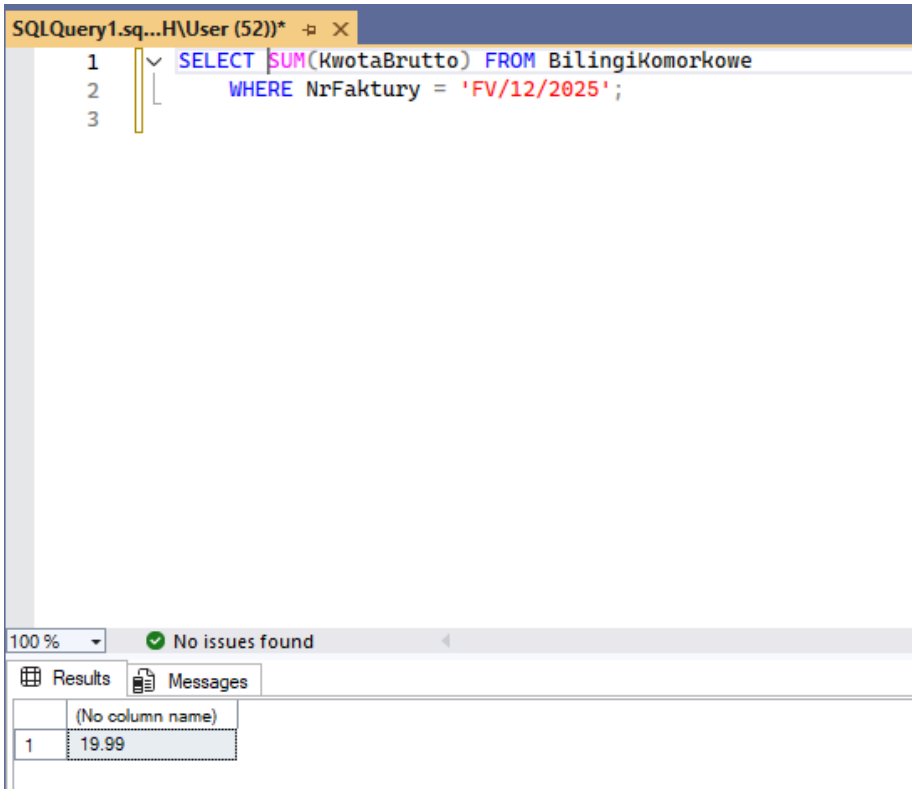
2. Wykonaniu importu pliku przez aplikację.



Rysunek 4.11: Podsumowanie brutto w aplikacji.

3. Wykonaniu niezależnego zapytania SQL sumującego kwoty w bazie danych:

```
SELECT SUM(KwotaNetto), SUM(KwotaBrutto) FROM BilingiKomorkowe
WHERE NrFaktury = 'FV/12/2025';
```



Rysunek 4.12: Podsumowanie brutto w bazie danych.

4. Porównaniu zwracanych wartości.

Testy wykazały pełną zgodność kwot (z dokładnością do dwóch miejsc po przecinku), co potwierdza poprawność zaimplementowanego algorytmu parsowania liczb oraz właściwą konfigurację typów danych w bazie SQL (`decimal(19,99)`).

.

Rozdział 5

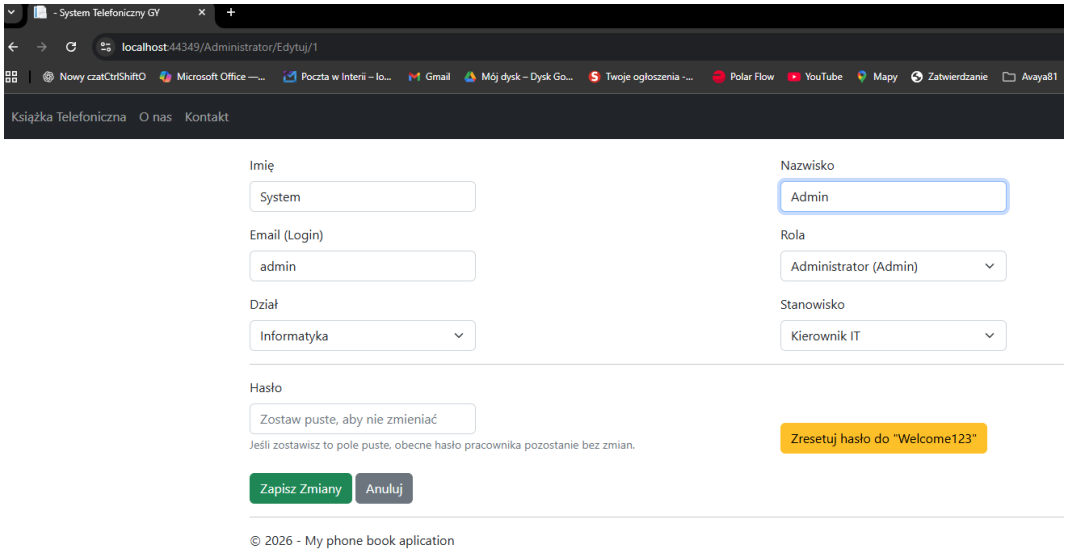
Opis kluczowych funkcjonalności

System został zaprojektowany w architekturze modułowej, dzieląc funkcjonalności na dwa główne obszary w zależności od poziomu uprawnień zalogowanego użytkownika (rola *Administrator* oraz *User*).

5.1 Moduł Administratora

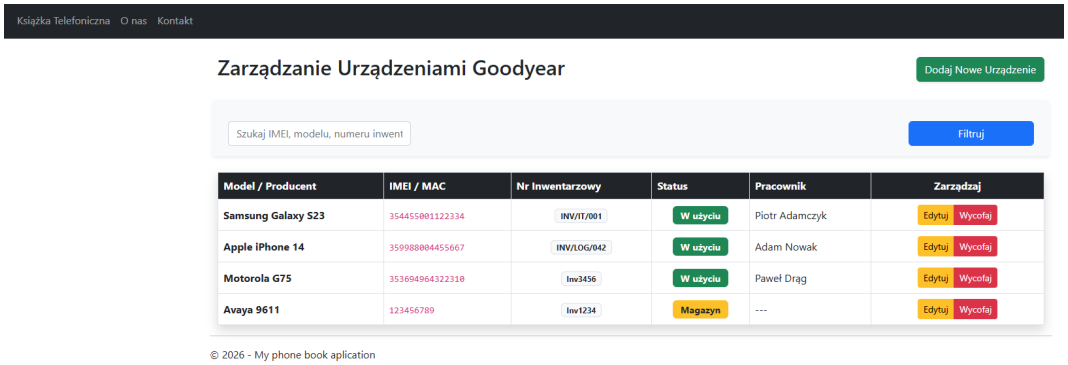
Administrator posiada pełne uprawnienia do zarządzania danymi referencyjnymi oraz procesami bilingowymi. Do kluczowych funkcji tego modułu należą:

- **Zarządzanie Pracownikami:** Implementacja pełnego interfejsu CRUD (Create, Read, Update, Delete) dla kont użytkowników. Funkcjonalność obejmuje przypisywanie pracowników do działów, stanowisk oraz mechanizm resetowania hasła do wartości domyślnej w przypadku jego utraty.



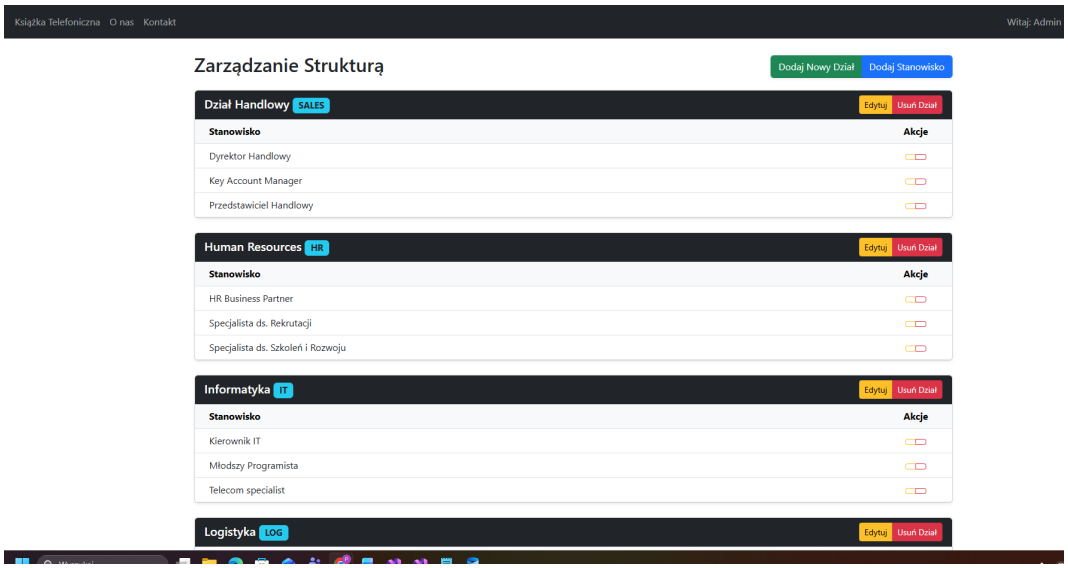
Rysunek 5.1: Panel zarządzania pracownikami i edycji uprawnień.

- **Zarządzanie Zasobami IT:** Ewidencja sprzętu telekomunikacyjnego (telefony komórkowe, stacjonarne, modemy). Moduł pozwala na dodawanie nowych urządzeń do bazy, edycję ich parametrów (IMEI, Model, Numer seryjny) oraz przypisywanie ich do konkretnych pracowników.



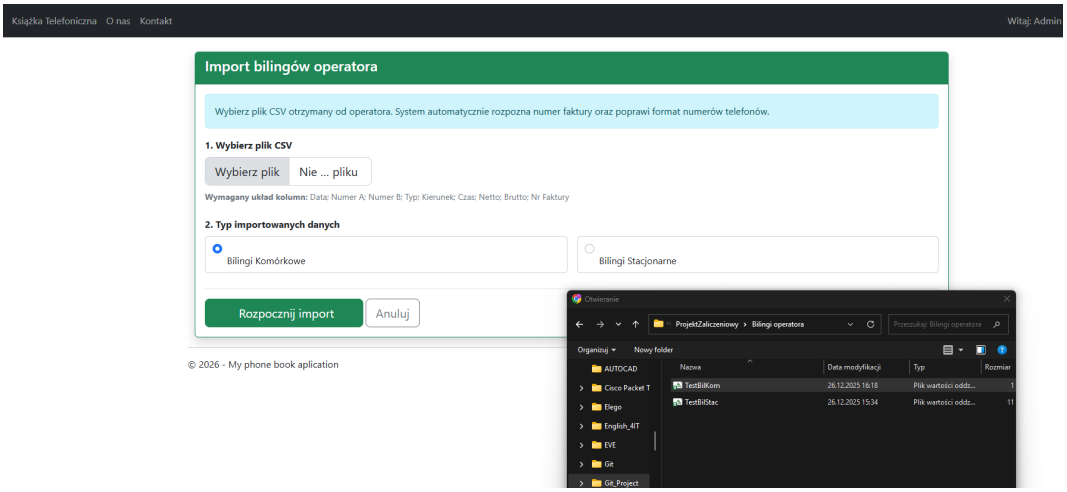
Rysunek 5.2: Interfejs ewidencji zasobów sprzętowych.

- **Zarządzanie Strukturą Organizacyjną:** Możliwość definiowania i edycji struktury działów firmy, co jest kluczowe dla poprawnego raportowania kosztów w ujęciu ’MPK’ (Miejsce Powstawania Kosztów).



Rysunek 5.3: Widok zarządzania strukturą organizacyjną firmy.

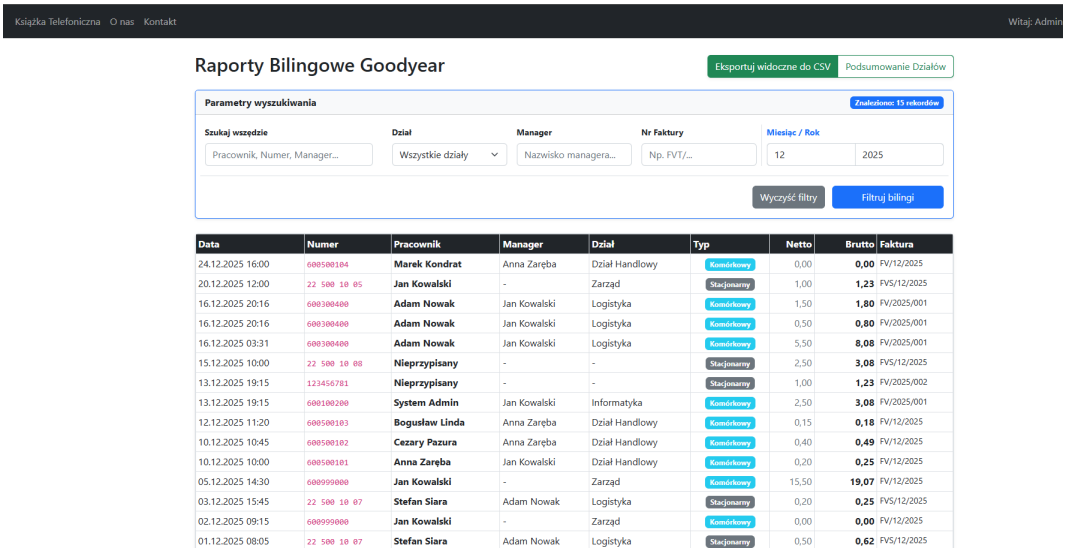
- **Import i Przetwarzanie Danych:** Funkcja `ImportujCSV` umożliwia wczytanie surowych plików bilingowych od operatora. System automatycznie weryfikuje poprawność struktury pliku, eliminuje duplikaty (na podstawie numeru faktury) oraz parsuje niestandardowe formaty dat.



Rysunek 5.4: Moduł importu plików bilingowych CSV.

- **Raportowanie i Analiza:** Zaawansowany silnik filtrowania danych bilingowych. Administrator ma możliwość generowania zestawień kosztów według:
  - Pracownika lub Managera,
  - Działu organizacyjnego,
  - Zakresu dat,
  - Numeru faktury lub numeru telefonu.

  Dostępna jest również funkcja eksportu przefiltrowanych widoków do formatu CSV w celu dalszej obróbki np. w arkuszach kalkulacyjnych.

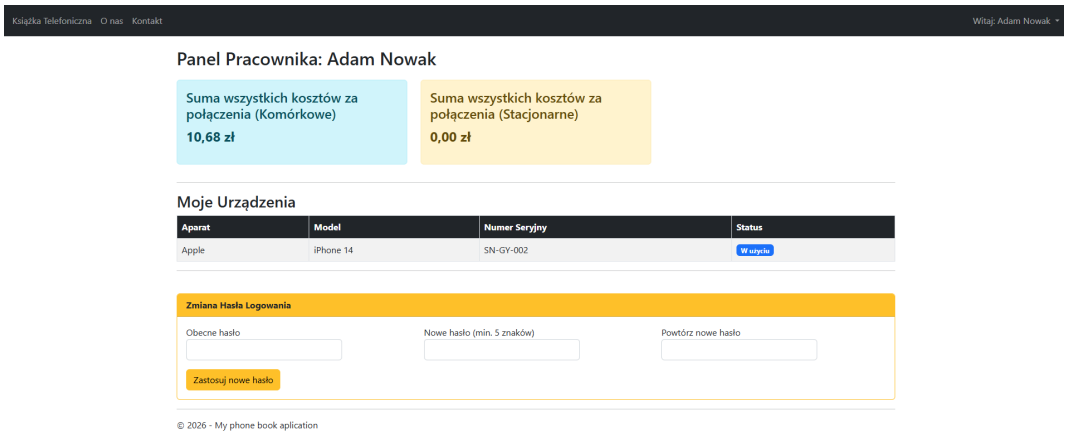


Rysunek 5.5: Narzędzie do filtrowania i generowania raportów kosztowych.

5.2 Moduł Użytkownika

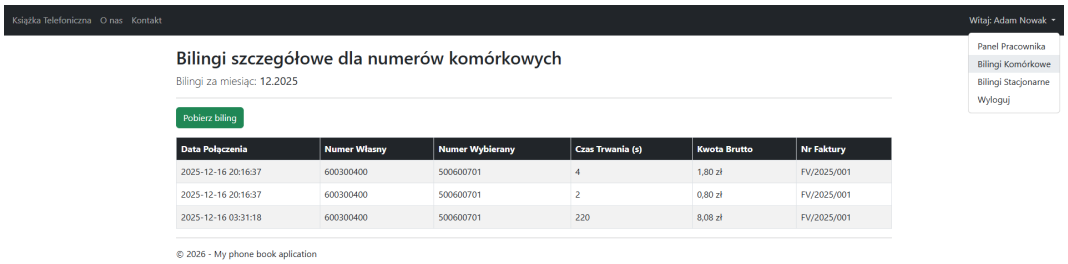
Pracownik szeregowy po zalogowaniu otrzymuje dostęp do panelu samoobsługowego (Self-Service), który umożliwia monitorowanie zasobów oraz kosztów. Funkcjonalności podzielono na widok ogólny (Pulpit) oraz widoki szczegółowe.

- **Panel Pracownika (Pulpit Główny):** Jest to centralny widok aplikacji integrujący trzy kluczowe informacje::
  - **Podsumowanie kosztów (Kafelki):** Sekcja prezentująca sumaryczne kwoty za połączenia komórkowe i stacjonarne ze wszystkich dostępnych faktur. Umożliwia szybką ocenę wszystkich wydatków wygenerowanych na numerach przypisanych do danego pracownika.
  - **Moje Urządzenia:** Tabela inwentarzowa wyświetlająca sprzęt służbowy (model, numer seryjny) aktualnie przypisany do użytkownika.
  - **Zmiana Hasła:** Formularz umożliwiający bezpieczną zmianę poświadczeń logowania. Moduł waliduje poprawność starego hasła oraz wymusza minimalną złożoność nowego ciągu znaków.



Rysunek 5.6: Widok „Panel Pracownika” w panelu pracownika.

- **Szczegółowa Kontrola Kosztów (Bilingi):** Funkcjonalność dostępna z poziomu menu głównego (rozwijana lista). System prezentuje szczegółowy wykaz połączeń służbowych (komórkowych i stacjonarnych).
  - Dane są automatycznie filtrowane – użytkownik widzi wyłącznie rekordy powiązane z jego numerami telefonów.
  - Widok obejmuje historię połączeń z ostatniego miesiąca.
  - Dostępny jest przycisk eksportu, umożliwiający pobranie zestawienia do pliku formatu CSV w celu dalszej analizy.



Rysunek 5.7: Podgląd kosztów i historii połączeń dla zalogowanego użytkownika.

.

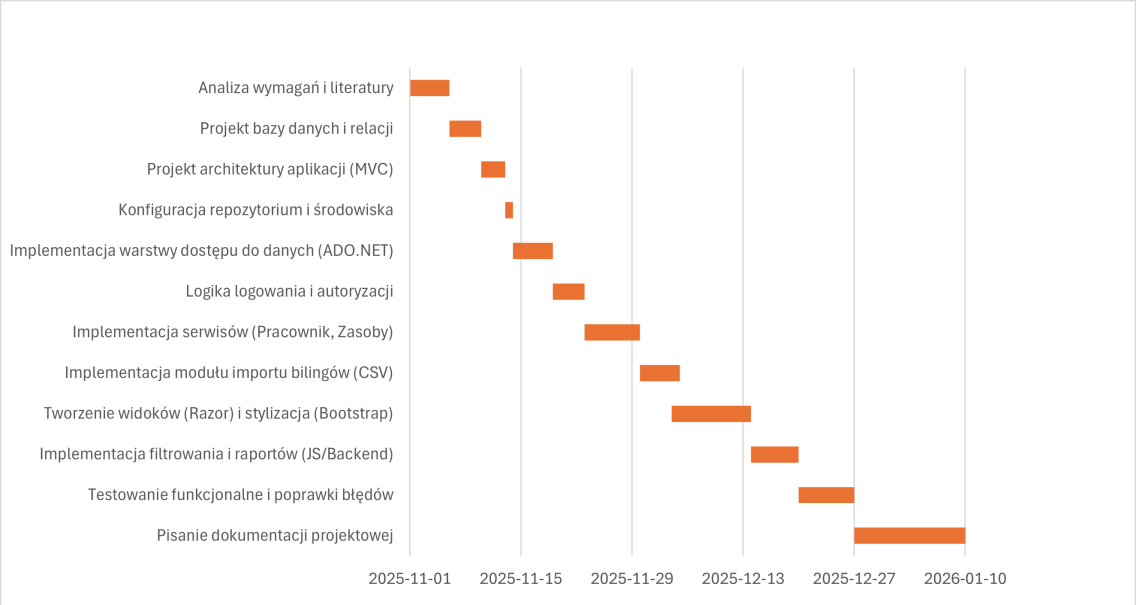
Rozdział 6

Harmonogram realizacji projektu

Realizacja projektu „SystemTelefonicznyGY” przebiegała zgodnie z przyjętym planem w okresie od 1 listopada 2025 roku do 10 stycznia 2026 roku. Prace zostały podzielone na etapy, obejmujące analizę, projektowanie, implementację, testy oraz opracowanie dokumentacji końcowej.

Poniższy wykres Gantta (Rysunek 6.1) obrazuje rozkład czasowy poszczególnych zadań.

Rysunek 6.1: Harmonogram realizacji projektu (Wykres Gantta). Źródło: Opracowanie własne.



6.1 Szczegółowy przebieg prac

Prace rozpoczęto od **fazy analitycznej** (pierwsza połowa listopada), w której skupiono się na analizie literatury, określeniu wymagań funkcjonalnych oraz zaprojektowaniu relacyjnej bazy danych i architektury MVC. Po skonfigurowaniu repozytorium kodu przystąpiono do właściwej implementacji.

W **drugiej połowie listopada** zrealizowano kluczowe elementy warstwy back-endowej: autorski mechanizm dostępu do danych (ADO.NET), logikę autoryzacji użytkowników oraz serwisy biznesowe (`PracownikService`, `ZasobyService`). Miesiąc zakończono implementacją jednego z trudniejszych modułów – importu i parsowania plików bilingowych CSV.

Grudzień poświęcono głównie na warstwę prezentacji (Front-end) oraz logikę raportową. W pierwszej połowie miesiąca stworzono widoki Razor wykorzystujące bibliotekę Bootstrap, a następnie zaimplementowano mechanizmy filtrowania danych. Ostatnia dekada grudnia oraz początek stycznia

zostały przeznaczone na testy funkcjonalne, poprawę wykrytych błędów oraz sporządzenie niniejszej dokumentacji projektowej.

Tabela 6.1 przedstawia szczegółowy wykaz zadań wraz z datami ich rozpoczęcia oraz czasem trwania.

Tabela 6.1: Szczegółowy harmonogram zadań projektowych

Realizowane zadanie	**Data rozp.**	**Dni**
Analiza wymagań i literatury	2025-11-01	5
Projekt bazy danych i relacji	2025-11-06	4
Projekt architektury aplikacji (MVC)	2025-11-10	3
Konfiguracja repozytorium i środowiska	2025-11-13	1
Implementacja warstwy dostępu do danych (ADO.NET)	2025-11-14	5
Logika logowania i autoryzacji	2025-11-19	4
Implementacja serwisów (Pracownik, Zasoby)	2025-11-23	7
Implementacja modułu importu bilingów (CSV)	2025-11-30	5
Tworzenie widoków (Razor) i stylizacja (Bootstrap)	2025-12-04	10
Implementacja filtrowania i raportów (JS/Backend)	2025-12-14	6
Testowanie funkcjonalne i poprawki błędów	2025-12-20	7
Pisanie dokumentacji projektowej	2025-12-27	14

.

Rozdział 7

Podsumowanie i wnioski

Celem niniejszej pracy zaliczeniowej było zaprojektowanie i zaimplementowanie systemu informatycznego „SystemTelefonicznyGY”, wspomagającego zarządzanie zasobami telekomunikacyjnymi oraz automatyzację analizy kosztów bilingowych w przedsiębiorstwie.

Główny cel pracy został w pełni osiągnięty. Stworzona aplikacja webowa w technologii ASP.NET MVC pozwala na efektywną ewidencję struktury organizacyjnej, pracowników oraz sprzętu IT (telefonów komórkowych, stacjonarnych i modemów). Kluczową funkcjonalnością systemu jest zautomatyzowany moduł importu plików CSV, który eliminuje konieczność ręcznego rozliczania kosztów połączeń, znacząco skracając czas pracy działów administracyjnych.

W trakcie realizacji projektu spełniono wszystkie założone wymagania funkcjonalne i techniczne, w tym:

- **Złożoność struktury obiektowej:** Zrealizowano wymóg projektowy dotyczący hierarchii klas składającej się z minimum 5 elementów. Zastosowanie polimorfizmu oraz klasy abstrakcyjnej `ElementSystemu` umożliwiło elastyczne zarządzanie różnymi typami urządzeń.

- **Architektura i Wzorce:** Aplikacja została zbudowana zgodnie ze wzorcem MVC (Model-View-Controller) z wydzieloną warstwą logiki biznesowej. Wykorzystanie interfejsów (Dependency Injection) zapewniło modułowość kodu i jego testowalność.

- **Wydajność:** Zastosowanie autorskiego wrappera ADO.NET zamiast ciężkich mechanizmów ORM w module importu pozwoliło na szybkie przetwarzanie dużych plików bilingowych przy minimalnym zużyciu pamięci RAM.

- **Bezpieczeństwo:** Zaimplementowano mechanizm autoryzacji oparty na sesji oraz podział ról (Administrator/Użytkownik), co skutecznie chroni dane wrażliwe przed niepowołanym dostępem.

Podczas realizacji projektu napotkano wyzwania inżynierskie, głównie związane z niespójnością formatów danych dostarczanych przez operatorów telekomunikacyjnych oraz koniecznością zachowania integralności relacyjnej bazy danych. Problemy te rozwiązano poprzez implementację zaawansowanych algorytmów parsowania dat i numerów oraz zastosowanie transakcji SQL przy operacjach usuwania powiązanych rekordów (np. pracowników posiadających przypisane aktywne urządzenia). Przeprowadzone testy funkcjonalne oraz weryfikacja spójności danych potwierdziły stabilność przyjętych rozwiązań.

Projekt „SystemTelefonicznyGY” posiada duży potencjał rozwojowy. Otwarta architektura systemu pozwala na jego dalszą rozbudowę, m.in. o:

- Moduł automatycznych powiadomień e-mail dla managerów w przypadku przekroczenia limitów kosztów przez podległych pracowników.
- Integrację z usługą katalogową Active Directory (LDAP) w celu centralizacji zarządzania tożsamością.
- API umożliwiające współpracę z zewnętrznymi systemami ERP lub aplikacją mobilną do podglądu kosztów w czasie rzeczywistym.

Podsumowując, stworzone oprogramowanie jest kompletnym, funkcjonalnym narzędziem, gotowym do wdrożenia w środowisku produkcyjnym, spełniającym standardy współczesnych aplikacji internetowych.

.

Bibliografia

[1] J. Matulewski, *C#: lekcje programowania: praktyczna nauka programowania dla platform .NET i .NET Core*, Helion, Gliwice 2021.

[2] R.S. Miles, *C#: zacznij programować!*, Helion, Gliwice 2020.

[3] I. Ben-Gan, *T-SQL Fundamentals*, Microsoft Press, 2016.

[4] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku*, Helion, Gliwice 2010.

[5] Microsoft, *Overview of ASP.NET Core MVC*, https://learn.microsoft.com/en-us/aspnet/core/mvc/overview z dnia 5.11.2025.

[6] W3Schools, *C# Tutorial*, https://www.w3schools.com/cs/index.php z dnia 01.11.2025.

.

Spis rysunków

2.1 Diagram klas projektu SystemTelefonicznyGY. Źródło: Opracowanie własne. 10

3.1 Diagram ERD bazy danych SystemTelefonicznyGY. Źródło: Opracowanie własne. 12
3.2 Wynik zapytania złączeniowego (JOIN). Źródło: Opracowanie własne. 17
3.3 Wynik zapytania agregującego. Źródło: Opracowanie własne. 18
3.4 Wynik aktualizacji rekordu (UPDATE) wraz z weryfikacją (SELECT). Źródło: Opracowanie własne. . . . 19

4.1 Implementacja metody importującej dane z CSV w BilingService.cs 21
4.2 Metoda logowania w KontoController.cs z użyciem serwisu 23
4.3 Test jednostkowy metody Login z wykorzystaniem biblioteki Moq 24
4.4 Podsumowanie wyników testów jednostkowych w Test Explorer (Visual Studio). 25
4.5 Efekt działania Scenariusza 1 – odmowa dostępu. 26
4.6 Przekierowanie do logowania przy próbie wejścia bez uprawnień. 26
4.7 Poprawny import danych bilingowych (Scenariusz 3). 27
4.8 Komunikat błędu przy próbie wgrania niedozwolonego pliku. 27
4.9 Walidacja formularza zmiany hasła. 28
4.10 Podsumowanie brutto w importowanym pliku. 28
4.11 Podsumowanie brutto w aplikacji. 28
4.12 Podsumowanie brutto w bazie danych. 29

5.1 Panel zarządzania pracownikami i edycji uprawnień. 30
5.2 Interfejs ewidencji zasobów sprzętowych. 31
5.3 Widok zarządzania strukturą organizacyjną firmy. 31
5.4 Moduł importu plików bilingowych CSV. 32
5.5 Narzędzie do filtrowania i generowania raportów kosztowych. 32
5.6 Widok „Panel Pracownika” w panelu pracownika. 33
5.7 Podgląd kosztów i historii połączeń dla zalogowanego użytkownika. 33

6.1 Harmonogram realizacji projektu (Wykres Gantta). Źródło: Opracowanie własne. 34

Wyższa Szkoła Informatyki i Zarządzania z siedzibą w Rzeszowie
Kolegium Informatyki Stosowanej

Streszczenie pracy zaliczeniowej
System telefoniczny

Autor: Paweł Drąg
Promotor: mgr inż. Ewa Żesławska
Słowa kluczowe: ASP.NET MVC, C#, SQL Server, System Bilingowy, Zarządzanie Zasobami IT, Aplikacja Webowa, ADO.NET.

Projekt „SystemTelefonicznyGY” to aplikacja webowa zrealizowana w technologii ASP.NET MVC, służąca do zarządzania zasobami telekomunikacyjnymi w przedsiębiorstwie. System umożliwia ewidencję pracowników, strukturę organizacyjną oraz zarządzanie urządzeniami. Kluczową funkcjonalnością jest import i analiza bilingów telefonicznych, co pozwala na automatyczne rozliczanie kosztów. Architektura aplikacji oparta jest na wzorcu MVC z wydzieloną warstwą serwisową i wykorzystaniem interfejsów, co zapewnia modularność i skalowalność rozwiązania.

The University of Information Technology and Management in Rzeszow
Faculty of Applied Information Technology

Thesis Summary
SystemTelefonicznyGY - Telecommunication Resource Management System

Author: Paweł Drąg
Supervisor: mgr inż. Ewa Żesławska
Key words: ASP.NET MVC, C#, SQL Server, Billing System, IT Resource Management, Web Application, ADO.NET.

The "SystemTelefonicznyGY"project is a web application implemented using ASP.NET MVC technology, designed for managing telecommunication resources within an enterprise. The system enables the records of employees, organizational structure, and device management. A key feature is the import and analysis of telephone billings, allowing for automatic cost accounting. The application architecture is based on the MVC pattern with a separated service layer and the use of interfaces, ensuring modularity and scalability of the solution.