

**POLITECHNIKA WROCŁAWSKA**  
**WYDZIAŁ ELEKTRONIKI**

---

**PROJEKT Z BAZ DANYCH**

**System obsługi połączeń kolejowych oparty o  
relacyjną bazę danych**

**AUTORZY:**

Piotr Chmiel

Indeks: 200608

Maciej Stelmaszuk

Indeks: 200654

**PROWADZĄCY ZAJĘCIA:**

Dr inż. Robert Wójcik, W4/I-6

**OCENA PRACY:**

## Spis treści

Spis tabel.....	5
Spis listingów .....	6
Spis rysunków .....	7
1. Wstęp .....	8
1.1. Cel i zakres projektu .....	8
1.2. Opis działania systemu .....	8
2. Analiza wymagań .....	8
2.1. Wymagania funkcjonalne .....	8
2.1.1. Diagram przypadków użycia .....	9
2.1.2. Scenariusze użycia .....	11
2.1.2.1. Nazwa PU: Załóż konto .....	11
2.1.2.2. Nazwa PU: Wyszukaj połączenie .....	12
2.1.2.3. Nazwa PU: Wyszukiwanie zaawansowane .....	13
2.1.2.4. Nazwa PU: Przegląd Dworców .....	14
2.1.2.5. Nazwa PU: Sesja.....	14
2.1.2.6. Nazwa PU: Logowanie.....	15
2.1.2.7. Nazwa PU: Edytuj profil .....	16
2.1.2.8. Nazwa PU: Kup Bilet .....	17
2.1.2.9. Nazwa PU: Wyświetl listę biletów.....	18
2.1.2.10. Nazwa PU: Wylogowanie .....	19
2.1.2.11. Nazwa PU: Sesja(Administrator) .....	20
2.1.2.12. Zbiorcze opracowanie PU: Zarządzanie Kontami Użytkowników, Zarządzanie Połączeniami, Zarządzanie Zasobami Ludzkimi, Zarządzanie Dworcami.....	21
2.1.2.13. Nazwa PU: Dodaj rekord.....	21
2.1.2.14. Nazwa PU: Edytuj rekord .....	22
2.1.2.15. Nazwa PU: Usuń rekord .....	23
2.2. Wymagania нефункционалне .....	23
2.2.1. Wybrane technologie i narzędzia.....	23
2.2.1.1. Python .....	23
2.2.1.2. Django.....	24
2.2.1.3. Pycharm.....	24

2.2.1.4.	Bootstrap .....	24
2.2.1.5.	Javascript .....	24
2.2.1.6.	jQuery .....	24
2.2.2.	Parametry wydajnościowe .....	25
2.2.2.1.	Serwer internetowy .....	25
2.2.2.2.	Baza danych .....	25
2.2.2.3.	Bezpieczeństwo .....	25
2.3.	Założenia przyjęte podczas realizacji systemu .....	26
2.4.	Charakterystyka wykorzystywanych technologii i narzędzi projektowych.....	26
2.4.1.	MySQL Serwer .....	26
2.4.2.	Apache .....	26
2.4.3.	Język UML .....	26
2.4.4.	Microsoft Visio 2013 .....	27
2.4.5.	Visual Paradigm Community Edition .....	27
2.4.6.	Test Driven Development .....	27
3.	Projekt bazy danych.....	27
3.1.	Model konceptualny - diagram ERD .....	27
3.2.	Model logiczny - diagram CDM .....	28
3.3.	Model fizyczny – diagram PDM.....	28
3.4.	Mechanizmy komunikacji i sterowania przepływem danych .....	29
3.4.1.	Schemat komunikacji struktura systemu .....	29
3.5.	Mechanizmy przetwarzania danych .....	29
4.	Projekt aplikacji.....	29
4.1.	Projekt graficzny interfejsu .....	29
4.2.	Struktura logiczna menu.....	30
4.3.	Sposób mapowania tabel .....	31
4.4.	Bezpieczeństwo aplikacji .....	32
4.4.1.	Kontrola dostępu do wybranych funkcjonalności aplikacji .....	32
4.4.2.	Ochrona przed atakami CSRF.....	34
4.4.3.	Ochrona przed atakami typu Clickjacking .....	35
4.4.4.	Ochrona przed SQL Injection .....	35
4.4.5.	Szyfrowanie i ochrona haseł użytkownika w bazie danych .....	37

<b>5. Implementacja bazy danych .....</b>	<b>37</b>
5.1. Generowanie kodu SQL .....	40
5.2. Realizacja ograniczeń integralnościowych .....	42
<b>6. Implementacja aplikacji .....</b>	<b>42</b>
6.1. Przepływ danych w Django .....	42
6.2. Implementacja wybranych funkcjonalności .....	43
6.2.1. Logowanie .....	43
6.2.2. Wylogowanie.....	44
6.2.3. Edycja profilu użytkownika .....	44
6.2.4. Zmiana hasła .....	45
6.2.5. Wyszukiwanie połączeń .....	45
6.2.6. Rejestracja nowego użytkownika .....	46
6.2.7. Funkcjonalność administratora.....	46
<b>7. Konfigurowanie i testowanie systemu .....</b>	<b>47</b>
7.1. Konfiguracja środowiska produkcyjnego .....	47
7.2. Konfiguracja środowiska testowego .....	49
7.3. Testy z wykorzystaniem narzędzia Selenium .....	49
7.4. Testy jednostkowe z wykorzystaniem biblioteki Djangounittest .....	51
7.5. Wydajność systemu .....	53
<b>8. Podsumowanie .....</b>	<b>54</b>
<b>Literatura .....</b>	<b>56</b>

## **Spis tabel**

- 1. Wyniki pomiaru wydajności serwera w stosunku do obciążenia ..... 54**

## Spis Listingów

1. Fragment pliku views.py - przykład zastosowania dekoratorów.....	30
2. Fragment szablonu base.html - kontrola dostępu .....	31
3. Fragment pliku settings.py .....	32
4. Fragment pliku forms.py - validatory i formularze .....	33
5. Zawartość pliku models.py .....	35
6. Polecenia użyte do implementacji bazy danych .....	38
7. Plik urls.py .....	41
8. Konfiguracja pliku httpd.conf .....	48
9. Fragment pliku tests.py - przykład zastosowania narzędzia Selenium .....	49
10. Fragment pliku tests.py - przykład zastosowania biblioteki djangounittest .....	51

## Spis rysunków

1. Diagram przypadków użycia cz.1.....	9
2. Diagram przypadków użycia cz.2.....	10
3. Diagram ERD .....	27
4. Diagram CDM .....	28
5. Diagram PDM .....	28
6. Schemat komunikacji .....	29
7. Model implementacyjny bazy danych Systemu Obsługi Połączeń Kolejowych .....	40
8. Strona główna Systemu Obsługi Połączeń Kolejowych .....	42
9. Drzewo zagnieżdżenia szablonów .....	43
10. Strona logowania do Systemu Obsługi Połączeń Kolejowych.....	43
11. Strona edycji profilu użytkownika .....	44
12. Strona zmiany hasła .....	45
13. Strona wyszukiwania połączeń.....	45
14. Strona wyników wyszukiwania .....	45
15. Formularz rejestracji nowego użytkownika.....	46
16. Panel administratora.....	46
17. Wyniki dla 2000 użytkowników w programie Apache JMeter.....	53
18. Średni czas odpowiedzi serwera w stosunku do liczby użytkowników .....	53

## **1. Wstęp**

### **1.1. Cel i zakres projektu**

Celem projektu jest zamodelowanie oraz implementacja aplikacji umożliwiającej dostęp z poziomu Internetu do bazy danych przeznaczonej do obsługi połączeń kolejowych.

### **1.2. Opis działania systemu**

System umożliwiać będzie zarządzanie połączeniami kolejowymi w oparciu o relacyjną bazę danych (tabele opisujące dane o połączeniach np. miasto początkowe, miasto docelowe, ilość miejsc, data połączenia). Dla bazy danych o znanej strukturze należy zrealizować dostęp do danych z poziomu przeglądarki internetowej za pomocą odpowiedniej aplikacji umieszczonej na serwerze pośredniczącym WWW. W zależności od poziomu uprawnień użytkownika system umożliwiać będzie wykonywanie następujących zadań.

#### **1.3. Lista głównych zadań:**

##### **1.3.1. Z poziomu administratora:**

- 1.3.1.1. Dodawanie połączeń.
- 1.3.1.2. Edytowanie połączeń.
- 1.3.1.3. Usuwanie połączeń.
- 1.3.1.4. Zarządzanie rezerwacjami.
- 1.3.1.5. Zarządzanie kontami użytkowników

##### **1.3.2. Z poziomu użytkownika:**

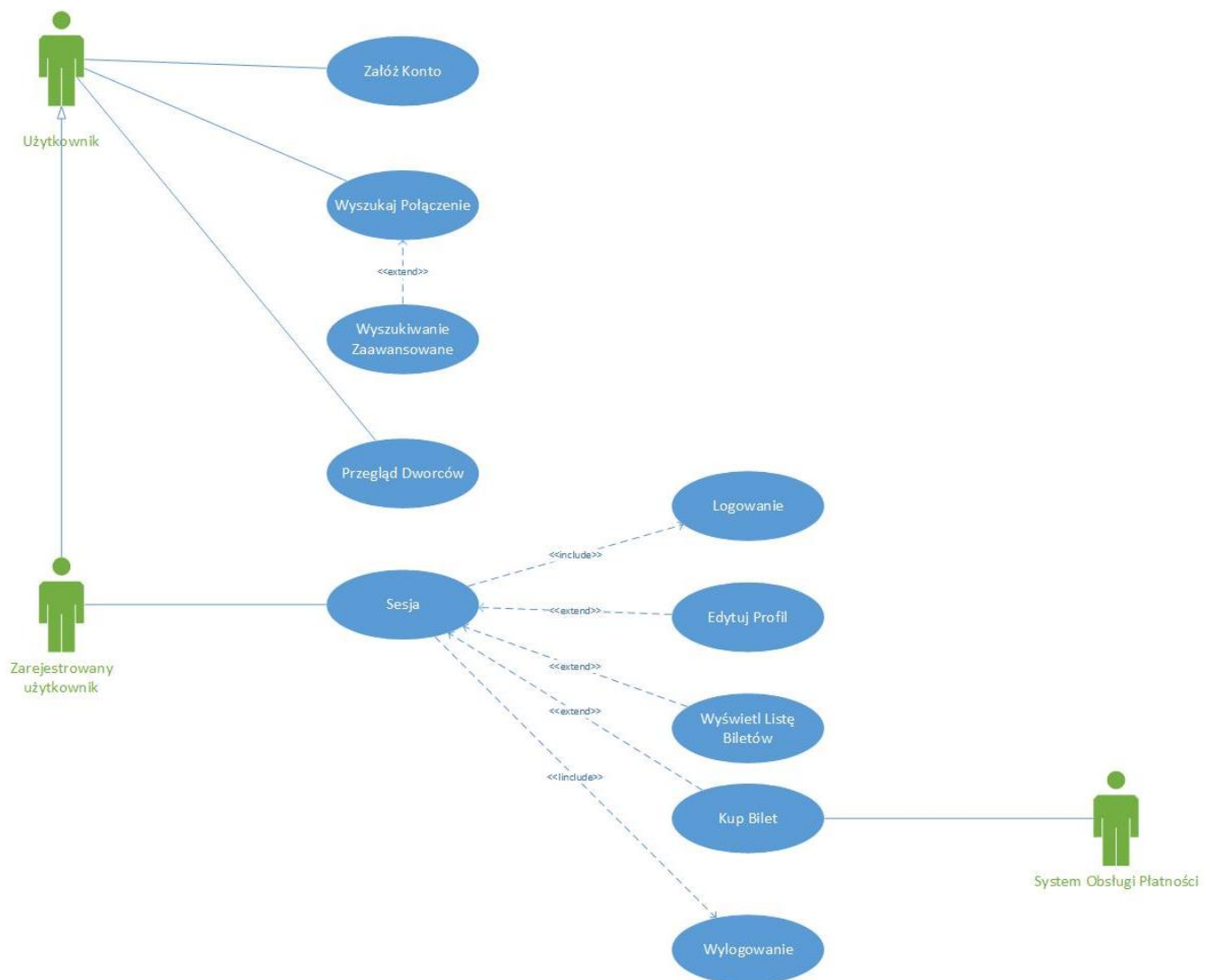
- 1.3.2.1. Rejestracja.
- 1.3.2.2. Logowanie i wylogowanie.
- 1.3.2.3. Wyszukiwanie połączeń.
- 1.3.2.4. Rezerwacja połączeń.
- 1.3.2.5. Anulowanie rezerwacji.
- 1.3.2.6. Przeglądanie dworców.
- 1.3.2.7. Edycja profilu.
- 1.3.2.8. Zmiana hasła.

## **2. Analiza wymagań**

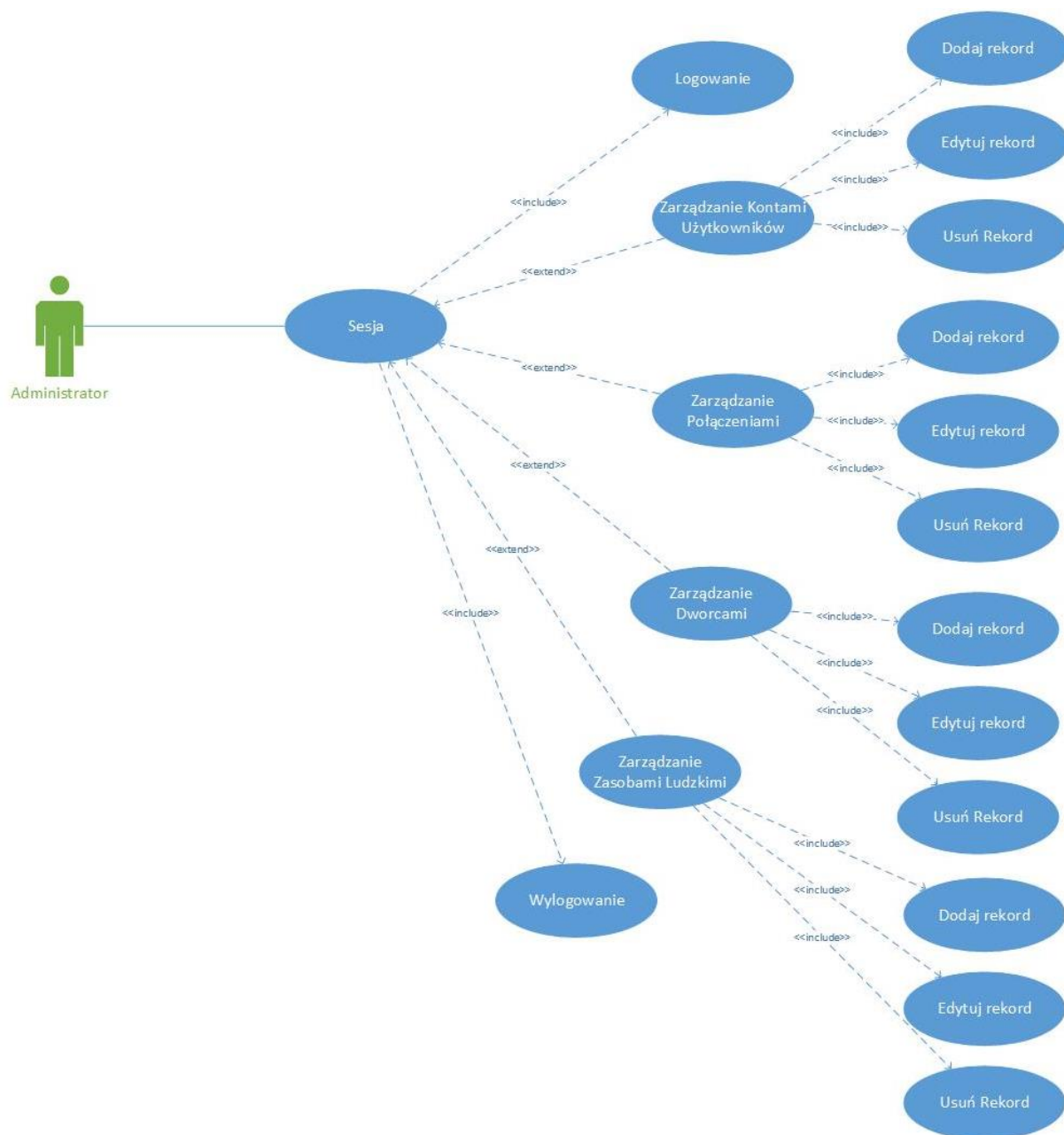
### **2.1. Wymagania funkcjonalne**



### 2.1.1. Diagram przypadków użycia



Rysunek 1. Diagram przypadków użycia cz.1



Rysunek 2. Diagram przypadków użycia cz.2

## 2.1.2. Scenariusze użycia

### 2.1.2.1. Nazwa PU: Załóż Konto

Cel	Rejestracja w systemie nowego użytkownika. Udostępnienie nowo zarejestrowanemu użytkownikowi możliwości zakupu biletu oraz przeglądania listy biletów.
Warunki Początkowe	Klient chce zostać nowym użytkownikiem systemu w celu zakupu biletu. Przypadek inicjuje aktor Użytkownik.
Warunki Końcowe	PU „Załącz Konto” zostaje zakończony. Nowy rekord z danymi podanymi podczas rejestracji zostaje dodany do bazy danych. Klient może zalogować się do systemu i korzystać z opcji dostępnych dla zarejestrowanych użytkowników
Przebieg PU	<ol style="list-style-type: none"><li>1. Użytkownik inicjuje PU wchodząc w zakładkę „Zarejestruj” dostępną na stronie głównej.</li><li>2. System wyświetla formularz rejestracji.</li><li>3. System oczekuje na wypełnienie formularza i wciśnięcie przycisku „Zarejestruj”.</li><li>4. Jeżeli system wykryje wciśnięcie przycisku „Zarejestruj” zostaje sprawdzana poprawność formularza”.</li><li>5. Jeżeli formularz został wypełniony niepoprawnie system wyświetla komunikat z rodzajem błędu, następuje skok do punktu 2. W przeciwnym wypadku przejście do punktu 6.</li><li>6. System zapisuje nowy rekord z danymi użytkownika do bazy danych.</li><li>7. System wyświetla komunikat z podziękowaniem za rejestrację oraz z linkiem przekierowującym do strony głównej.</li><li>8. Zakończenie PU.</li></ol>

### 2.1.2.2. Nazwa PU: Wyszukaj Połączenie

Cel	Wyświetlenie listy dostępnych połączeń w wybranym przez klienta terminie.
Warunki Początkowe	Klient chce sprawdzić dostępne połączenia w wybranym przez siebie terminie. Przypadek inicjuje aktor Użytkownik.
Warunki Końcowe	PU „Wyszukaj Połączenie” zostaje zakończony. Zostaje wyświetlona tabela z podstawowymi danymi połączeń wyszukiwanych przez klienta. Zostaje udostępniona możliwość sortowania tabeli.
Przebieg PU	<ol style="list-style-type: none"> <li>1. Użytkownik inicjuje PU wchodząc w zakładkę „Wyszukaj połączenie” dostępną na stronie głównej.</li> <li>2. System wyświetla formularz wyszukiwania.</li> <li>3. System oczekuje na wypełnienie formularza i wciśnięcie przycisku „Szukaj”.</li> <li>4. Jeżeli system wykryje wciśnięcie przycisku „Wyszukiwanie Zaawansowane” system inicjuje PU: „Wyszukiwanie Zaawansowane”</li> <li>5. Jeżeli został zainicjowany PU: „Wyszukiwanie Zaawansowane” system oczekuje na zakończenie wyżej wymienionego PU.</li> <li>6. Jeżeli system wykryje wciśnięcie przycisku „Szukaj” następuje sprawdzanie poprawności formularza.</li> <li>7. Jeżeli formularz został wypełniony niepoprawnie system wyświetla komunikat z rodzajem błędu, następuje skok do punktu 2. W przeciwnym wypadku przejście do punktu 8.</li> <li>8. System przetwarza zapytanie klienta i pobiera odpowiednie rekordy z bazy danych.</li> <li>9. System przekierowuje do strony z wynikami wyszukiwania.</li> <li>10. Jeżeli zbiór pobranych z bazy danych jest pusty system wyświetla komunikat o braku dostępnych połączeń, w</li> </ol>

	<p>przeciwnym wypadku zostaje wyświetlona tabela z danymi dotyczącymi dostępnych połączeń.</p> <p>11. Zakończenie PU.</p>
--	---

### 2.1.2.3. Wyszukiwanie Zaawansowane

Cel	<p>Rozszerzenie możliwości wyszukiwania podstawowego.</p> <p>Udostępnienie użytkownikowi zaawansowanych opcji filtrowania w celu wykonania wyszukiwania idealnie odpowiadającego potrzebom klienta.</p>
Warunki Początkowe	<p>Klient chce skonkretyzować wyszukiwanie. Przypadek inicjuje aktor Użytkownik podczas realizacji PU „Wyszukaj Połączenie”.</p>
Warunki Końcowe	<p>PU „Wyszukiwanie Zaawansowane” zostaje zakończony.</p> <p>Następuje dalszy ciąg PU „Wyszukaj Połączenie” w oparciu o filtr wybrany przez użytkownika w PU „ Wyszukiwanie zaawansowane”.</p>
Przebieg PU	<ol style="list-style-type: none"> <li>1. PU zostaje zainicjowany przez użytkownika podczas realizacji PU „Wyszukaj Połączenie”, jeżeli system wykryje wciśnięcie przycisku „Wyszukiwanie Zaawansowane”</li> <li>2. System wyświetla formularz z listą dostępnych filtrów oraz przypisanymi do nich checkboxami.</li> <li>3. System oczekuje na wypełnienie formularza w postaci zaznaczenia checkboxów oraz wciśnięcie przycisku „ok”.</li> <li>4. Jeżeli system zarejestruje wciśnięcie przycisku „anuluj”. Skok do punktu 6.</li> <li>5. Jeżeli system zarejestruje wciśnięcie przycisku „ok” filtry zostają ustawione Filtry zostają nałożone w dalszej realizacji PU „Wyszukaj Połączenie”.</li> <li>6. PU zostaje zakończony.</li> </ol>

#### 2.1.2.4. Nazwa PU: Przegląd Dworców

Cel	Dostarczenie informacji klientowi na temat wybranych dworców. Udostępnienie użytkownikowi danych dworca oraz zdjęć.
Warunki Początkowe	Klient potrzebuje informacji na temat wybranego dworca. Przypadek inicjuje aktor Użytkownik podczas realizacji PU „Przegląd Dworców”.
Warunki Końcowe	PU „Przegląd Dworców” zostaje zakończony. Zastaje wyświetlona strona z informacjami o wybranym dworcu.
Przebieg PU	<ol style="list-style-type: none"><li>1. Użytkownik inicjuje PU wchodząc w zakładkę „Dorce” dostępną na stronie głównej.</li><li>2. System wyświetla listę dostępnych dworców.</li><li>3. System oczekuje na kliknięcie na nazwę dworca na wyświetlonej liście.</li><li>4. Jeżeli system zarejestruje kliknięcie wykonane na jednej z nazw system pobiera dane dworca z bazy danych.</li><li>5. Następuje przekierowanie do strony z danymi oraz zdjęciem wybranego dworca.</li><li>6. Zakończenie PU.</li></ol>

#### 2.1.3. Nazwa PU: Sesja

Cel	Udostępnienie zarejestrowanemu użytkownikowi możliwości edycji profilu, zakupu oraz zwrotu biletu, a także wyświetlenia listy zakupionych biletów.
Warunki Początkowe	Użytkownik potrzebuje skorzystać z opcji dostępnych dla zarejestrowanych klientów. PU inicjuje aktor Zarejestrowany użytkownik.
Warunki Końcowe	PU „Sesja” zostaje zakończony. Upřednio PU „Wylogowanie” zakończony pomyślnie.
Przebieg PU	<ol style="list-style-type: none"><li>1. Zarejestrowany użytkownik inicjuje PU na stronie głównej inicjując jednocześnie PU „Logowanie”.</li></ol>

	<ol style="list-style-type: none"> <li>2. Jeżeli PU „Logowanie” zostaje zakończone sukcesem przejście do punktu 3, w przeciwnym razie skok do punktu 1.</li> <li>3. System udostępnia w menu na stronie głównej nowe zakładki „Edycja Profilu”, „Moje Bilety”, „Kup Bilet”.</li> <li>4. System oczekuje na kliknięcie na nazwę jednej z zakładek „Edycja Profilu”, „Moje Bilety”, „Kup Bilet”, jednocześnie system oczekuje na wciśnięcie przycisku wyloguj.</li> <li>5. Jeżeli zostanie wybrana zakładka „Edycja Profilu” system inicjuje PU „Edytuj Profil”. Skok do punktu 4.</li> <li>6. Jeżeli zostanie wybrana zakładka „Moje bilety” system inicjuje PU „Wyświetl listę biletów”. Skok do punktu 4.</li> <li>7. Jeżeli zostanie wybrana zakładka „Kup Bilet” system inicjuje PU „Kup Bilet”. Skok do punktu 4.</li> <li>8. Jeżeli system zarejestruje wciśnięcie przycisku „Wyloguj” system inicjuje PU „Wylogowanie”.</li> <li>9. Jeżeli PU „Wylogowanie zakończony sukcesem” PU „Sesja” zostaje zakończony.</li> </ol>
--	---

#### 2.1.4. Nazwa PU: Logowanie

Cel	Weryfikacja tożsamości klienta. Utworzenie nowej sesji. Umożliwienie dalszego przebiegu PU „Sesja”.
Warunki Początkowe	Klient chce zalogować się do systemu. Przypadek inicjuje aktor Zarejestrowany Użytkownik.
Warunki Końcowe	PU „Logowanie” zostaje zakończony. Następuje ponowne przekierowanie do strony głównej.
Przebieg PU	<ol style="list-style-type: none"> <li>1. Zarejestrowany użytkownik inicjuje PU wchodząc na stronę główną podczas realizacji PU „Sesja”.</li> <li>2. System oczekuje na wpisanie danych użytkownika w polach login i hasło oraz na wciśnięcie przycisku zaloguj.</li> </ol>

	<ol style="list-style-type: none"> <li>3. Jeżeli system wykrył wciśnięcie przycisku zaloguj system autoryzuje użytkownika.</li> <li>4. Jeżeli autoryzacja przebiegła pomyślnie skok do punktu 6, w przeciwnym razie przejście do punktu 5.</li> <li>5. Następuje przekierowanie do strony logowania oraz wyświetlenie komunikatu błędu, skok do punktu 2.</li> <li>6. System rejestruje nową sesję.</li> <li>7. System ponownie przekierowuje na stronę główną.</li> <li>8. W prawym górnym rogu zamiast pól logowania i przycisku „zaloguj” system tworzy przycisk „wyloguj” oraz nazwę użytkownika.</li> <li>9. Zakończenie PU.</li> </ol>
--	--

#### 2.1.2.5. Nazwa PU: Edytuj Profil

Cel	Aktualizacja danych klienta.
Warunki Początkowe	Klient chce zmienić swoje dane w systemie. PU inicjuje aktor „Zarejestrowany użytkownik” podczas realizacji PU „Sesja”.
Warunki Końcowe	PU „Edytuj Profil” zostaje zakończony. Zmiany zostały zapisane w bazie danych
Przebieg PU	<ol style="list-style-type: none"> <li>1. Zarejestrowany użytkownik inicjuje PU klikając na zakładkę „Edytuj Profil” podczas realizacji PU „Sesja”.</li> <li>2. System pobiera dane użytkownika z bazy danych.</li> <li>3. System wyświetla formularz z wypełnionymi danymi użytkownika oraz linki do zakładek „Zmień hasło” oraz „Powrót na stronę główną”.</li> <li>4. System oczekuje na wciśnięcie przycisku zapisz oraz na kliknięcie w zakładkę „Zmień hasło”.</li> <li>5. Jeżeli system wykrył wciśnięcie przycisku „Zapisz” następuje sprawdzanie poprawności formularza.</li> <li>6. Jeżeli formularz wypełniony poprawnie skok następuje aktualizacja danych użytkownika w bazie danych oraz</li> </ol>



	<p>skok do punktu 2. W przeciwnym razie wyświetlenie komunikatu o błędzie oraz skok do punktu 3.</p> <p>7. Jeżeli system wykryje kliknięcie na zakładkę „Zmień hasło” następuje przekierowanie do strony „Zmień hasło”.</p> <p>8. System wyświetla formularz zmiany hasła.</p> <p>9. System oczekuje na wciśnięcie przycisku „Zmień hasło” lub kliknięcie na jedną z zakładek: „Edytuj Profil” lub „Powrót na stronę główną”.</p> <p>10. Jeżeli został wciśnięty przycisk „Zmień hasło”, następuje weryfikacja poprawności formularza.</p> <p>11. Jeżeli formularz został wypełniony poprawnie, system aktualizuje rekord z danymi użytkownika w bazie danych oraz wyświetlenie komunikatu o sukcesie z linkami do zakładek „Edytuj Profil” oraz „Powrót na stronę główną” oraz W przeciwnym wypadku następuje wyświetlenie komunikatu z błędem oraz skok do punktu 9.</p> <p>12. Jeżeli system zarejestruje kliknięcie na zakładkę „Edytuj Profil” skok do punktu 2.</p> <p>13. Jeżeli system zarejestruje kliknięcie na zakładkę „Powrót do na stronę główną” przejście do punktu 14.</p> <p>14. Zakończenie PU.</p>
--	--

#### 2.1.2.6. Nazwa PU: Kup Bilet

Cel	Sprzedaż biletu na wybrane połączenie.
Warunki Początkowe	Klient chce zakupić nowy bilet. Przypadek inicjuje aktor Zarejestrowany Użytkownik.
Warunki Końcowe	PU „Kup Bilet” zostaje zakończony. Następuje przekierowanie do strony głównej.
Przebieg PU	1. Zarejestrowany użytkownik inicjuje PU klikając na zakładkę „Kup Bilet” podczas realizacji PU „Sesja”.

	<ol style="list-style-type: none"> <li>2. Zostaje zainicjowany PU: „Wyszukaj Połączenie” w celu znalezienia dostępnych połączeń.</li> <li>3. Jeżeli PU: „Wyszukaj Połączenie” zwrócił wyniki w postaci tabeli obok rekordu system generuje przycisk „Kup bilet”. W przeciwnym razie skok do punktu</li> <li>4. System oczekuje na wciśnięcie przycisku „Kup Bilet”.</li> <li>5. Jeżeli został wciśnięty przycisk „Kup Bilet” system wyświetla okno typu pop-up z prośbą o potwierdzenie zakupu oraz checkboxem do zaakceptowania regulaminu.</li> <li>6. Jeżeli system wykryje potwierdzenie zakupu. Następuje przekierowanie do systemu płatności. Jednocześnie system zmniejsza liczbę wolnych miejsc w bazie danych</li> <li>7. Jeżeli przyjdzie potwierdzenie do systemu od serwisu z płatnościami system generuje nowy bilet. Następuje zapisanie nowego rekordu z danymi biletu w bazie danych. System wysyła bilet w postaci pdf na adres e-mail użytkownika. W przeciwnym wypadku system ponownie dodaje wolne miejsce w bazie danych.</li> <li>8. Przekierowanie do strony głównej.</li> <li>9. Zakończenie PU</li> </ol>
--	--

### 2.1.3. Nazwa PU: Wyświetl listę biletów

Cel	Wyświetlenie historii zakupionych biletów
Warunki Początkowe	Klient chce wyświetlić historię zakupionych biletów, pobrać pdf z biletem, wyświetlić szczegółowe dane biletów. Przypadek inicjuje aktor Zarejestrowany Użytkownik.
Warunki Końcowe	PU „Wyświetl listę biletów” zostaje zakończony. Następuje przekierowanie do strony głównej.
Przebieg PU	<ol style="list-style-type: none"> <li>1. Zarejestrowany użytkownik inicjuje PU klikając na zakładkę „Moje Bilety” podczas realizacji PU „Sesja”.</li> </ol>

	<ol style="list-style-type: none"> <li>2. System pobiera z bazy danych bilety przypisane do użytkownika.</li> <li>3. System wyświetla listę biletów postaci tabeli.</li> <li>4. System oczekuje na dwukrotne kliknięcie na rekord tabeli lub kliknięcie jednego z przycisków: „eksportuj do pdf” lub „powrót do strony głównej”.</li> <li>5. Jeżeli system wykryje podwójne kliknięcie na rekord tabeli wyświetla okno typu pop-up ze szczegółami biletu. Skok do punktu 4.</li> <li>6. Jeżeli system wykryje wciśnięcie przycisku „eksportuj do pdf” system generuje pdf i rozpoczyna wysyłanie pliku do użytkownika. Skok do punktu 4.</li> <li>7. Jeżeli system wykryje wciśnięcie przycisku „powrót do strony głównej” następuje przekierowanie do strony głównej.</li> <li>8. Zakończenie PU.</li> </ol>
--	---

#### 2.1.2.7. Nazwa PU: Wylogowanie

Cel	Zamknięcie sesji użytkownika
Warunki Początkowe	Zalogowany użytkownik chce zakończyć korzystanie z serwisu. Przypadek inicjuje aktor Zarejestrowany Użytkownik.
Warunki Końcowe	PU „Wylogowanie” zostaje zakończony. Następuje przekierowanie do strony głównej. Sesja zostaje zamknięta.
Przebieg PU	<ol style="list-style-type: none"> <li>1. Zarejestrowany użytkownik inicjuje PU klikając na przycisk „Wyloguj” podczas realizacji PU „Sesja”.</li> <li>2. System zamyka sesję użytkownika.</li> <li>3. Następuje przekierowanie do strony głównej.</li> <li>4. Zakończenie PU.</li> </ol>

#### 2.1.2.8. Nazwa PU: Sesja (Administrator)

Cel	Udostępnienie zarejestrowanemu użytkownikowi możliwości edycji tabel w bazie danych
Warunki Początkowe	Administrator chce wprowadzić zmiany w bazie danych lub przeglądać jej zawartość. PU inicjuje aktor „Administrator” podczas realizacji PU „Sesja”.
Warunki Końcowe	PU „Sesja” zostaje zakończony. Uprzednio PU „Wylogowanie” zakończony pomyślnie.
Przebieg PU	<ol style="list-style-type: none"><li>1. Administrator inicjuje PU na stronie głównej inicjując jednocześnie PU „Logowanie”.</li><li>2. Jeżeli PU „Logowanie” zostaje zakończone sukcesem przejście do punktu 3, w przeciwnym razie skok do punktu 1.</li><li>3. System wyświetla listę dostępnych opcji na stronie administratora.</li><li>4. System oczekuje na kliknięcie na nazwę jednej z zakładek „Użytkownicy”, „Bilety”, „Dworce”, „Połączenia”, „Pracownicy”, „Profile Użytkowników”, „Stanowiska”, „Taryfy” jednocześnie system oczekuje na wciśnięcie przycisku „wyloguj”.</li><li>5. Jeżeli zostanie wybrana jedna z zakładek system inicjuje przypadek użycia związany z zarządzaniem wybraną tabelą. Skok do punktu 4.</li><li>6. Jeżeli system zarejestruje wciśnięcie przycisku „Wyloguj” system inicjuje PU „Wylogowanie”.</li><li>7. Jeżeli PU „Wylogowanie” zakończony sukcesem PU „Sesja” zostaje zakończony.</li></ol>

**2.1.2.9. Zbiorcze opracowanie PU: Zarządzanie Kontami Użytkowników, Zarządzanie Połączeniami, Zarządzanie Zasobami Ludzkimi, Zarządzanie Dworcami**

Cel	Modyfikacja wybranego segmentu bazy danych
Warunki Początkowe	Administrator chce wprowadzić zmiany w wybranej tabeli w bazie danych. PU inicjuje aktor „Administrator” podczas realizacji PU „Sesja”.
Warunki Końcowe	Przekierowanie do strony głównej administratora.
Przebieg PU	<ol style="list-style-type: none"> <li>1. Zarejestrowany użytkownik inicjuje PU klikając na wybraną zakładkę na stronie administratora podczas realizacji PU „Sesja”.</li> <li>2. System pobiera z bazy danych zawartość wybranej tabeli.</li> <li>3. System wyświetla listę rekordów dostępnych w tabeli.</li> <li>4. Jeżeli system wykryje wciśnięcie przycisku „Dodaj ...” inicjuje PU „Dodaj Rekord”. Skok do punktu 2.</li> <li>5. Jeżeli system wykryje kliknięcie na rekordzie tabeli inicjuje PU „Edytuj Rekord”. Skok do punktu 2.</li> <li>6. Jeżeli system wykryje wybór akcji „Usuń wybrane elementy” inicjuje PU „Usuń Rekord” dla wszystkich zaznaczonych w checkboxach rekordów. Skok do punktu 2.</li> <li>7. Jeżeli system wykryje kliknięcie na zakładce „Strona Administratora” system przekierowuje do strony administratora.</li> <li>8. Zakończenie PU.</li> </ol>

**2.1.2.10. Nazwa PU: Dodaj rekord**

Cel	Dodanie nowego rekordu do tabeli
-----	----------------------------------

Warunki Początkowe	Administrator chce dodać nowy rekord do bazy danych. PU inicjuje aktor „Administrator” podczas realizacji PU opisanego w punkcie 1.12.
Warunki Końcowe	PU „Dodaj rekord” zostaje zakończony. Nowy rekord został dodany do tabeli.
Przebieg PU	<ol style="list-style-type: none"> <li>1. Administrator inicjuje PU klikając na zakładkę „Dodaj ..” podczas realizacji PU opisanego w punkcie 1.12.</li> <li>2. System wyświetla formularz dodania nowego rekordu.</li> <li>3. Systemu oczekuje na wciśnięcie przycisku „zapisz”.</li> <li>4. Jeżeli przycisk „zapisz” został wciśnięty, następuje weryfikacja formularza.</li> <li>5. Jeżeli formularz został wypełniony poprawnie system zapisuje nowy rekord w bazie danych. W przeciwnym razie zostaje wyświetlony komunikat o błędach. Skok do punktu 2.</li> <li>6. Zakończenie PU.</li> </ol>

#### 2.1.2.11. Nazwa PU: Edytuj rekord

Cel	Edycja rekordu tabeli.
Warunki Początkowe	Administrator chce zaktualizować tabelę w bazie danych. PU inicjuje aktor „Administrator” podczas realizacji PU opisanego w punkcie 1.12.
Warunki Końcowe	PU „Dodaj rekord” zostaje zakończony. Nowy rekord został dodany do tabeli.
Przebieg PU	<ol style="list-style-type: none"> <li>1. Administrator inicjuje PU klikając na rekord tabeli podczas realizacji PU opisanego w punkcie 1.12.</li> <li>2. System pobiera wybranego rekordu z bazy danych.</li> <li>3. System wyświetla formularz z wypełnionymi danymi.</li> <li>4. Jeżeli formularz wypełniony poprawnie skok następuje aktualizacja danych użytkownika w bazie danych. W</li> </ol>

	<p>przeciwnym razie wyświetlenie komunikatu o błędzie oraz skok do punktu 2.</p> <p>5. Zakończenie PU.</p>
--	--

### 2.1.3. Nazwa PU: Usuń rekord

Cel	Usunięcie zbędnego rekordu
Warunki Początkowe	Administrator chce usunąć wybrany rekord tabeli w bazie danych. PU inicjuje aktor „Administrator” podczas realizacji PU opisanego w punkcie 1.12.
Warunki Końcowe	PU „Dodaj rekord” zostaje zakończony. Nowy rekord został dodany do tabeli.
Przebieg PU	<ol style="list-style-type: none"> <li>1. Administrator inicjuje PU wybierając akcję „Usuń wybrane elementy” podczas realizacji PU opisanego w punkcie 1.12.</li> <li>2. System odwołuje się do wybranego rekordu w bazie danych.</li> <li>3. System usuwa rekord z bazy danych.</li> <li>4. Zakończenie PU.</li> </ol>

## 2.2. Wymagania нефunkcjonalne

### 2.2.1. Technologie i narzędzia

Projekt Systemu Obsługi Połączeń Kolejowych zakłada napisanie aplikacji webowej. Z tego względu wybraliśmy najnowocześniejsze technologie i narzędzia, które umożliwiły nam napisanie wydajnej aplikacji.

#### 2.2.1.1. Python

Python to język programowania wysokiego poziomu ogólnego przeznaczenia, o rozbudowanym pakiecie bibliotek standardowych, którego ideą przewodnią jest czytelność i klarowność kodu źródłowego. Jego składnia cechuje się przejrzystością i zwięzłością. Python wspiera różne paradygmaty programowania: obiektowy, imperatywny oraz w mniejszym stopniu funkcyjny. Posiada w pełni dynamiczny system typów i automatyczne zarządzanie pamięcią. Podobnie jak inne języki dynamiczne jest często używany, jako język skryptowy.

Interpretery Pythona są dostępne na wiele systemów operacyjnych. Python rozwijany jest, jako projekt Open Source zarządzany przez Python Software Foundation, która jest organizacją non-profit.

#### **2.2.1.2. Django**

Django to wysokopoziomowy, opensource'owy framework przeznaczony do tworzenia aplikacji internetowych, napisany w Pythonie. Powstał pod koniec 2003 roku, jako ewolucyjne rozwinięcie aplikacji internetowych, tworzonych przez grupę programistów związanych z Lawrence Journal-World. W 2005 roku kod Django został wydany na licencji BSD. Nazwa frameworku pochodzi od imienia gitarzysty Django Reinhardta. Django opiera się na wzorcu projektowym podobnym do MVC nazywanym MVT (Model-View-Template).

#### **2.2.1.3. Pycharm**

Pycharm to IDE dla języka Python ze szczególnym wsparciem dla tworzenia aplikacji webowych z wykorzystaniem frameworku Django.

#### **2.2.1.4. Bootstrap**

Twitter Bootstrap to framework CSS, rozwijany przez programistów Twittera, wydawany na licencji MIT. Zawiera zestaw przydatnych narzędzi ułatwiających tworzenie interfejsu graficznego [stron](#) oraz aplikacji internetowych. Bazuje głównie na gotowych rozwiązaniach HTML oraz CSS (kompilowanych z plików Less) i może być stosowana m.in. do stylizacji takich elementów jak teksty, formularze, przyciski, wykresy, nawigacje i innych komponentów wyświetlanych na stronie. Framework korzysta także z języka JavaScript.

#### **2.2.1.5. Javascript**

JavaScript, JS to skryptowy język programowania, stworzony przez firmę Netscape, najczęściej stosowany na stronach internetowych. Pod koniec lat 90. XX wieku organizacja ECMA wydała na podstawie JavaScriptu standard języka skryptowego o nazwie ECMAScript. Głównym autorem JavaScriptu jest Brendan Eich.

#### **2.2.1.6. jQuery**

jQuery to lekka biblioteka programistyczna dla języka JavaScript, ułatwiająca korzystanie z JavaScriptu kosztem niewielkiego spadku wydajności w stosunku do



profesjonalnie napisanego kodu w niewspomagany JavaScript pozwala osiągnąć interesujące efekty animacji, dodać dynamiczne zmiany strony, wykonać zapytania AJAX.

## **2.2.2. Parametry Wydajnościowe**

### **2.2.2.1. Serwer Internetowy**

Docelowo sprzętem realizującym usługi sieciowe w środowisku produkcyjnym będzie serwer Serwer HP ProLiant DL320e Gen8. Parametry:

- Procesor: 1 x 4-Core Intel Xeon E3-1220v2 3.10 GHz
- Pamięć RAM: 16GB (1x8GB) DDR3 1333MHz ECC UDIMM
- Dyski Twarde: 2 x HP 1TB SATA\_7,2k obr/min 3,5 Hot Plug
- Poziom Raid: 0,1,10

Cena: ok 4000 zł.

Szacowana liczba jednocześnie korzystających użytkowników: 2000

Zakładany czas odpowiedzi serwera: poniżej 200 ms

Zakładane opóźnienie serwera (latency): poniżej 117 ms

### **2.2.2.2. Baza Danych**

Przyjęte zostały założenia, co do wydajności bazy danych:

Szacowana liczba rekordów bazy danych w środowisku produkcyjnym: 3 000 000 (dane o starszych połączeniach usuwane)

Oczekiwany czas oczekiwania na odczyt pojedynczego bloku z pliku danych: mniejszy niż 5 ms.

Pamięć serwera używana przez menadżera bazy danych: 40 %

Zakładana liczba transakcji w bazie danych uruchamianych w ciągu minuty: 300

Przeciętna liczba użytkowników jednocześnie łącząca się z bazą danych: 200

### **2.2.2.3. Bezpieczeństwo**

Aplikacja powinna zapewniać bezpieczeństwo przed większością typowych zagrożeń internetowych:

- SQL Injection;
- Cross Site Request Forgery;
- Clickjacking;

Co więcej aplikacja powinna zapewniać kontrolę dostępu dla poszczególnych grup użytkowników zgodnie z przypadkami użycia przedstawionymi na diagramie. Hasła

użytkownika powinny być w bazie danych zaszyfrowane i niedostępne dla innych użytkowników. Szerszy opis funkcji bezpieczeństwa znajduje się w rozdziale „Bezpieczeństwo Aplikacji”.

### **2.3. Założenia przyjęte podczas realizacji systemu**

W projekcie zastosujemy 3-warstwowy model komunikacji klient/serwer. Przetwarzanie danych realizowane będzie po stronie serwera aplikacji natomiast zarządzanie danymi po stronie serwera bazy danych MySQL. Do prezentacji danych użyjemy przeglądarki WWW. Dostęp do bazy danych realizowany będzie w oparciu o funkcje Django framework i Python, które komunikują się bezpośrednio z systemem bazy danych.

### **2.4. Charakterystyka wykorzystywanych technologii i narzędzi projektowania**

#### **2.4.1. MySQL Server**

Serwer bazy danych zdecydowaliśmy się zrealizować w oparciu o środowisko MySQL. MySQL to wolnodostępny system zarządzania relacyjnymi bazami danych. Pakiet oprogramowania MySQL Server Community w wersji 5.6 składa się z oprogramowania do zarządzania bazą danych, serwera, na którym umieszczona zostanie baza danych oraz narzędzia mysql.connector umożliwiające frameworkowi Django komunikację z bazą danych.

#### **2.4.2. Apache**

Serwer aplikacji zdecydowaliśmy zrealizować w oparciu o oprogramowanie Apache 2.2 Apache jest najszerszej stosowanym serwerem HTTP w Internecie. W lutym 2014 jego udział wśród serwerów wynosił 38%. Serwer wspiera wiele systemów operacyjnych (m.in. UNIX, GNU/Linux, BSD, OS X, Microsoft Windows).

#### **2.4.3. Język UML**

W fazie projektowania do modelowania systemu wykorzystaliśmy język UML. Język pozwala tworzyć modele systemów informatycznych. Obecnie jest to najbardziej rozbudowany standard modelowania, a w jego skład wchodzi 13 rodzajów diagramów.

#### 2.4.4. Microsoft VISIO 2013

Narzędziem, które wykorzystaliśmy do projektowania systemu był program Microsoft Visio 2013. Z jego pomocą skonstruowaliśmy diagram przypadków użycia. Umożliwia on szybką edycję diagramu oraz wyeksportowanie nowego diagramu do pliku.

#### 2.4.5. Visual Paradigm Community Edition

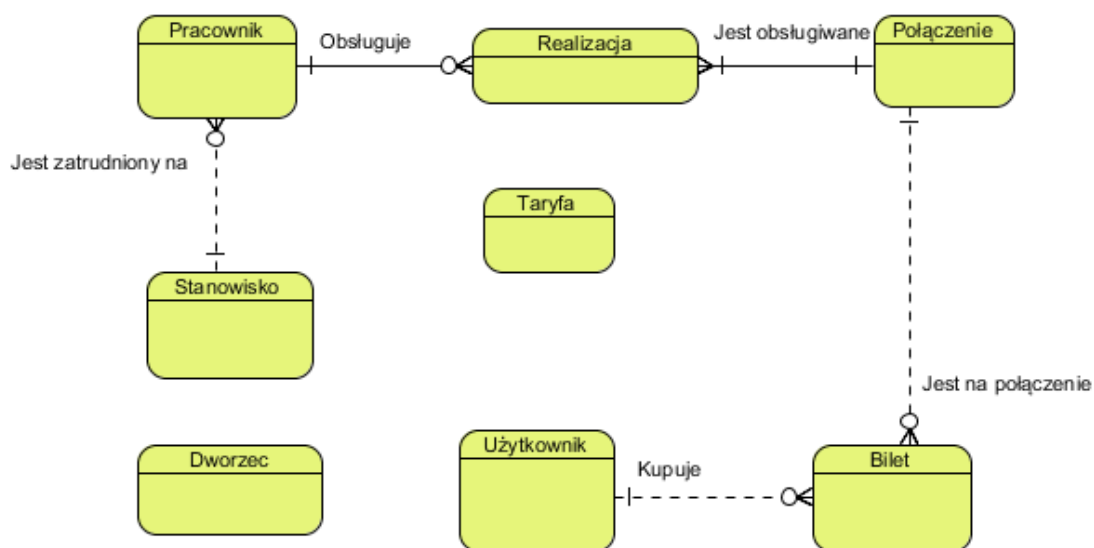
Kolejnym narzędziem umożliwiającym nam modelowanie systemu był program Visual Paradigm Community Edition. Korzystając z tej aplikacji narysowaliśmy większość diagramów w języku UML.

#### 2.4.6. Test Driven Development

Projekt napisaliśmy w oparciu o sposób tworzenia oprogramowania TDD. Jest to technika zaliczona do metodyk zwinnych Agile. Implementacja nowych funkcjonalności polega tutaj na pisaniu testów. Zanim napiszemy nową funkcjonalność piszemy test, który ją sprawdza. Początkowo test nie powinien się udać. Później implementujemy funkcjonalność oraz ponownie uruchamiamy wcześniej napisany test. Tym razem test powinien się udać. W ostatnim kroku dokonujemy refaktoryzacji napisanego kodu tak, żeby spełniał on oczekiwane standardy.

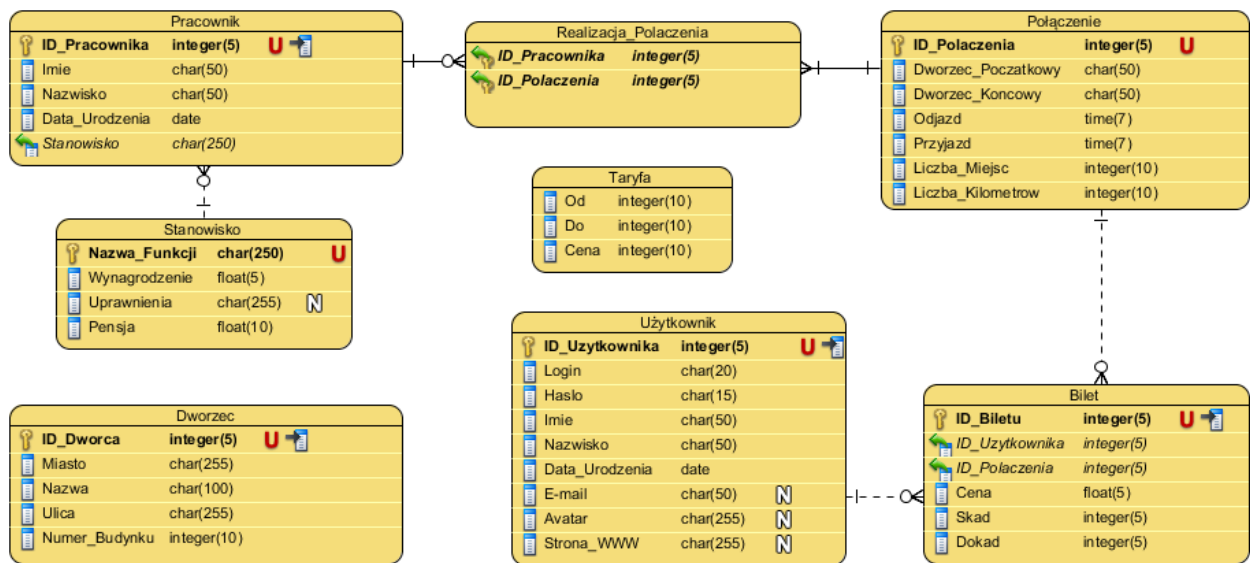
### 3. Projekt bazy danych

#### 3.1. Model konceptualny - diagram ERD



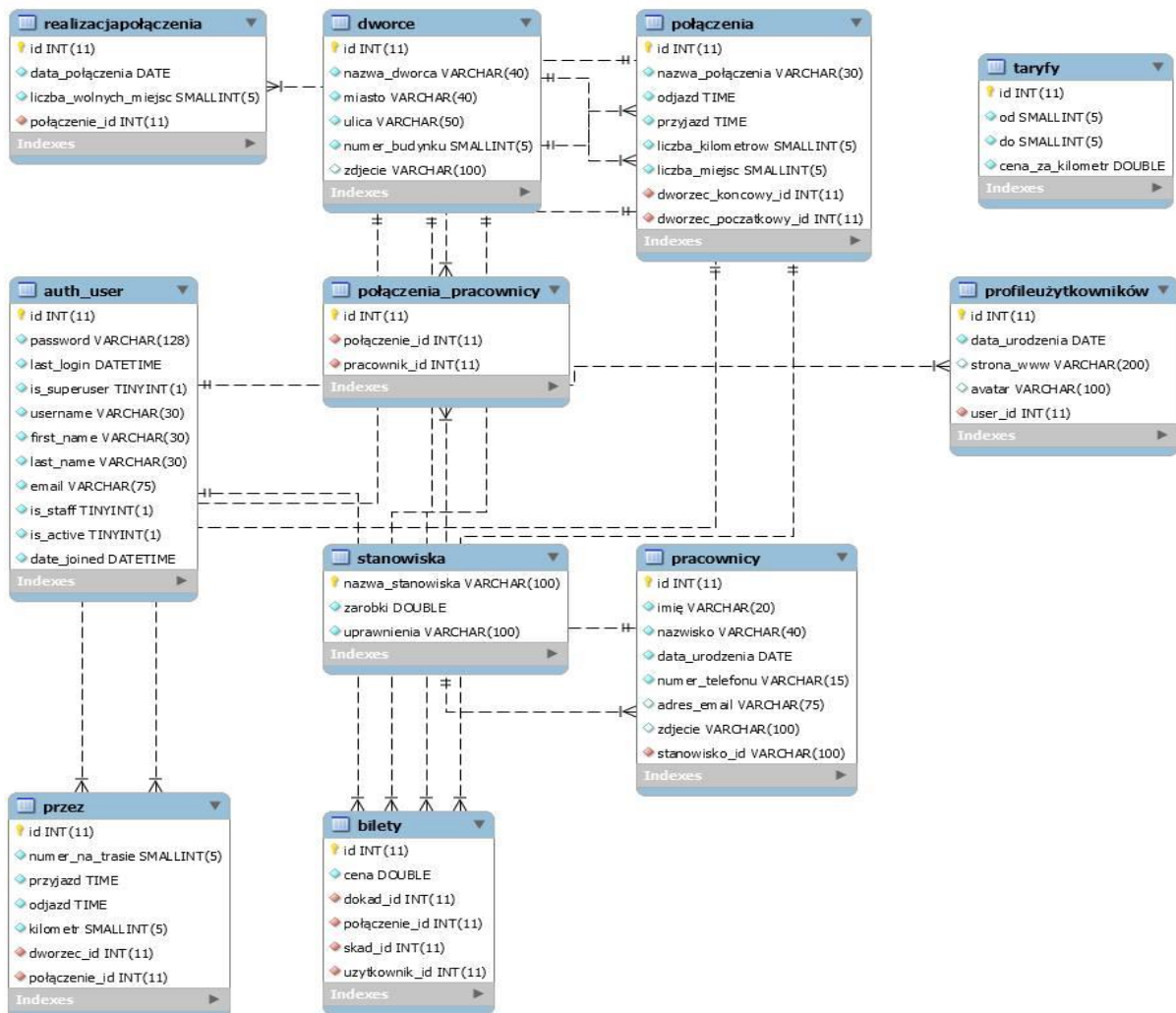
Rysunek 3. Diagram ERD

### 3.2. Model logiczny - diagram CDM



Rysunek 4. Diagram CDM

### 3.3. Model fizyczny - diagram PDM

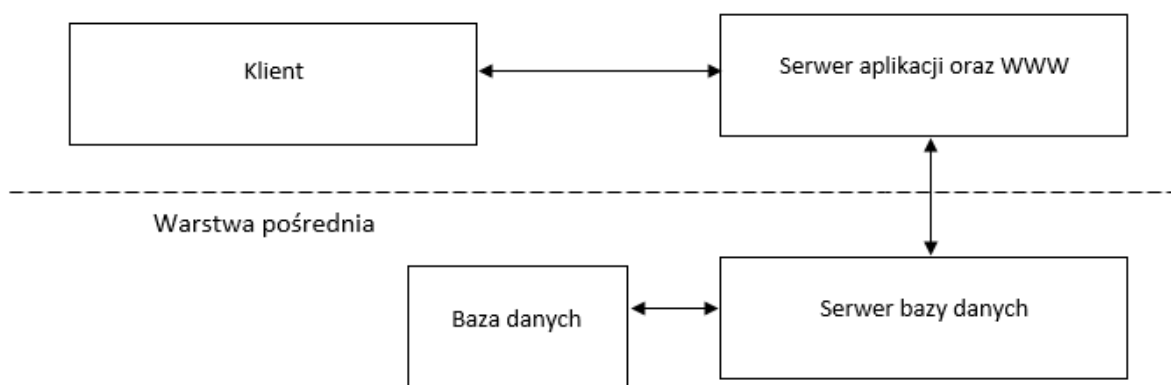


Rysunek 5. Diagram PDM

### 3.4. Mechanizmy komunikacji i sterowania przepływem danych

W projekcie zakładamy, że każda tabela posiada indeks na kluczu głównym. Indeksowanie zwiększy wydajność bazy danych, umożliwi szybsze wyszukiwanie rekordów oraz zwiększy szybkość dostępu do danych. Zamierzamy też wykorzystać wyzwalacze. Będą one automatycznie zwiększały ID tabeli podczas dodawania nowego wiersza. Do komunikacji z bazą danych wykorzystujemy protokół komunikacyjny TCP/IP, a służy nam do tego MySQL Connector.

#### 3.4.1. Schemat komunikacji, struktura systemu



Rysunek 6. Schemat komunikacji

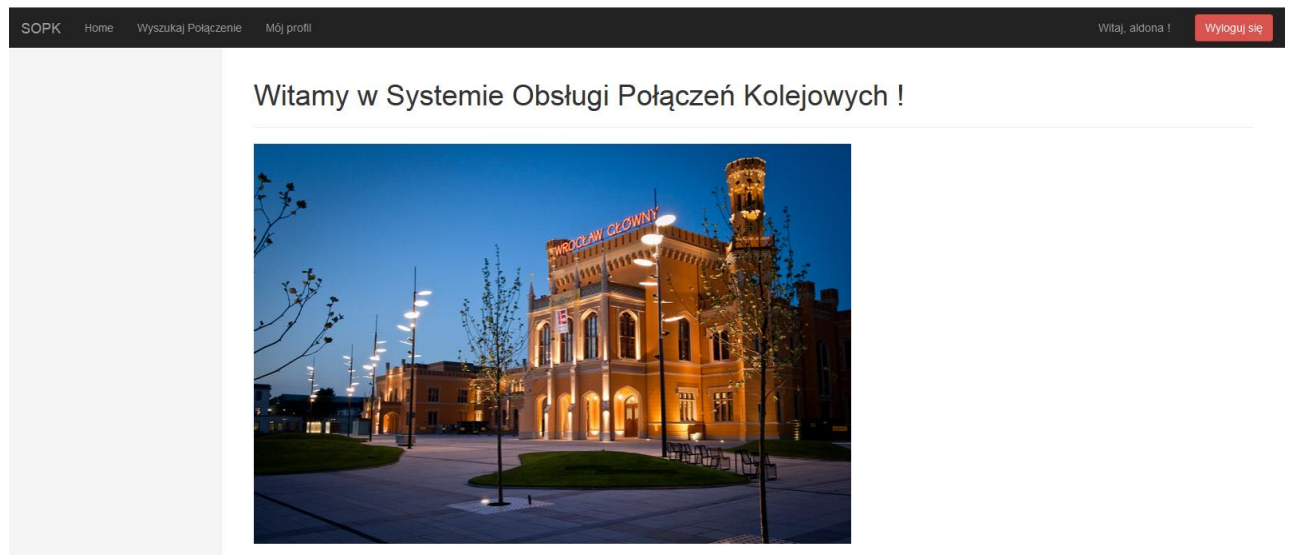
### 3.5. Mechanizmy przetwarzania danych

Mechanizmy przetwarzania danych udostępniły nam funkcje API oraz mapowanie relacyjno-obiektowe frameworku Django. Tuż po stworzeniu modeli (odpowiednik tabeli SQL) framework Django udostępnia nam „wirtualną obiektową bazę danych”, w której możemy tworzyć, edytować oraz usuwać obiekty używając języka Python. Odpowiada to poleceniom SQL – INSERT, ALTER oraz DELETE.

## 4. Projekt aplikacji

### 4.1. Projekt graficzny interfejsu

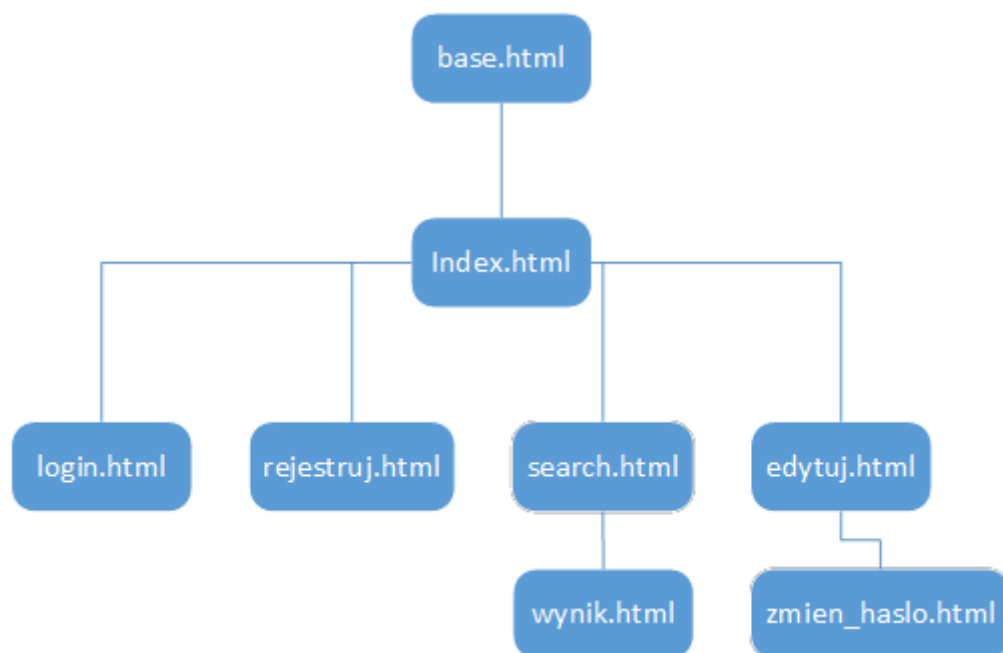
Na rysunku x prezentujemy graficzny projekt aplikacji. Jak widać na górze strony znajduje się pasek nawigacyjny (Navigation Bar). W chwili obecnej zagospodarowana jest tylko centralna część strony głównej, panele boczne pozostają puste.



Rysunek 7. Strona główna Systemu Obsługi Połączeń Kolejowych

#### 4.2. Struktura logiczna menu

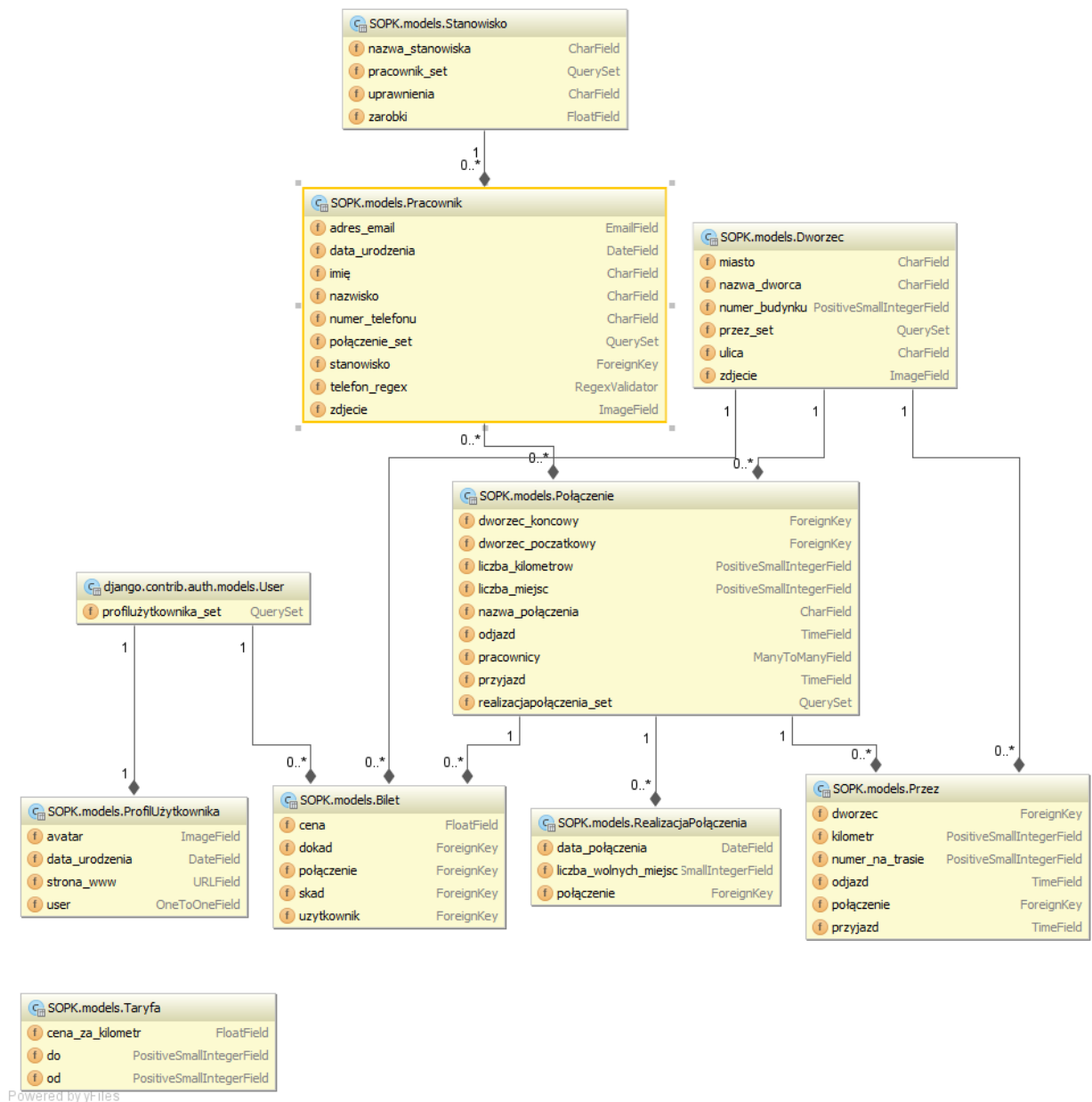
W zaimplementowanym przez nas projekcie do każdej opcji można się dostać nie więcej niż trzema kliknięciami myszy (często trzeba, także wypełnić odpowiedni formularz). Dostęp do poszczególnych szablonów uzyskujemy poprzez przejście kolejnych poziomów drzewa przedstawionego na rysunku 9. Szablony generowane są przez widoki znajdujące się w pliku views.py same szablony znajdują się w katalogu templates w folderze static.



Rysunek 8. Drzewo zagnieżdżenia szablonów

### 4.3. Sposób mapowania bazy danych

Mapowanie bazy danych odbywa się przy pomocy ORM dostarczonego nam przez framework Django. Poniżej przedstawiamy model implementacyjny bazy danych Systemu Obsługi Połączeń Kolejowych.



Rysunek 9. Model implementacyjny bazy danych Systemu Obsługi Połączeń Kolejowych

#### **4.4. Bezpieczeństwo aplikacji**

Współcześnie luki w oprogramowaniu oraz błędy w zabezpieczeniach wydają się codziennością. Wirusy rozprzestrzeniają się z niesamowitą prędkością, jednocześnie wykorzystując ogromne ilości zainfekowanych komputerów, jako broń do ataków internetowych. W niekończącym się wyścigu zbrojeń programiści próbują walczyć z hakerami oraz spamerami tworząc, co raz bardziej nowoczesne zabezpieczenia. Każdego dnia dochodzi do nas wiele doniesień o kradzieży tożsamości z zaatakowanych stron internetowych. Właśnie, dlatego w projekcie „Systemu Obsługi Połączeń Kolejowych” kluczową rolę stanowi bezpieczeństwo. Tutaj z pomocą przychodzi nam framework webowy Django, który wykorzystaliśmy do napisania naszej aplikacji. Został on stworzony, aby automatycznie chronić programistę przed wieloma najbardziej powszechnymi błędami dotyczącymi ochrony oprogramowania, które popełniają początkujący, jak również doświadczeni programiści. Django dostarcza wiele ciekawych narzędzi, które umożliwiają programiście między innymi:

- Kontrolę dostępu do wybranych funkcjonalności aplikacji w zależności od statusu użytkownika;
- Ochronę przed atakami CSRF;
- Ochronę przed atakami typu Clickjacking;
- Ochronę przed SQL Injection;
- Szyfrowanie i ochronę haseł użytkownika;

Przedstawione działania zapewniające bezpieczeństwo to tylko niektóre spośród szerokiego wachlarza zabezpieczeń, które oferuje nam Django. Wszystkie wyżej wymienione zastosowaliśmy w naszej aplikacji.

##### **4.4.1. Kontrola dostępu do wybranych funkcjonalności aplikacji**

W aplikacji System Obsługi Połączeń Kolejowych zabezpieczamy niektóre funkcjonalności przed dostępem nieautoryzowanych użytkowników np. edycję profilu, funkcje administratora. Jednocześnie w zależności od statusu użytkownika udostępniamy pewne możliwości np. dostęp do funkcjonalności administratora mają tylko użytkownicy, którzy posiadają uprawnienia administratora. Do tego celu wykorzystujemy dekoratory dostępne w bibliotece standardowej django oraz wstawki pythonowe w szablonach stron. Django oferuje kilka



dekoratorów, które mogą być stosowane w odniesieniu do obsługi różnych funkcji http. Dekorator powoduje to, że

*Listing 1. Fragment pliku views.py - przykład zastosowania dekoratorów*

```
@csrf_protect
@login_required
def password_change(request):

    zmieniono_haslo = False
    error_message = collections.OrderedDict()
    context = {}
    form = PasswordChangeForm(request.user)
    context["form"] = form
```

widok akceptuje tylko określony typ metody request. Listing 1 przedstawia fragment pliku views.py, gdzie definiowaliśmy widoki. Pokazuje przykład użycia dekoratorów w odniesieniu do widoku odpowiedzialnego za zmianę hasła. Dekoratory umieszczamy bezpośrednio nad definicją widoku i poprzedzamy znakiem '@'. Najczęściej używanym przez nas dekoratorem jest '@login\_required'. Powoduje on udostępnienie widoku tylko dla zarejestrowanych użytkowników. Kolejnym dekoratorem przedstawionym na listingu 1 jest „@csrf\_protect”. Powoduje on aktywację automatycznych mechanizmów ochrony przed atakami CSRF. Ochronę przed CSRF omówimy w kolejnym punkcie. W naszym projekcie korzystamy z tych dwóch dekoratorów. Kontrolę dostępu realizujemy, także na poziomie szablonów. Listing 2 przedstawia fragment głównego szablonu base.html, w którym wykorzystaliśmy wstawki pythonowe do kontroli dostępu. W pliku \*.html wstawki pythonowe rozpoczynamy od nawiasu '{' oraz symbolu '%' i kończymy w ten sam sposób w odwrotnej kolejności. W zależności od statusu użytkownika udostępniamy odpowiednie pola menu. Jeżeli użytkownik jest autoryzowany udostępniamy mu menu mój profil. Jeżeli dodatkowo ma status superuser, czyli

*Listing 2. Fragment szablonu base.html - kontrola dostępu*

```
<div class="navbar-collapse collapse">
    <ul class="nav navbar-nav navbar-left">
        <li><a href="{% url 'SOPK:index' %}">Home</a></li>
        <li><a href="{% url 'SOPK:szukaj' %}">Wyszukaj Połączenie</a></li>

        {% if user.is_authenticated %}
```

```
<li><a href="{% url 'SOPK:restricted' %}">Mój profil</a></li>
{% if user.is_superuser %}
    <li><a href="admin/">Admin</a></li>
{% endif %}
{% endif %}
</ul>
```

jest administratorem udostępniamy mu link admin, który przenosi go do strony administratora. Menu „Wyszukaj połączenie” jest dostępne dla wszystkich i nie wymaga specjalnego warunku.

#### 4.4.2. Ochrona przed atakami CSRF

Atak Cross site request forgery umożliwia złośliwemu użytkownikowi wykonywać działania przy użyciu poświadczeń innego użytkownika bez wiedzy i zgody tego użytkownika. Ataki CSRF nie są wymierzone w strony internetowe i nie muszą powodować zmiany ich treści. Celem hakera jest wykorzystanie uprawnień ofiary do wykonania operacji. Django posiada wbudowaną ochronę przed większością rodzajów CSRF jednak należy odpowiednio z niej korzystać. W naszej aplikacji dodaliśmy 'django.middleware.csrf.CsrfViewMiddleware' w sekcji MIDDLEWARE\_CLASSES w pliku settings.py. Zostało to pokazane na listingu 3. Co więcej w szablonach gdzie znajdują się formularze umieszczamy wstawki pythonowe {% csrf\_token %}, które generują jednorazowy kody podczas każdej metody post. Używamy, także dekoratora „CSRS protect”

*Listing 3. Fragment pliku settings.py*

```
MIDDLEWARE_CLASSES = (
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.auth.middleware.SessionAuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
)
```

Ochrona CSRF działa poprzez zbadanie kodu jednorazowego w każdym żądaniu POST. Gwarantuje to, że złośliwy użytkownik nie może po prostu powtórzyć metody POST na stronie

internetowej, a jednocześnie inny zalogowany użytkownik nieświadomie potwierdzi przesłany formularz. Szkodliwy użytkownik musiałby znać jednorazowy kod, który jest specyficzny dla danego użytkownika (pliki cookie).

#### 4.4.3. Ochrona przed atakami typu Clickjacking

Clickjacking, czyli porywanie kliknięć to nowe zagrożenie w Internecie. Polega na wykorzystywaniu często używanych przycisków popularnych serwisów internetowych. Zasada działania: przez luki w zabezpieczeniach stron internetowych hakerzy uzyskują dostęp do kodu programu danej strony i w sposób niezauważalny dla użytkowników do ważnych przycisków menu przypisują własne funkcje. Od tego momentu kliknięcia użytkowników przestają prowadzić do oczekiwanej strony. Zamiast tego użytkownik ląduje na stronie zaprogramowanej przez hakerów - kliknięcie zostało porwane. W naszej aplikacji dodaliśmy 'django.middleware.clickjacking.XFrameOptionsMiddleware' w sekcji MIDDLEWARE\_CLASSES w pliku settings.py. To zabezpieczenie w przeglądarce internetowej uniemożliwia renderowanie wewnątrz ramki i jednocześnie zapobiega atakom clickjacking.

#### 4.4.4. Ochrona przed SQL Injection

SQL Injection to rodzaj ataku, gdzie szkodliwy użytkownik jest w stanie wykonać dowolny kod SQL w bazie danych. Może to spowodować, że rekordy tabeli zostaną usunięte lub do wycieku danych. Stosując API Django do komunikacji z bazą danych zapytania SQL będą w stanie być wolne od SQL Injection. W naszym projekcie nie wykonujemy „surowych” zapytań SQL, a korzystamy z wbudowanego API. Rozważmy następujące surowe zapytanie:

```
SELECT * FROM user_contacts WHERE username = " OR 'a' = 'a';
```

Ponieważ w takim zapytaniu mamy do czynienia z niezabezpieczonym SQL dodane przez atakującego klauzula or sprawia, że każdy wiersz jest zwracany. W przypadku skorzystania z API Django mielibyśmy do czynienia z następującą sytuacją:

```
foo.get_list(bar__exact="" OR 1=1")
```

Django uniknie stworzenia niebezpiecznego zapytania, API przetworzy to na:

```
SELECT * FROM foos WHERE bar = '' OR 1=1'
```

Dostaliśmy bezpieczne zapytanie. Właśnie z tego powodu w projekcie korzystamy wyłącznie z API Django. Dodatkowo zabezpieczamy się przed SQL Injection korzystając z formularzy i validatorów.

*Listing 4. Fragment pliku forms.py - validatorzy i formularze*

```
def validate_dlugosc(password):
    if len(password) < 8:
        raise ValidationError("Hasło powinno mieć co najmniej 8 znaków")

def validate_liczba(password):
    jestLiczba = any(char.isdigit() for char in password)
    if not jestLiczba:
        raise ValidationError("Hasło powinno mieć co najmniej 1 cyfrę")

def validate_spacja(password):
    jestSpacja = any(char == " " for char in password)
    if jestSpacja:
        raise ValidationError("Hasło nie może zawierać spacji")

class UzytkownikFormularz(forms.ModelForm):

    username = forms.CharField(help_text="Login:")
    password = forms.CharField(widget=forms.PasswordInput(), help_text="Hasło:",
        validators=[validate_dlugosc,
                                                             validate_liczba,
                                                             validate_spacja])

    email = forms.EmailField(help_text="E-Mail:")
    first_name = forms.CharField(help_text="Imię:")
    last_name = forms.CharField(help_text="Nazwisko:")
```

Na listingu 4 przedstawiliśmy formularz wykorzystywany przez nas podczas rejestracji użytkownika. Już samo definiowanie formularzy umieszczanych w pliku forms.py jest wolne od SQL Injection, ponieważ korzystamy z API Django. Jednak framework udostępnia nam narzędzie validatorów. Służą one do wymuszenia na użytkowniku konkretnej zawartości wpisywanej do formularza. Definiuje się je, jako funkcje sprawdzające zawartość pola. W przypadku, gdy zawartość jest niezgodna ze zwracanym warunkiem funkcja validatora zwraca Validation Error. Na listingu 4 znajdują się trzy validatorzy. Pierwszy sprawdza czy hasło zawiera, co najmniej 8 znaków, drugi czy w hasle znajdują się spacje, trzeci czy w hasle znajdują się spacje. Validatorzy przekazujemy, jako parametr pola klasy formularza, jako listę funkcji. W przypadku, gdy validator zwróci błąd użytkownik jest o tym informowany stosownym

komunikatem. Stosowanie formularzy wraz z validatorami dodatkowo zabezpiecza naszą aplikację przed SQL Injection.

#### 4.4.5. Szyfrowanie i ochrona haseł użytkownika w bazie danych

W naszej aplikacji korzystamy z elastycznego systemu przechowywania haseł dostarczanego przez Django. Hasła szyfrujemy za pomocą PBKDF2 z hashem sha256. Hasło w bazie danych przechowywane są, jako stringi w formacie: <algorytm>\$<iteracja>\$<numer>\$<hash>. Są to składniki wykorzystywane do przechowywania hasła Użytkownika, rozdzielonych znakiem dolara. Hasło składa się z algorytmu mieszającego, liczby iteracji algorytmu, losowego numeru oraz hashu. Do sprawdzania hasła używamy API zawartego w bibliotece django.auth.contrib. Hasła użytkowników są niewidoczne dla administratora przeglądającego tabelę.

## 5. Implementacja bazy danych

Bazę danych zaimplementowaliśmy wykorzystując framework Django i jego API do komunikacji z bazą danych. Poszczególne tabele deklarowaliśmy, jako klasy w pliku models.py widocznym na listingu 5. Do komunikacji z bazą danych wykorzystaliśmy narzędzie udostępnione przez mysql: mysql.connector.

*Listing 5. Zawartość pliku models.py*

```
# -*- coding: utf-8 -*-
from django.db import models
from django.core.validators import RegexValidator
from django.contrib.auth.models import User

class Dworzec (models.Model):

    def __str__(self):
        return '{0}: {1}'.format(self.nazwa_dworca, self.miasto)

    nazwa_dworca = models.CharField(max_length=40, verbose_name="Nazwa Dworca")
    miasto = models.CharField(max_length=40, verbose_name="Miasto")
    ulica = models.CharField(max_length=50, verbose_name="Ulica")
    numer_budynku = models.PositiveSmallIntegerField(verbose_name="Numer Budynku")
    zdjecie = models.ImageField("Zdjęcie Dworca", upload_to= "media/dworce/", blank = True, null = True,)
    class Meta:
        db_table = "Dworce"
        verbose_name_plural = "Dworce"

class Stanowisko (models.Model):

    def __str__(self):
        return self.nazwa_stanowiska
```

```

nazwa_stanowiska = models.CharField(primary_key=True, unique=True, max_length=100,
                                     verbose_name="Nazwa Stanowiska")
zarobki = models.FloatField(verbose_name="Zarobki [zł]")
uprawnienia = models.CharField(max_length=100, verbose_name="Uprawnienia")

class Meta:
    db_table = "Stanowiska"
    verbose_name_plural = "Stanowiska"

class Pracownik (models.Model):

    def __str__(self):
        return '{0}: {1} {2}'.format(self.stanowisko, self.imię, self.nazwisko)

    imię = models.CharField(max_length= 20, verbose_name="Imię")
    nazwisko = models.CharField(max_length=40, verbose_name="Nazwisko")
    data_urodzenia = models.DateField(verbose_name="Data Urodzenia")
    telefon_regex = RegexValidator(regex=r'^\+?1?\d{9,15}$',
                                   message="Phone number must be entered in the format: '+9999999999'. Up to 15
digits allowed.")
    numer_telefonu = models.CharField(validators=[telefon_regex], max_length=15, blank=True,
                                     verbose_name="Numer Telefonu")
    adres_email = models.EmailField(blank=True, null= True, verbose_name="Adres E-Mail")
    stanowisko = models.ForeignKey(Stanowisko, verbose_name="Stanowisko")
    zdjecie = models.ImageField("Zdjęcie Legitymacyjne", upload_to= "media/pracownicy/", blank =
True, null = True)

    class Meta:
        db_table = "Pracownicy"
        verbose_name_plural = "Pracownicy"

class Połączenie (models.Model):

    def __str__(self):
        return '{0} {1} Relacja: {2}-{3}'.format(self.id, self.nazwa_połączenia, self.dworzec_początkowy,
        self.dworzec_koncowy)

    nazwa_połączenia = models.CharField(max_length=30, verbose_name="Nazwa Połączenia")
    dworzec_początkowy = models.ForeignKey(Dworzec, related_name='dworzec_początkowy',
    verbose_name="Dworzec Początkowy")
    dworzec_koncowy = models.ForeignKey(Dworzec, related_name='dworzec_koncowy',
    verbose_name="Dworzec Końcowy")
    odjazd = models.TimeField(verbose_name="Odjazd")
    przyjazd = models.TimeField(verbose_name="Przyjazd")
    liczba_kilometrow = models.PositiveSmallIntegerField(verbose_name="Liczba Kilometrów")
    liczba_miejsc = models.PositiveSmallIntegerField(verbose_name="Liczba Miejsc")
    pracownicy = models.ManyToManyField(Pracownik)

    class Meta:
        db_table = "Połączenia"
        verbose_name_plural = "Połączenia"

class Przez (models.Model):

    def __str__(self):
        return '{0} {1} Odjazd: {2}'.format(self.numer_na_trasie, self.dworzec, self.odjazd)

```

```

połączenie = models.ForeignKey(Połączenie, verbose_name="Połączenie")
numer_na_trasie = models.PositiveSmallIntegerField(verbose_name="Numer Na Trasie")
dworzec = models.ForeignKey(Dworzec, verbose_name="Dworzec")
przyjazd = models.TimeField(verbose_name="Przyjazd")
odjazd = models.TimeField(verbose_name="Odjazd")
kilometr = models.PositiveSmallIntegerField(verbose_name="Kilometr")

class Meta:
    db_table = "Przez"
    verbose_name_plural = "Przez"

class Bilet (models.Model):

    def __str__(self):
        return '{0} {1} {2} zł'.format(self.id, self.połączenie, self.cena)

    połączenie = models.ForeignKey(Połączenie)
    uzytkownik = models.ForeignKey(User)
    skad = models.ForeignKey(Dworzec, related_name='skad', verbose_name="Skąd")
    dokad = models.ForeignKey(Dworzec, related_name='dokad', verbose_name="Dokąd")
    cena = models.FloatField(verbose_name="Cena [zł]")

    class Meta:
        db_table = "Bilety"
        verbose_name_plural = "Bilety"

class Taryfa(models.Model):

    def __str__(self):
        return 'Od: {0}km do {1}km cena\km {3}zł: {2}'.format(self.od, self.do, self.cena_za_kilometr)

    od = models.PositiveSmallIntegerField(verbose_name= "Od")
    do = models.PositiveSmallIntegerField(verbose_name= "Do")
    cena_za_kilometr = models.FloatField(verbose_name="Cena Za Kilometr [zł]")

    class Meta:
        db_table = "Taryfy"
        verbose_name_plural = "Taryfy"

class ProfilUzytkownika(models.Model):

    user = models.OneToOneField(User)
    data_urodzenia = models.DateField(verbose_name= "Data Urodzenia")
    strona_www = models.URLField(blank=True, null = True, verbose_name= "Strona WWW")
    avatar = models.ImageField(upload_to='media/zdjecia_profilowe', blank=True, null=True,
    verbose_name= "Avatar")

    class Meta:
        db_table = "ProfileUzytkowników"
        verbose_name_plural = "Profile Użytkowników"

    def __str__(self):
        return self.user.username

class RealizacjaPołączenia(models.Model):

```

```

def __str__(self):
    return 'Odjazd: {0}'.format(self.data_połączenia)

data_połączenia = models.DateField(null=False, blank=False, verbose_name="Data Połączenia")
połączenie = models.ForeignKey(Połączenie)
liczba_wolnych_miejsc = models.PositiveSmallIntegerField()

class Meta:
    db_table = "RealizacjaPołączenia"
    verbose_name_plural = "Realizacje Połączeń"

```

### 5.1. Generowanie kodu SQL

Każda klasa zawiera definicję pól oraz klasę meta zawierającą metadane (nazwę tabeli, odmianę nazwy tabeli w liczbie mnogiej). Bazę zaimplementowaliśmy korzystając z poleceń python manage.py makemigrations oraz migrate. W tym celu API pythona zamieniło kod zawarty w pliku models.py na następujące polecenia sql przedstawione na listingu 6.

*Listing 6. Polecenia użyte do implementacji bazy danych*

```

BEGIN;
CREATE TABLE `Bilety` (`id` integer AUTO_INCREMENT NOT NULL PRIMARY KEY, `cena` double precision NOT NULL);
CREATE TABLE `Dworce` (`id` integer AUTO_INCREMENT NOT NULL PRIMARY KEY, `nazwa_dworca` varchar(40) NOT NULL, `miasto` varchar(40) NOT NULL, `ulica` varchar(50) NOT NULL, `numer_budynku` smallint UNSIGNED NOT NULL, `zdjecie` varchar(100) NULL);
CREATE TABLE `Połączenia` (`id` integer AUTO_INCREMENT NOT NULL PRIMARY KEY, `nazwa_połączenia` varchar(30) NOT NULL, `odjazd` time(6) NOT NULL, `przyjazd` time(6) NOT NULL, `liczba_kilometrow` smallint UNSIGNED NOT NULL, `liczba_miejsc` smallint UNSIGNED NOT NULL, `dworzec_koncowy_id` integer NOT NULL, `dworzec_początkowy_id` integer NOT NULL);
CREATE TABLE `Pracownicy` (`id` integer AUTO_INCREMENT NOT NULL PRIMARY KEY, `imię` varchar(20) NOT NULL, `nazwisko` varchar(40) NOT NULL, `data_urodzenia` date NOT NULL, `numer_telefonu` varchar(15) NOT NULL, `adres_email` varchar(75) NULL, `zdjecie` varchar(100) NULL);
CREATE TABLE `ProfileUżytkowników` (`id` integer AUTO_INCREMENT NOT NULL PRIMARY KEY, `data_urodzenia` date NOT NULL, `strona_www` varchar(200) NULL, `avatar` varchar(100) NULL, `user_id` integer NOT NULL UNIQUE);
CREATE TABLE `Przez` (`id` integer AUTO_INCREMENT NOT NULL PRIMARY KEY, `numer_na_trasie` smallint UNSIGNED NOT NULL, `przyjazd` time(6) NOT NULL, `odjazd` time(6) NOT NULL, `kilometr` smallint UNSIGNED NOT NULL, `dworzec_id` integer NOT NULL, `połączenie_id` integer NOT NULL);
CREATE TABLE `RealizacjaPołączenia` (`id` integer AUTO_INCREMENT NOT NULL PRIMARY KEY, `data_połączenia` date NOT NULL, `liczba_wolnych_miejsc` smallint UNSIGNED NOT NULL, `połączenie_id` integer NOT NULL);
CREATE TABLE `Stanowiska` (`nazwa_stanowiska` varchar(100) NOT NULL PRIMARY KEY, `zarobki` double precision NOT NULL, `uprawnienia` varchar(100) NOT NULL);
CREATE TABLE `Taryfy` (`id` integer AUTO_INCREMENT NOT NULL PRIMARY KEY, `od` smallint UNSIGNED NOT NULL, `do` smallint UNSIGNED NOT NULL, `cena_za_kilometr` double precision NOT NULL);
ALTER TABLE `Pracownicy` ADD COLUMN `stanowisko_id` varchar(100) NOT NULL;

```



```

ALTER TABLE `Pracownicy` ALTER COLUMN `stanowisko_id` DROP DEFAULT;
CREATE TABLE `Połączenia_pracownicy` (`id` integer AUTO_INCREMENT NOT NULL
PRIMARY KEY, `połączenie_id` integer NOT NULL, `pracownik_id` integer NOT NULL,
UNIQUE (`połączenie_id`, `pracownik_id`));
ALTER TABLE `Bilety` ADD COLUMN `dokad_id` integer NOT NULL;
ALTER TABLE `Bilety` ALTER COLUMN `dokad_id` DROP DEFAULT;
ALTER TABLE `Bilety` ADD COLUMN `połączenie_id` integer NOT NULL;
ALTER TABLE `Bilety` ALTER COLUMN `połączenie_id` DROP DEFAULT;
ALTER TABLE `Bilety` ADD COLUMN `skad_id` integer NOT NULL;
ALTER TABLE `Bilety` ALTER COLUMN `skad_id` DROP DEFAULT;
ALTER TABLE `Bilety` ADD COLUMN `uzytkownik_id` integer NOT NULL;
ALTER TABLE `Bilety` ALTER COLUMN `uzytkownik_id` DROP DEFAULT;
CREATE INDEX Połączenia_555d8584 ON `Połączenia` (`dworzec_koncowy_id`);
ALTER TABLE `Połączenia` ADD CONSTRAINT
`Połączenia_dworzec_koncowy_id_626b10b8_fk_Dworce_id` FOREIGN KEY
(`dworzec_koncowy_id`) REFERENCES `Dworce` (`id`);
CREATE INDEX Połączenia_c3a5c785 ON `Połączenia` (`dworzec_poczkotkowy_id`);
ALTER TABLE `Połączenia` ADD CONSTRAINT
`Połączenia_dworzec_poczkotkowy_id_eb02f11_fk_Dworce_id` FOREIGN KEY
(`dworzec_poczkotkowy_id`) REFERENCES `Dworce` (`id`);
ALTER TABLE `ProfileUzytkownikow` ADD CONSTRAINT
`ProfileUzytkownikow_user_id_1b286f36_fk_auth_user_id` FOREIGN KEY (`user_id`)
REFERENCES `auth_user` (`id`);
CREATE INDEX Przez_4bdb464d ON `Przez` (`dworzec_id`);
ALTER TABLE `Przez` ADD CONSTRAINT `Przez_dworzec_id_70747c68_fk_Dworce_id`
FOREIGN KEY (`dworzec_id`) REFERENCES `Dworce` (`id`);
CREATE INDEX Przez_1af73be4 ON `Przez` (`połączenie_id`);
ALTER TABLE `Przez` ADD CONSTRAINT `Przez_połączenie_id_362169a3_fk_Połączenia_id`
FOREIGN KEY (`połączenie_id`) REFERENCES `Połączenia` (`id`);
CREATE INDEX RealizacjaPołączenia_1af73be4 ON `RealizacjaPołączenia` (`połączenie_id`);
ALTER TABLE `RealizacjaPołączenia` ADD CONSTRAINT
`RealizacjaPołączenia_połączenie_id_df2e282_fk_Połączenia_id` FOREIGN KEY
(`połączenie_id`) REFERENCES `Połączenia` (`id`);
CREATE INDEX Pracownicy_6457299b ON `Pracownicy` (`stanowisko_id`);
ALTER TABLE `Pracownicy` ADD CONSTRAINT
`Pracownicy_stanowisko_id_3b2e59c6_fk_Stalowiska_nazwa_stalowiska` FOREIGN KEY
(`stanowisko_id`) REFERENCES `Stalowiska` (`nazwa_stalowiska`);
CREATE INDEX Połączenia_pracownicy_1af73be4 ON `Połączenia_pracownicy`
(`połączenie_id`);
ALTER TABLE `Połączenia_pracownicy` ADD CONSTRAINT
`Połączenia_pracownicy_połączenie_id_4ef67cec_fk_Połączenia_id` FOREIGN KEY
(`połączenie_id`) REFERENCES `Połączenia` (`id`);
CREATE INDEX Połączenia_pracownicy_b91259e4 ON `Połączenia_pracownicy`
(`pracownik_id`);
ALTER TABLE `Połączenia_pracownicy` ADD CONSTRAINT
`Połączenia_pracownicy_pracownik_id_3163c1c4_fk_Pracownicy_id` FOREIGN KEY
(`pracownik_id`) REFERENCES `Pracownicy` (`id`);
CREATE INDEX Bilety_b9e270a7 ON `Bilety` (`dokad_id`);
ALTER TABLE `Bilety` ADD CONSTRAINT `Bilety_dokad_id_1ffe1331_fk_Dworce_id`
FOREIGN KEY (`dokad_id`) REFERENCES `Dworce` (`id`);
CREATE INDEX Bilety_1af73be4 ON `Bilety` (`połączenie_id`);
ALTER TABLE `Bilety` ADD CONSTRAINT `Bilety_połączenie_id_288df576_fk_Połączenia_id`
FOREIGN KEY (`połączenie_id`) REFERENCES `Połączenia` (`id`);

```

```
CREATE INDEX Bilety_02b6620c ON `Bilety` (`skad_id`);
ALTER TABLE `Bilety` ADD CONSTRAINT `Bilety_skad_id_29ccf048_fk_Dworce_id` FOREIGN
KEY (`skad_id`) REFERENCES `Dworce` (`id`);
CREATE INDEX Bilety_71f8319f ON `Bilety` (`uzytkownik_id`);
ALTER TABLE `Bilety` ADD CONSTRAINT `Bilety_uzytkownik_id_6f5ff7e0_fk_auth_user_id`
FOREIGN KEY (`uzytkownik_id`) REFERENCES `auth_user` (`id`);
```

```
COMMIT;
```

## 5.2. Realizacja ograniczeń integralnościowych

Wszystkie typy wykorzystywanych przez nas pól są szczegółowo opisane na stronie <https://docs.djangoproject.com/en/dev/ref/models/fields/>. Poczyniliśmy pewne ograniczenia, co do rozmiaru niektórych pól. W wielu przypadkach korzystamy z `PostivielIntegerSmallField` 0 to 32767, ponieważ większe wartości nie są nam potrzebne. W polach oznaczonych, jako `CharField` maksymalną długość stringa ustalaliśmy na poziomie modelu, przekazując, jako parametr pola `max_length` z odpowiednio wpisaną długością. Pole `ImageField` nie przechowuje bezpośrednio zdjęcia, tylko ścieżkę do zdjęcia, które samo w sobie znajduje się w folderze media. Pola, które są kluczami głównymi tabeli, a nie zostały zadeklarowane jawnie w pliku `models.py` tworzone są automatycznie, jako id tabeli. Kluczem obcym jest id tabeli, będącej parametrem pola `ForeignKey`. Każde pole ma przekazane, jako parametr `verbose name`, które jest wykorzystywane w implementacji szablonów aplikacji. Tabele zostały zaimplementowane zgodnie z przyjętym projektem.

## 6. Implementacja aplikacji

### 6.1. Przepływ danych w Django

Przepływ danych rozpoczyna się od przeglądarki internetowej. Następuje żądanie poprzez wpisanie odpowiedniego adresu na pasku przeglądarki. Żądanie wędruje do serwera Apache, następnie do pliku `wsgi.py`, a następnie przekazywane jest pliku `urls.py`. Plik `urls.py` naszego projektu został przedstawiony na listingu 7. W pliku znajduje się zbiór funkcji url. Funkcja `url` jako pierwszy argument przyjmuje wyrażenie regularne. Jeżeli żądanie zostanie dopasowane do wyrażenia regularnego żądanie przekazywane jest do widoku będącego drugim argumentem funkcji.

*Listing 7. Plik `urls.py`*

```
from django.conf.urls import patterns, url
from django.contrib import admin
from SOPK import views
```

```
admin.autodiscover()
urlpatterns = patterns('',
    url(r'^$', views.index, name='index'),
    url(r'^rejestruj/$', views.rejestruj, name='rejestruj'),
    url(r'^logowanie/$', views.logowanie, name='logowanie'),
    url(r'^edycja_profilu/$', views.edycja, name='restricted'),
    url(r'^zmiana_hasla/$', views.password_change, name='password'),
    url(r'^szukaj/$', views.szukaj, name='szukaj'),
    url(r'^wyloguj/$', views.wylogowywanie, name='wylogowywanie'),
)
```

Widoki zdefiniowane są w pliku views.py. Kiedy przychodzi odpowiednie żądanie w zależności od jego rodzaju mogą pobierać oraz przetwarzać elementy z bazy danych. Wszystkie dane z widoku przekazywane są do szablonu, który jest w końcowym etapie renderowany użytkownikowi podczas przeglądania strony.

## **6.2. Implementacja wybranych funkcjonalności**

Podczas implementacji funkcjonalności aplikacji do stworzenia graficznego interfejsu użytkownika wykorzystaliśmy bibliotekę twitter bootstrap dostępną na stronie <http://getbootstrap.com/>

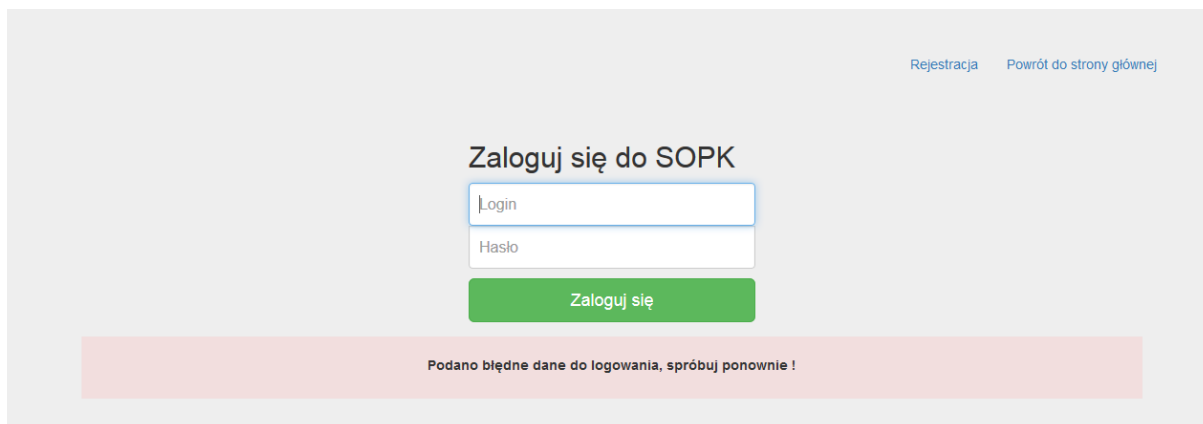
Zostały zaimplementowane następujące funkcjonalności:

- logowanie;
- wylogowanie;
- edycja profilu użytkownika;
- zmiana hasła;
- wyszukiwanie połączeń
- rejestracja nowego użytkownika;
- funkcjonalność administratora;

### **6.2.1. Logowanie**

Funkcjonalność logowania napisaliśmy z wykorzystaniem API Django. Logowanie dostępne jest zarówno ze strony głównej jak i strony logowania. Żądanie logowania przetwarza widok logowanie. Do widoku metodą POST zostają przekazane login i hasło. Metoda authenticate autoryzuje użytkownika sprawdzając w bazie danych poprawność danych. Następnie, jeżeli autoryzacja przebiegła z błędem renderowana jest strona z błędem przedstawiona na rysunku

10. Natomiast w przypadku poprawnej autoryzacji wykonywana jest metoda login, która tworzy nową sesję użytkownika. Następuje przekierowanie do strony głównej.



*Rysunek 10. Strona logowania do Systemu Obsługi Połączeń Kolejowych*

### **6.2.2. Wylogowanie**

Wylogowanie korzysta z widoku wylogowanie. Jeżeli użytkownik wciśnie przycisk wyloguj request z przeglądarki idzie właśnie do tego widoku. Widok zawiera jedną metodę logout przyjmującą request. W przypadku wylogowania zamykana jest sesja użytkownika i następuje przekierowanie do strony głównej.

### **6.2.3. Edycja profilu użytkownika**

Edycja profilu korzysta z widoku edycja. Do edycji profilu widok korzysta z formularzy AktualizujProfilFormularz oraz ProfilUżytkownikaFormularz. Jeżeli nastąpi żądanie z metodą post. Sprawdzana jest poprawność formularzy metodami is\_valid(). Jeżeli formularze są wypełnione poprawnie wykonywana jest na nich metoda save() i oba zapisywane są do bazy danych w przeciwnym wypadku do kontekstu szablonu wysyłana jest lista błędów.

Rysunek 11. Strona edycji profilu użytkownika

#### 6.2.4. Zmiana hasła

Zmiana hasła wykorzystuje widok `password_change`. Widok korzysta z formularza `PasswordChangeForm`. Jeżeli formularz jest wypełniony poprawnie wykonywana jest na nim metoda `save()`, a dane zostają zapisane w bazie danych. Następnie wykonuje się metoda `update_session_auth_hash`. Aktualizowany jest hash sesji użytkownika, dzięki czemu sesja jest utrzymana. W przeciwnym wypadku do kontekstu przekazywana jest lista błędów.

Rysunek 12. Strona zmiany hasła

#### 6.2.5. Wyszukiwanie połączeń

Wyszukiwanie połączeń widok `szukaj`. Gdy następuje żądanie ze strony wyszukiwania przedstawionej na rysunku 13 za pomocą funkcji `filter` pobierane są dane z bazy danych: `RealizacjaPołączenia.objects.filter(data_połączenia = poszukiwana_data)`. Następnie są one sortowane wg. godziny odjazdu. Następnie pętlą `for` przechodzimy po wszystkich połączeniach i wybieramy te spełniające warunki wyszukiwania i dodajemy je do listy. Następnie lista przekazywana jest do kontekstu i renderowany jest szablon `wyniki.html` przedstawiony na rysunku 14. Jeżeli połączenia nie zostały znalezione do kontekstu przekazujemy listę błędów.

Wyszukaj Połączenie
Powrót na stronę główną

Szukaj Połączenia

Skąd:

Dokąd:

Data odjazdu:

Szukaj

Rysunek 13. Strona wyszukiwania połączeń

Powrót do wyszukiwania
Powrót do strony głównej

Wyniki wyszukiwania:  
Gdynia - Wrocław Odjazd: 2014-12-30 10:50

✓ Przesuwaj wiersze, aby zmienić kolejność

	Nazwa Połączenia	Skąd	Dokąd	Data Odjazdu	Odjazd	Data Przyjazdu	Przyjazd	Czas trwania	Liczba Kilometrów
1	Husarz	Dworzec Główny: Gdynia	Dworzec Główny: Wrocław	30 grudnia 2014	10:55:01	30 grudnia 2014	17:55:06	7:00:05	600
2	Balon	Dworzec Główny: Gdynia	Dworzec Główny: Wrocław	30 grudnia 2014	17:54:21	31 grudnia 2014	06:54:22	13:00:01	600

Rysunek 14. Strona wyników wyszukiwania

## 6.2.6. Rejestracja nowego użytkownika

Rejestracja użytkownika korzysta z widoku rejestruj. Widok ten wykorzystuje formularze UżytkownikFormularz oraz ProfilUżytkownikaFormularz. Jeżeli przychodzi żądanie post weryfikacja formularzy i rejestracja następują analogicznie jak w przypadku edycji profilu użytkownika.

Rejestracja
Powrót na stronę główną

Login:

Hasło:

E-Mail:

Imię:

Nazwisko:

Data Urodzenia:

Strona WWW:

Avatar:  
 Nie wybrano pliku.

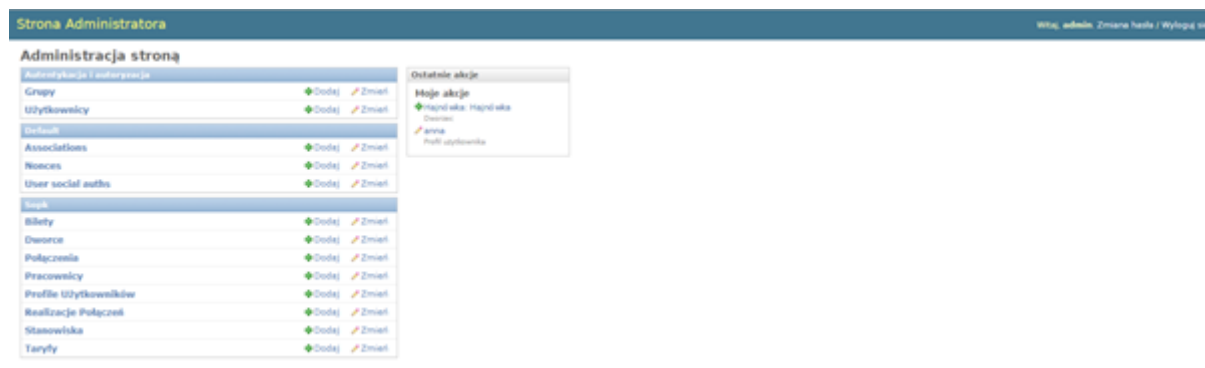
Zarejestruj

Rysunek 15. Formularz rejestracji nowego użytkownika

## 6.2.7. Funkcjonalność administratora

Kod panelu administratora znajduje się w pliku admin.py. Zostały tam opisane klasy będące reprezentacją każdej tabeli. W tych klasach określiliśmy metody służące do filtrowania

tabeli oraz sposobu wyświetlania danych. Umieszczaliśmy je w panelu administratora korzystając z funkcji `admin.site.register`.



Rysunek 16. Panel administratora

## 7. Konfigurowanie i testowanie systemu

Aplikacja z poziomu użytkownika nie wymaga instalacji. Jedynym wymaganiem jest posiadanie przeglądarki internetowej z obsługą języka javascript. Jako preferowaną przeglądarkę sugerujemy Mozillę Firefox w wersji 34 lub wyższej. Aplikacja będzie działać na przeglądarkach Google Chrome oraz Opera. Nie zalecamy stosowania przeglądarki Internet Explorer. Po uruchomieniu przeglądarki należy wpisać adres aplikacji. W kolejnym punkcie przedstawimy konfigurację środowiska produkcyjnego. Jeżeli środowisko zostanie skonfigurowane zgodnie z naszymi instrukcjami strona powinna się uruchomić po wpisaniu „localhost” w pasku przeglądarki.

### 7.1. Konfiguracja środowiska produkcyjnego

Opisywana konfigurację środowiska produkcyjnego dotyczy systemu MS Windows w wersji 7 lub wyższej. Konfigurację można, także przeprowadzić w systemie Linux, jednak w tym przypadku konfiguracja zależy od rodzaju dystrybucji.

W pierwszym kroku należy skopiować katalog z naszą aplikacją do dowolnego katalogu na dysku. W kolejnym kroku należy pobrać ze strony <https://www.python.org/downloads/> Pythona w wersji 3 (preferowana wersja to 3.4.2). Następnie należy dodać katalog Pythona do ścieżki w zmiennej środowiskowej. W celu upewnienia się należy uruchomić cmd i wpisać polecenie „python”. Powinien się uruchomić interpreter pythona. Następnie w konsoli cmd należy użyć narzędzia ‘pip’ do pobrania pakietów wymaganych do działania aplikacji. ‘Pip’ jest managerem pakietów pythona. Poleceniem `pip –install nazwa_pakietu` należy zainstalować wymagane pakiety. W miejsce `nazwa_pakietu` trzeba kolejno wpisywać: `django`, `Pillow`, `python social-auth`.

W kolejnych krokach należy przejść do konfiguracji bazy danych mysql i serwera bazy danych. Trzeba pobrać ze strony <http://dev.mysql.com/downloads/windows/installer/> najnowszą wersję pakietu mysql i zainstalować pełną wersję korzystając z instalatora. Wszystkie ustawienia pozostawić, jako domyślne. Potem należy dodać folder, gdzie została zainstalowana baza mysql do ścieżki w zmiennej środowiskowej. Na zakończenie uruchomić cmd i wpisać polecenie `mysql -u root -p` i dwukrotnie kliknąć enter (jeżeli podano hasło użytkownika podczas instalacji wpisać hasło). Następnie przejść do konfiguracji bazy danych. Kolejno wpisywać polecenia:

```
CREATE DATABASE SOPK CHARACTER SET utf8;
CREATE USER 'admin'@'localhost' IDENTIFIED BY 'admin';
GRANT ALL ON django.* TO 'admin'@'localhost';
```

Po wpisaniu powyższych poleceń wyjść z konfiguracji mysql i za pomocą cmd wejść do głównego katalogu projektu. Należy wpisać następujące polecenia:

```
python manage.py makemigrations;
python manage.py migrate;
python manage.py collectstatic;
python manage.py runserver;
```

Pierwsza komenda przygotowuje migracje z pliku `models.py` natomiast druga komenda wykonuje te migracje. Ostatnia komenda uruchamia prosty lightwebowy serwer. Umożliwi to sprawdzenie czy do tego momentu konfiguracja została wykonana poprawnie

Ostatnim krokiem jest konfiguracja serwera Apache 2.2. Serwer możemy pobrać ze strony <http://www.programosy.pl/program,apache-http-server.html>. Instalujemy go za pomocą instalatora. Po instalacji w prawym dolnym rogu ekranu powinna pojawić się ikona serwera umożliwiająca wykonywanie poleceń `start`, `restart`, `stop`. Później należy pobrać i zainstalować `mod_wsgi` ze strony <http://code.google.com/p/modwsgi/wiki/DownloadTheSoftware?tm=2> `mod_wsgi` dla pythona w wersji 3.4.2. Moduł ten jest odpowiedzialny za komunikację serwera apache z aplikacją django. Następnie musimy odnaleźć folder instalacji serwera Apache, znaleźć katalog `conf` i otworzyć dowolnym edytorem tekstu plik `httpd.conf`. W tym pliku trzeba dodać fragmenty kodu przedstawione na poniższym listingu.



Listing 8. Konfiguracja pliku httpd.conf

```
LoadModule wsgi_module modules/mod_wsgi.so

.....

Alias /static/ {sciezka}/BazyDanych2/static/

<Directory {sciezka}//BazyDanych2/static>
    Order deny,allow
    Allow from all
</Directory>

WSGIScriptAlias / {sciezka}//BazyDanych2/BazyDanych2/wsgi.py
WSGIProxyPath {sciezka}//BazyDanych2/

<Directory {sciezka}//BazyDanych2/BazyDanych2>
    <Files wsgi.py>
        Order deny,allow
        Allow from all
    </Files>
</Directory>
```

Na zakończenie należy wykonać restart serwera. Po wpisaniu localhost w przeglądarce uruchomi się nasz program.

## 7.2. Konfiguracja środowiska testowego

Konfiguracja środowiska testowego jest bardzo prosta. Django dostarcza nam narzędzia do wykonywania testów automatycznych. Musimy dograć za pomocą narzędzia pip moduł selenium. Wykorzystanie narzędzia pip przedstawiliśmy w poprzednim punkcie. Testy piszemy w pliku tests.py. Testy uruchamiamy w katalogu głównym aplikacji korzystając z cmd za pomocą komendy:

```
python manage.py test
```

Na potrzeby testów tworzona jest oddzielna baza danych, dlatego nie ma niebezpieczeństwa uszkodzenia danych produkcyjnych.

## 7.3. Testy z wykorzystaniem narzędzia Selenium

Pierwsze narzędzie testowy, z którego korzystaliśmy nazywa się Selenium. Wykorzystaliśmy wersję Selenium, jako moduł do pythona. Selenium wykonuje testy w

przeglądarce Internetowej symulując scenariusze wykonywane przez użytkownika. Narzędzie porównuje zawartość strony z zawartością oczekiwaną przez testera. Kryterium przejścia testu to zgodność zawartości strony z zawartością oczekiwaną. Jeżeli zawartość się nie zgadza test nie przechodzi. Listing 9 przedstawia test formularza rejestracji w naszej aplikacji.

*Listing 9. Fragment pliku tests.py - przykład zastosowania narzędzia Selenium*

```
from django.test import TestCase, LiveServerTestCase
from selenium.webdriver.firefox.webdriver import WebDriver
from selenium.webdriver.support.wait import WebDriverWait
from django.core.urlresolvers import reverse
from django.contrib.auth.models import User

class SeleniumTests(LiveServerTestCase):

    @classmethod
    def setUpClass(cls):
        cls.selenium = WebDriver()
        super(SeleniumTests, cls).setUpClass()

    @classmethod
    def tearDownClass(cls):
        cls.selenium.quit()
        super(SeleniumTests, cls).tearDownClass()

    def test_register(self):
        timeout = 2

        self.selenium.get('%s%s' % (self.live_server_url, '/rejestruj/'))
        username_input = self.selenium.find_element_by_name('username')
        username_input.send_keys("zbyszek")
        password_input = self.selenium.find_element_by_name('password')
        password_input.send_keys("secret")
        email_input = self.selenium.find_element_by_name('email')
        email_input.send_keys("zbigniew.stolarz@wp.pl")
        first_name_input = self.selenium.find_element_by_name('first_name')
        first_name_input.send_keys("Zbigniew")
        last_name_input = self.selenium.find_element_by_name('last_name')
        last_name_input.send_keys("Stolarz")
        data_urodzenia_input =
self.selenium.find_element_by_name('data_urodzenia')
        data_urodzenia_input.send_keys("12.12.1991")
        self.selenium.find_element_by_xpath('//*[@value="Zarejestruj"]').click()
        WebDriverWait(self.selenium, 10).until(lambda driver :
driver.find_element_by_tag_name('body'))
```

```
label_input = self.selenium.find_element_by_partial_link_text('Powrót do
strony głównej.')
```

Testy Selenium umieściliśmy w klasie SeleniumTests dziedziczącej po LiveServerTestCase. W metodach setUpClass i tearDownClass znajdują się warunki rozpoczęcia i zakończenia testów. Każdy test zaczyna się od słowa kluczowego test. Do testów wykorzystujemy przeglądarkę internetową Mozilla Firefox. Test rejestracji rozpoczyna się uruchomieniem przeglądarki. Następnie wpisywany jest adres formularza rejestracji. Potem za pomocą selenium.find\_element\_by\_name wyszukujemy odpowiednie pola, żeby potem funkcją send\_keys przypisać im wartości. Metodą self.selenium.find\_element\_by\_xpath('//input[@value="Zarejestruj"]') wyszukujemy przycisk zarejestruj i funkcją click() symulujemy kliknięcie. Następnie czekamy dopóki nie załaduje się nowa strona. Jeżeli na nowej stronie znaleźliśmy link powrotu do strony głównej oznacza to że test zakończył się pomyślnie.

#### 7.4. Testy jednostkowe z wykorzystaniem biblioteki djangounittest

Kolejnym narzędziem, które wykorzystaliśmy do testowania jest biblioteka django unittest. Pozwoliła nam ona napisać testy jednostkowe do istniejącego kodu. Testy jednostkowe w odróżnieniu do testów Selenium nie wykorzystują przeglądarki. Symulacja scenariuszów testowych odbywa się na poziomie kodu. W testach znajdują się funkcje typu assert przerywające test w przypadku, gdy warunek nie jest spełniony. Kryterium powodzenia testu jest spełnienie wszystkich warunków testu. Listing 3 przedstawia dwa spośród wielu testów zawartych w pliku test.py w naszym projekcie. Testy umieszczone są w klasie dziedziczącej po TestCase. Każdy test rozpoczyna się od słowa kluczowego test.

*Listing 10. Fragment pliku tests.py - przykład zastosowania biblioteki django unittest*

```
from django.test import TestCase, LiveServerTestCase
from selenium.webdriver.firefox.webdriver import WebDriver
from selenium.webdriver.support.wait import WebDriverWait
from django.core.urlresolvers import reverse
from django.contrib.auth.models import User

class ViewsTests(TestCase):

    def test_index_positive_response(self):
        response = self.client.get(reverse('SOPK:index'))
```

```
self.assertEqual(response.status_code, 200)
self.assertTemplateUsed(response, 'SOPK/index.html')

def test_positive_login(self):

    User.objects._create_user(username='marek', email='marek@wp.pl',
password='secret',is_staff=False,
                                is_superuser=True)
    response = self.client.post(reverse('SOPK:logowanie'), {'username': 'marek',
                                                                'password': 'secret'})
    self.assertEqual(response.status_code, 302)
```

Pierwszy test bada odpowiedź serwera na żądanie od przeglądarki o wyświetlenie strony głównej. Test przechodzi, jeżeli kod odpowiedzi jest równy 200, a szablon użyty do wyświetlenia strony to index.html. Drugi test sprawdza działanie logowania. Najpierw na potrzeby testu tworzony jest nowy użytkownik. Potem następuje logowanie. Jeżeli kod odpowiedzi wyniósł 302 logowanie odbyło się poprawnie.

## 7.5. Wydajność systemu

Do pomiaru wydajności systemu wykorzystaliśmy program Apache JMeter. Jest to zaawansowane narzędzie do testowania obciążenia dla analizy i pomiaru wydajności różnorodnych usług, z naciskiem na aplikacje internetowe. Badania wykonaliśmy dla następującej konfiguracji sprzętowej:

- System Operacyjny: MS Windows 8.1 Pro (64-bit);
- Processor : Intel Core i5-3210M CPU @ 2.50 GHz;
- Pamięć operacyjna: 12 GB RAM;
- Dysk Twardy: 750 GB;
- Karta Graficzna: NVidia GeForce GT 630 M.

Niestety nie dysponujemy profesjonalnym sprzętem przestawionym w wymaganiach niefunkcjonalnych, jako urządzenie do udostępniania aplikacji w wersji produkcyjnej. W programie Apache JMeter, aby zmierzyć obciążenie serwera dla określonej liczby użytkowników oraz średni czas odpowiedzi serwera zmienialiśmy liczbę wątków, które odpowiadały liczbie użytkowników. Wątki były uruchamiane w ciągu 60s. Badanie wykonaliśmy dla 10, 100, 500, 1000 oraz 2000 użytkowników. Program JMeter zwrócił nam wyniki w postaci tabeli przedstawionej na Rysunku 17.

Sample #	Start Time	Thread Name	Etykieta	Sample Time(ms)	Status	Bytes	Latency
1	12-29-45.522	Grupa wątków 1-23	HTTP Request	130	Success	4217	121
2	12-29-45.523	Grupa wątków 1-7	HTTP Request	149	Success	4217	144
3	12-29-45.523	Grupa wątków 1-13	HTTP Request	171	Success	4217	160
4	12-29-45.522	Grupa wątków 1-14	HTTP Request	173	Success	4217	169
5	12-29-45.448	Grupa wątków 1-17	HTTP Request	251	Success	4217	243
6	12-29-45.522	Grupa wątków 1-9	HTTP Request	177	Success	4217	172
7	12-29-45.450	Grupa wątków 1-18	HTTP Request	257	Success	4217	249
8	12-29-45.453	Grupa wątków 1-11	HTTP Request	263	Success	4217	255
9	12-29-45.522	Grupa wątków 1-22	HTTP Request	198	Success	4217	193
10	12-29-45.522	Grupa wątków 1-12	HTTP Request	213	Success	4217	205
11	12-29-45.451	Grupa wątków 1-4	HTTP Request	292	Success	4217	284
12	12-29-45.445	Grupa wątków 1-1	HTTP Request	299	Success	4217	295
13	12-29-45.579	Grupa wątków 1-28	HTTP Request	166	Success	4217	165
14	12-29-45.523	Grupa wątków 1-24	HTTP Request	223	Success	4217	223
15	12-29-45.523	Grupa wątków 1-21	HTTP Request	224	Success	4217	213
16	12-29-45.523	Grupa wątków 1-19	HTTP Request	229	Success	4217	226
17	12-29-45.629	Grupa wątków 1-30	HTTP Request	136	Success	4217	136
18	12-29-45.599	Grupa wątków 1-29	HTTP Request	184	Success	4217	177
19	12-29-45.690	Grupa wątków 1-32	HTTP Request	94	Success	4217	93
20	12-29-45.720	Grupa wątków 1-33	HTTP Request	77	Success	4217	73
21	12-29-45.659	Grupa wątków 1-31	HTTP Request	148	Success	4217	138
22	12-29-45.751	Grupa wątków 1-34	HTTP Request	56	Success	4217	56
23	12-29-45.780	Grupa wątków 1-35	HTTP Request	33	Success	4217	33
24	12-29-45.814	Grupa wątków 1-36	HTTP Request	13	Success	4217	12
25	12-29-45.841	Grupa wątków 1-37	HTTP Request	12	Success	4217	11
26	12-29-45.872	Grupa wątków 1-38	HTTP Request	11	Success	4217	11
27	12-29-45.903	Grupa wątków 1-39	HTTP Request	13	Success	4217	13
28	12-29-45.932	Grupa wątków 1-40	HTTP Request	12	Success	4217	11
29	12-29-45.963	Grupa wątków 1-41	HTTP Request	12	Success	4217	12
30	12-29-45.993	Grupa wątków 1-42	HTTP Request	11	Success	4217	11
31	12-29-46.024	Grupa wątków 1-43	HTTP Request	11	Success	4217	11
32	12-29-46.053	Grupa wątków 1-44	HTTP Request	22	Success	4217	22
33	12-29-45.523	Grupa wątków 1-3	HTTP Request	570	Success	4217	567
34	12-29-45.522	Grupa wątków 1-25	HTTP Request	601	Success	4217	599

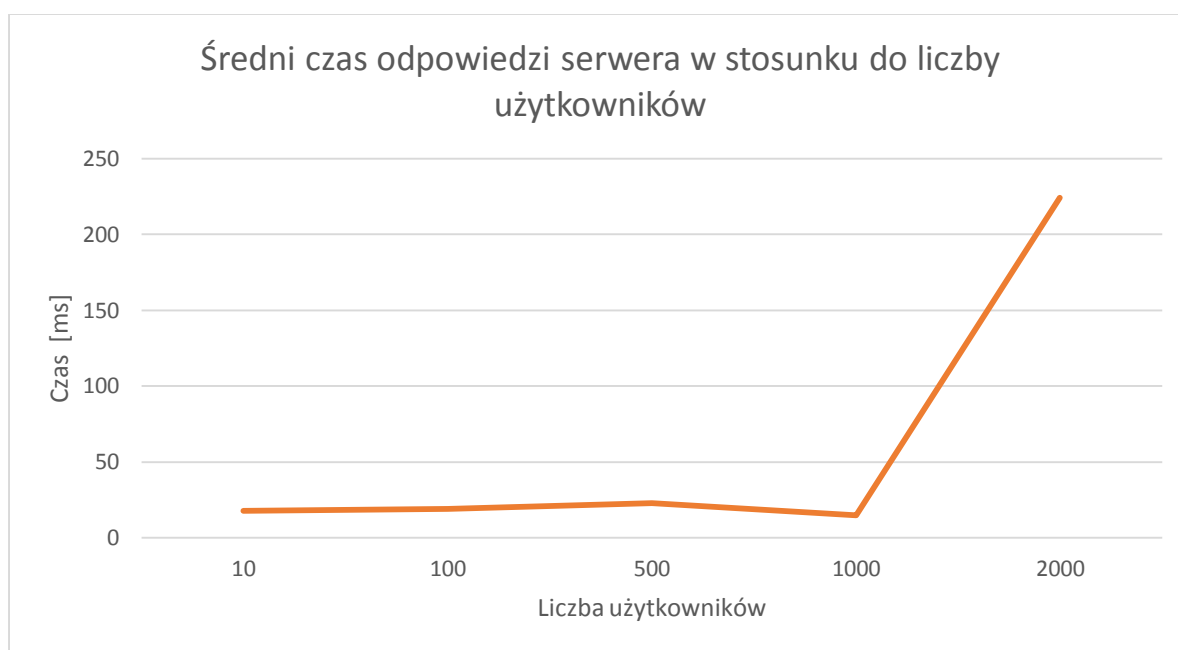
☐ Scroll automatically? ☐ Child samples? No of Samples 2000 Latest Sample 1102 Average 224 Deviation 452

Rysunek 17. Wyniki dla 2000 użytkowników w programie Apache JMeter

Wyniki eksperymentu przedstawiają się następująco:

Tabela 1. Wyniki pomiaru wydajności serwera w stosunku do obciążenia

Liczba użytkowników	Średni czas odpowiedzi serwera [ms]
10	18
100	19
500	23
1000	15
2000	224



Wykres 1. Średni czas odpowiedzi serwera w stosunku do liczby użytkowników

Wyniki dla 10, 100, 500 i 1000 użytkowników są porównywalne. Wydajność serwera podczas skoku z 10 do 1000 zmienia się nieznacznie. W przypadku 1000 użytkowników jest o 3 ms niższa niż w przypadku 10 użytkowników. Znaczny spadek wydajności widzimy przy zwiększeniu obciążenia z 1000 na 2000 użytkowników. Obserwujemy tu prawie 15-krotny wzrost średniego czasu odpowiedzi serwera. W założeniach funkcjonalnych przyjęliśmy zakładany czas odpowiedzi serwera poniżej 200 ms. W przypadku 2000 użytkowników czas ten jest nieznacznie gorszy. Jednak tutaj przyczyną rozbieżności pomiędzy wynikami pomiaru, a założeniami mogą być błąd pomiarowy oraz różnica środowiska, w którym przeprowadzano badania od zakładanego środowiska produkcyjnego.

## 8. Podsumowanie

Na podstawie przyjętych założeń powstał kompletny projekt bazy danych oraz aplikacji. W fazie projektowania bazy danych poznaliśmy od podstaw etapy powstawania projektu bazy

danych poczynając od diagramu związków encji, przez model logiczny, aż do powstania modelu fizycznego bazy danych. W czasie realizacji tej fazy projektu napotkaliśmy liczne problemy związane z późniejszym działaniem całej aplikacji. Aby wszystko działało sprawnie należało odpowiednio zamodelować bazę danych uwzględniając użyte później mechanizmy. W fazie projektowania aplikacji poznaliśmy zasady działania frameworku Django i języka Python służące do wdrożenia projektu w środowisku produkcyjnym. Przyjęta technologia implementacji – Test Driven Development umożliwiła nam zapoznanie się z metodami testowania aplikacji.

Rozwiązania zastosowane w projekcie pozwalają go w stosunkowo prosty sposób rozwijać. Korzystamy z najnowszych, obecnie dostępnych, technologii, które zyskują coraz większą popularność.

Mapowanie obiektowo-relacyjne jest to mechanizm znacznie ułatwiającym pracę z bazą danych. Dzięki niemu możemy komunikować się z bazą danych używając języka obiektowego zamiast wysyłania zapytań SQL. Wystarczy jedynie podstawowa znajomość języka SQL żeby móc utworzyć i używać bazy danych.

Framework Django umożliwia nam połączenie systemu generowania szablonów stron WWW z komunikacją z bazą danych. Dostarczone funkcje API pozwalają w prosty sposób operować danymi pobranymi z systemu zarządzania bazą danych MySQL.

Do utworzenia aplikacji potrzebna jest znajomość języka Python na poziomie średnio-zaawansowanym.

Najtrudniejszą rzeczą w całym projekcie jest integracja systemu Django z serwerem Apache 2 z powodu trudności w komunikacji pomiędzy plikiem `httpd.conf`, a `wsgi.py` oraz przetwarzaniem plików statycznych przez serwer.

System zarządzania bazą danych MySQL w pełni zaspokoił nasze potrzeby, jednak wybór tego systemu nie jest rzeczą kluczową w projekcie, ponieważ podczas korzystania z mapowania obiektowo-relacyjnego różnice wynikające z przyjętego systemu są niewielkie. Większość dostępnych systemów spisała by się równie dobrze.

## Literatura

- [1] Forcier J., Bissex P., Chun W., Python i Django. Programowanie aplikacji webowych, Helion, Gliwice, 2009.
- [2] Holovaty A., Kaplan-Moss, J., The Definitive Guide to Django: Web Development Done Right, Apress, Chicago, 2009.
- [3] Alchin M., Pro Django, Apress, Chicago, 2013.
- [4] DuBois P., MySQL. Vademecum profesjonisty, Helion, Gliwice, 2014.
- [5] Bloom R., Apache Server 2.0: The Complete Reference, Osborne Media, 2002.
- [6] Górski J., Inżynieria oprogramowania w projekcie informatycznym, Mikom, Warszawa, 2000.
- [7] strona: <https://docs.djangoproject.com/en/1.7/>, 13.01.2015
- [8] strona: <http://www.djangobook.com/en/2.0/index.html>, 13.01.2015
- [9] strona: <https://docs.python.org/3/>, 13.01.2015
- [10] strona: <http://httpd.apache.org/docs/2.2/>, 13.01.2015
- [11] strona: <http://dev.mysql.com/doc/refman/5.6/en/>, 13.01.2015
- [12] strona: <http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/>, 13.01.2015
- [13] strona: <http://getbootstrap.com/>, 13.01.2015
- [14] strona: <http://bootsnipp.com/>, 13.01.2015
- [15] strona: <http://dev.w3.org/html5/html-author/>, 13.01.2015