

Dział Ewidencji Ludności

AUTORZY:

Marcin Sitarz

Łukasz Wdowiak

WSPARCIE MERYTORYCZNE:

dr inż. Roman Ptak

Spis treści

1. Wstęp.....	4
1.1. Cel projektu	4
1.2. Zakres projektu.....	4
2. Analiza wymagań.....	4
2.1. Opis działania i schemat logiczny systemu.....	4
2.2. Wymagania funkcjonalne	4
2.3. Wymagania niefunkcjonalne	5
2.3.1. Wykorzystywane technologie i narzędzia	5
2.3.2. Wymagania dotyczące rozmiaru bazy danych	5
2.3.3. Wymagania dotyczące bezpieczeństwa systemu.....	5
2.3.4. Wymagania dotyczące weryfikacji danych.....	5
3. Projekt systemu	6
3.1. Projekt bazy danych	6
3.1.1. Model fizyczny i ograniczenia integralności danych	6
3.1.2. Inne elementy schematu – mechanizmy przetwarzania danych	6
3.1.3. Projekt mechanizmów bezpieczeństwa na poziomie bazy danych.....	8
3.1.4. Dostęp do poszczególnych tabel	8
3.2. Projekt aplikacji użytkownika	9
3.2.1. Architektura aplikacji i diagramy projektowe	9
3.2.2. Interfejs graficzny i struktura menu	9
3.2.3. Metoda podłączania do bazy danych – integracja z bazą danych.....	10
3.2.4. Projekt zabezpieczeń na poziomie aplikacji	11
4. Implementacja systemu baz danych.....	11
4.1. Tworzenie tabel i definiowanie ograniczeń.....	11
4.1.1. Tworzenie tabel	11
4.1.2. Tworzenie ograniczeń tabel	12
4.2 Implementacja mechanizmów przetwarzania danych.....	13
4.2.1. Tworzenie indeksów.....	13
4.2.2. Tworzenie widoków	13
4.2.3. Tworzenie procedur	15
4.2.4. Tworzenie trigerrów.....	17
4.3. Implementacja uprawnień i innych zabezpieczeń.....	19
4.3.1. Utworzenie ról.....	19

4.3.2. Nadanie uprawnień	19
4.4. Testowanie bazy danych na przykładowych danych	20
4.4.2. Testy procedur.....	21
4.4.3. Testy widoków.....	21
4.4.3. Testy wydajności zapytań.....	22
5. Implementacja i testy aplikacji.....	23
5.1. Instalacja i konfigurowanie systemu	23
5.2. Instrukcja użytkowania aplikacji.....	23
5.3. Testowanie opracowanych funkcji systemu.....	28
5.4. Omówienie wybranych rozwiązań programistycznych	30
5.4.1. Implementacja interfejsu dostępu do bazy danych	30
5.4.2. Implementacja wybranych funkcjonalności systemu	31
5.4.3. Implementacja mechanizmów bezpieczeństwa.....	33
6. Podsumowanie i wnioski.....	35
7. Literatura	36

1. Wstęp

Dział Ewidencji Ludności to organ administracji publicznej odpowiedzialny za gromadzenie, aktualizację i przechowywanie danych osobowych obywateli oraz rejestrowanie istotnych wydarzeń, takich jak urodzenia, małżeństwa i zgony. Jego głównym celem jest tworzenie i zarządzanie rejestrem ludności, który służy do określenia identyfikacji i statusu obywateli w gminie.

1.1. Cel projektu

Celem projektu jest zbudowanie bazy danych dla Działu Ewidencji Ludności oraz strony internetowej, aby zautomatyzować pracę urzędnika.

1.2. Zakres projektu

Naszym zadaniem jest przeanalizowaniem wymagań, zaprojektowanie struktury bazy danych, wybranie odpowiednich technologii, stworzenie schematu bazy danych, implementacja jej jak i również aplikacji webowej.

2. Analiza wymagań

2.1. Opis działania i schemat logiczny systemu

Uprawniony pracownik urzędu zatwierdza wnioski o: wpis nowej osoby do rejestru, wypis osoby z rejestru, zmianę danych (imię, nazwisko, płeć, miejsce zameldowania, stan cywilny), nadanie lub zmianę numeru PESEL, oświadczenie osoby jako zmarłej. Każda osoba z rejestru posiada zestaw podstawowych danych umożliwiających identyfikację i odnalezienie tej osoby, np. Imię, nazwisko, data urodzenia, miejsce urodzenia, numer PESEL, itp. Każdej zarejestrowanej osobie przypisywane jest konto, z którego możliwe jest sprawdzenie swoich danych osobowych jak i złożenie wniosków. Wnioski przechodzą przez wstępną walidację danych przez system pod względem poprawności, a następnie są zatwierdzane przez pracownika. Dostęp do danych osobowych użytkowników jest ograniczony tylko do uprawnionego personelu.

2.2. Wymagania funkcjonalne

- Użytkownik zamieszkały na terenie gminy składa wniosek o wpis swoich danych do rejestru mieszkańców
- Użytkownik, który jest prawnym opiekunem nowonarodzonego dziecka składa wniosek o wpis dziecka do rejestru mieszkańców
- Użytkownik ma możliwość złożenia wniosku o wyrobienie numeru PESEL
- Użytkownik ma możliwość złożenia wniosku o zmianę numeru PESEL w przypadku sprostowania daty urodzenia, zmiany płci lub nieprawidłowych danych
- Użytkownik ma możliwość złożenia wniosku o zmianę miejsca zameldowania
- Użytkownik ma możliwość złożenia wniosku o zmianę imię i nazwiska
- Użytkownik ma możliwość złożenia wniosku o zmianę płci

- Użytkownik ma możliwość złożenia wniosku o zmianę stanu cywilnego
- Użytkownik ma dostęp do wglądu do swoich danych osobowych poprzez stronę internetową
- Użytkownik ma dostęp do wglądu do złożonych przez siebie wniosków poprzez stronę internetową
- Użytkownik ma możliwość złożenia wniosku o wypis z rejestru mieszkańców z powodu przeprowadzki poza teren gminy
- Użytkownik ma możliwość poinformowania o śmierci członka rodziny, co skutkuje dodaniem numeru certyfikatu zgonu do zmarłej osoby

2.3. Wymagania нефункционалне

2.3.1. Wykorzystywane technologie i narzędzia

Pracownicy urzędu pracują na komputerach podłączonych do sieci lokalnej, która łączy się z internetem. Swoją pracę wykonują przy pomocy strony internetowej postawionej na serwerze, z którego wystawiony jest port zewnętrzny, przez który mogą połączyć się oni jak i użytkownicy. Pracownicy posiadają konta administratorskie z możliwością zarządzania wnioskami, o ile nie są to wnioski złożone przez nich samych. W bazie danych nie są przechowywane bezpośrednio certyfikaty tylko ich numery identyfikacyjne. Serwis bazodanowy jest napisany za pomocą języka MySQL. Dostęp do niego zapewnia serwis backendowy napisany za pomocą Python FastAPI. Z kolei za wygląd strony internetowej odpowiada JavaScript z wykorzystaniem biblioteki React.

2.3.2. Wymagania dotyczące rozmiaru bazy danych

Zakłada się, że w bazie danych wnioski będą najliczniejszą encją, której to wielkość będzie liczona w setkach tysięcy. Pozostałe będą liczone w dziesiątkach tysięcy. W Dziale Ewidencji Ludności gminy pracuje 10 pracowników. Każdy z nich rozpatruje dziennie około kilkudziesięciu wniosków. Zakłada się, że osób przeglądających stronę w jednym momencie nie będzie większa niż 1000.

2.3.3. Wymagania dotyczące bezpieczeństwa systemu

Użytkownicy mają możliwość logowania się na swoje konta za pomocą swojego loginu i hasła wygenerowanego przez system i dostarczonego na adres mailowy. Ponadto przy składaniu każdego wniosku są zobowiązani do przejścia Captchy. Dostęp do API mają tylko użytkownicy z wygenerowanym JWT.

2.3.4. Wymagania dotyczące weryfikacji danych

Dane podane we wniosku przez użytkownika przechodzą przez wstępną weryfikację systemową polegającą na: sprawdzeniu czy wszystkie pola są uzupełnione, czy każde pole wymagające konkretnej ilości znaków je posiada, czy wprowadzone zostały tylko dozwolone znaki. Następnie dane zatwierdzone przez system weryfikowane są przez pracownika urzędu. Na podstawie tego czy wniosek został poprawnie wypełniony pracownik może ten wniosek odrzucić bądź zatwierdzić.

3. Projekt systemu

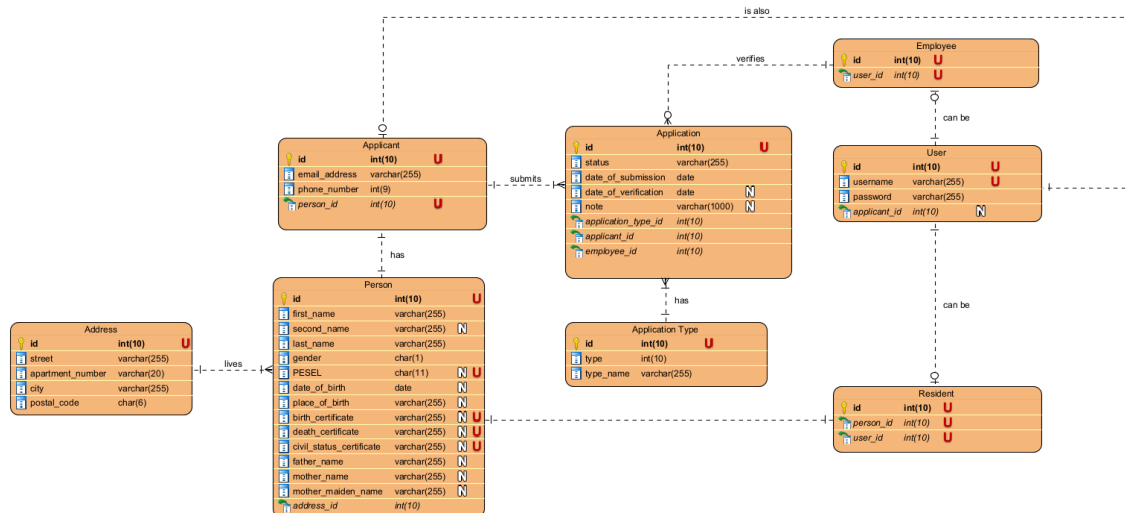
Projekt i struktury bazy danych, mechanizmów zapewniania poprawności przechowywanych informacji, oraz kontroli dostępu do danych.

3.1. Projekt bazy danych

Tworzona przez nas baza danych jest bazą relacyjną, nietemporalną opartą o system zarządzania MySQL.

3.1.1. Model fizyczny i ograniczenia integralności danych

Zdjęcie 1: Diagram związków encji



3.1.2. Inne elementy schematu – mechanizmy przetwarzania danych

1) Indeksy:

- Index 1: Tabela „Person” po „PESEL”
- Index 2: Tabela „Person” po „last_name” + „first_name”
- Index 3: Tabela „Application” po „applicant_id”
- Index 4: Tabela „Application” po „application_type_id”
- Index 5: Tabela „Application” po „date_of_submission” malejąco

2) Przykładowe widoki:

- Widok z informacjami o wnioskach i pracownikach
Nazwa widoku – ApplicationEmployeeView
Pola widoku: application_id, status, date_of_submission, date_of_verification, note, application_type_name, applicant_id, employee_id, employee_first_name, employee_last_name
- Widok z informacjami o użytkownikach i ich wnioskach
Nazwa widoku – UserApplicationView
Pola widoku: user_first_name, user_last_name, user_PESEL, status, date_of_submission, date_of_verification, note, application_type_name

3) Procedury składowe:

- Procedura walidacji numeru PESEL
- Procedura walidacji adresu e-mail
- Procedura walidacji numeru telefonu
- Procedura walidacji kodu pocztowego
- Procedura walidacji danych tekstowych (czy dane tekstowe nie zawierają znaków specjalnych i liczb)
- Procedura dodania nowego pracownika

4) Triggery:

- Trigger automatycznie tworzący użytkownika po złożeniu przez petenta pierwszego wniosku.
Nazwa: assign_employee_before_insert_application
Wyzwolenie: Dodanie nowego wniosku do tabeli Application.
Opis działania: Przy dodawaniu nowego wniosku, trigger przypisuje do niego pracownika z najmniejszą ilością wniosków. Jeżeli to pierwszy wniosek petenta, trigger tworzy dla niego konto użytkownika w tabeli User, generując losowe hasło i używając jego PESEL jako nazwy użytkownika.
- Trigger rejestrujący petenta jako mieszkańca po zaakceptowaniu wniosku.
Nazwa: after_application_update
Wyzwolenie: Aktualizacja wniosku w tabeli Application.
Opis działania: Po zmianie statusu wniosku na "approved" i gdy typ wniosku odpowiada wpisowi do rejestru mieszkańców, trigger dodaje petenta do tabeli Resident, rejestrując go jako mieszkańca.
- Trigger walidujący dane osobowe przed dodaniem do tabeli Person.
Nazwa: before_insert_person
Wyzwolenie: Dodanie nowego rekordu do tabeli Person.
Opis działania: Przed dodaniem osoby, trigger sprawdza poprawność danych takich jak PESEL, imiona, nazwiska, miejsce urodzenia, wykorzystując procedury walidacyjne. Zapobiega to dodaniu błędnych lub niekompletnych danych osobowych.
- Trigger walidujący dane aplikanta przed dodaniem do tabeli Applicant.
Nazwa: before_insert_applicant
Wyzwolenie: Dodanie nowego rekordu do tabeli Applicant.
Opis działania: Ten trigger sprawdza poprawność adresu e-mail i numeru telefonu aplikanta przed jego dodaniem do bazy danych. Użycie procedur walidacyjnych zapewnia, że dane kontaktowe są poprawne i aktualne.
- Trigger walidujący adres przed jego dodaniem do tabeli Address.
Nazwa: before_insert_address
Wyzwolenie: Dodanie nowego rekordu do tabeli Address.
Opis działania: Przed dodaniem nowego adresu, trigger sprawdza poprawność kodu pocztowego, ulicy i miasta. Jest to ważne dla utrzymania prawidłowych i spójnych danych adresowych w systemie.

3.1.3. Projekt mechanizmów bezpieczeństwa na poziomie bazy danych

Dane na samym początku walidowane są przez procedury składowe. Dane znajdujące się w bazie są szyfrowane, aby nawet w przypadku wycieku danych dostęp do nich był utrudniony. Dodatkowo tworzone są dzienniki zdarzeń pozwalające monitorować aktywność w bazie danych - system logowania operacji i śledzenia działań aplikacji. Codziennie wykonywany jest backup, aby w przypadku awarii dysku/serwera przechowywanego zebrane dane można było szybko przywrócić system na "nogi".

3.1.4. Dostęp do poszczególnych tabel

Wyróżniamy takie role jak: administrator, kierownik, pracownik

Administrator ma dostęp do wszystkich czynności we wszystkich tabelach.

1) Tabela „Application”:

- Kierownik i pracownik ma dostęp do przeglądania, dodawania i aktualizacji wniosków.

2) Tabela „Application Type”:

- Kierownik ma dostęp do przeglądania, dodawania, aktualizacji i usuwania typów wniosków.
- Pracownik ma dostęp do przeglądania typów wniosków.

3) Tabela „User”:

- Kierownik i pracownik ma dostęp do przeglądania, dodawania, aktualizacji i usuwania użytkowników.

4) Tabela „Applicant”:

- Kierownik i pracownik ma dostęp do przeglądania, dodawania, aktualizacji i usuwania petentów.

5) Tabela „Resident”:

- Kierownik i pracownik ma dostęp do przeglądania, dodawania, aktualizacji i usuwania mieszkańców.

6) Tabela „Person”:

- Kierownik i pracownik ma dostęp do przeglądania, dodawania, aktualizacji i usuwania danych osobowych.

7) Tabela „Address”:

- Kierownik i pracownik mają dostęp do przeglądania, dodawania, aktualizacji i usuwania adresów.

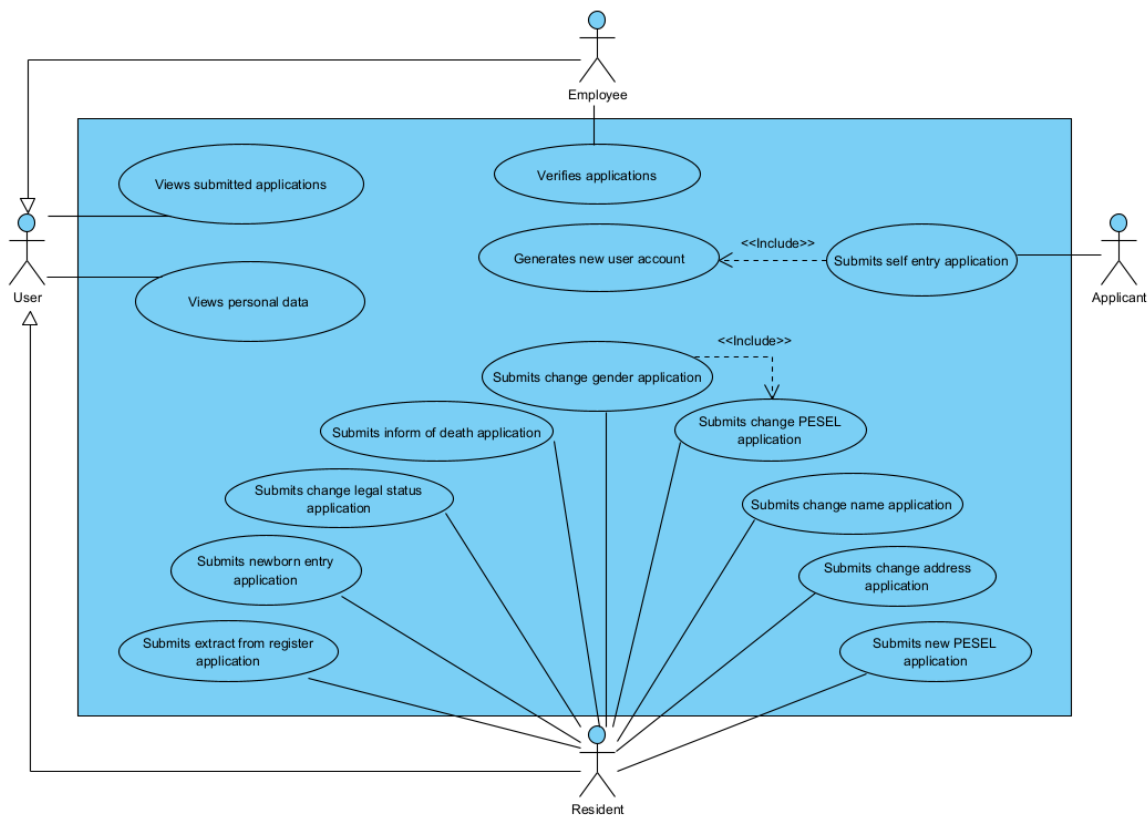
8) Tabela „Employee”:

- Kierownik ma dostęp do przeglądania, dodawania, aktualizacji i usuwania pracowników.

3.2. Projekt aplikacji użytkownika

3.2.1. Architektura aplikacji i diagramy projektowe

Zdjęcie 2: Diagram przypadków użycia



3.2.2. Interfejs graficzny i struktura menu

Graficzny projekt aplikacji został stworzony za pomocą strony internetowej „figma.com”.

Zdjęcie 3: Strona główna



Zdjęcie 4: Formularz składania wniosku o wpis

GMINA ZBĄSZYNEK

zaloguj

Wniosek o wpis do rejestru mieszkańców

Imię

Imię

Nazwisko

Nazwisko

Data urodzenia

DD-MM-YYYY

Email

Email

Miejsce urodzenia

Miejsce urodzenia

Numer Telefonu

+48 XXX XXX XXX

☐ mężczyzna

☐ kobieta

Numer PESEL

PESEL

Numer certyfikatu urodzenia

Numer certyfikatu

Numer certyfikatu stanu cywilnego

Numer certyfikatu

☒ Potwierdzam, że podane przeze mnie dane są prawdziwe.

☒ Zgadzam się na przetwarzanie danych.

Prześlij Wniosek

Zdjęcie 5: Panel Pracownika

GMINA ZBĄSZYNEK

Piotr

PANEL PRACOWNIKA

#	IMIE	NAZWISKO	TYP WNIOSKU	STATUS	DATA ZŁOŻENIA
1	Krzysztof	Kozik	Wpis	Do rozpatrzenia	17-11-2023
2	Tytus	Bomba	Wpis	Rozpatrzony	16-11-2023
3	Christian	Bale	Zmiana imienia	Odrzucony	15-11-2023

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

Podłączenie do bazy danych będzie odbywało się za pomocą biblioteki mysql-connector-library (natywny sterownik), dzięki której po podaniu poprawnych danych logowania do naszej bazy będziemy mogli wprost z naszej aplikacji backendowej wywoływać zapytania SQL. Następnie nasza aplikacja backendowa wystawi API na konkretnym porcie, dzięki czemu nasza aplikacja frontendowa będzie miała dostęp do danych z bazy.

3.2.4. Projekt zabezpieczeń na poziomie aplikacji

Aby obronić się przed takimi atakami jak SQL injection zapytania do backendu są sparametryzowane. Wiąże się to z tym, że użytkownik naszej aplikacji nie ma bezpośredniego dostępu do bazy danych, więc nie ma jak wstrzyknąć SQL`a.

Formularze które wypełniają użytkownicy automatycznie sprawdzają poprawność wpisanych przez nich danych. Przykładowo wyświetli błąd w przypadku podania za dużej liczby znaków w numerze PESEL, bądź numerze telefonu, sprawdzi czy każde pole zostało wypełnione oraz czy wypełnione pole zawiera dozwolone znaki.

Aby obronić się przed atakami, które miałyby na celu wysłanie dużej ilości fałszywych wniosków w momencie próby przesłania takiego wniosku wyświetli się captcha, która należy prawidłowo przejść, aby wniosek został przesłany do dalszej weryfikacji przez pracownika.

Zalogowanie się na konto generuje JWT dzięki któremu użytkownik może przeglądać dalej naszą stronę internetową. Token ten powoduje stworzenie tak zwanej sesji logowania, która wygaśnie po określonym czasie.

4. Implementacja systemu baz danych

4.1. Tworzenie tabel i definiowanie ograniczeń

Tabele oraz ograniczenia zostały wygenerowane przez program Visual Paradigm.

4.1.1. Tworzenie tabel

- Stworzenie tabeli "User":

```
CREATE TABLE `User` (id int(10) NOT NULL AUTO_INCREMENT, username varchar(255) NOT NULL UNIQUE, password varchar(255) NOT NULL, applicant_id int(10), CONSTRAINT id PRIMARY KEY (id));
```

- Stworzenie tabeli "Employee":

```
CREATE TABLE Employee (id int(10) NOT NULL AUTO_INCREMENT, user_id int(10) NOT NULL UNIQUE, CONSTRAINT id PRIMARY KEY (id));
```

- Stworzenie tabeli "Application":

```
CREATE TABLE Application (id int(10) NOT NULL AUTO_INCREMENT, status varchar(255) DEFAULT "in review" NOT NULL, date_of_submission DATETIME DEFAULT CURRENT_TIMESTAMP NOT NULL, date_of_verification DATETIME ON UPDATE CURRENT_TIMESTAMP, note varchar(1000), application_type_id int(10) NOT NULL, applicant_id int(10) NOT NULL, employee_id int(10) NOT NULL, CONSTRAINT id PRIMARY KEY (id));
```

- Stworzenie tabeli "Person":

```
CREATE TABLE Person (id int(10) NOT NULL AUTO_INCREMENT, first_name varchar(255) NOT NULL, second_name varchar(255), last_name varchar(255) NOT NULL, gender char(1) NOT NULL, PESEL char(11) UNIQUE, date_of_birth date, place_of_birth varchar(255), birth_certificate varchar(255) UNIQUE, death_certificate varchar(255) UNIQUE, civil_status_certificate varchar(255) UNIQUE, father_name varchar(255), mother_name varchar(255), mother_maiden_name varchar(255), address_id int(10) NOT NULL, CONSTRAINT id PRIMARY KEY (id));
```

- Stworzenie tabeli "Application Type":

```
CREATE TABLE `Application Type` (id int(10) NOT NULL AUTO_INCREMENT, type int(10) NOT NULL, type_name varchar(255) NOT NULL, CONSTRAINT id PRIMARY KEY (id));
```

- Stworzenie tabeli „Applicant”:

```
CREATE TABLE Applicant (id int(10) NOT NULL AUTO_INCREMENT, email_address varchar(255) NOT NULL, phone_number int(9) NOT NULL, person_id int(10) NOT NULL UNIQUE, CONSTRAINT id PRIMARY KEY (id));
```

- Stworzenie tabeli „Resident”:

```
CREATE TABLE Resident (id int(10) NOT NULL AUTO_INCREMENT, person_id int(10) NOT NULL UNIQUE, user_id int(10) NOT NULL UNIQUE, CONSTRAINT id PRIMARY KEY (id));
```

- Stworzenie tabeli „Address”:

```
CREATE TABLE Address (id int(10) NOT NULL AUTO_INCREMENT, street varchar(255) NOT NULL, apartment_number varchar(20) NOT NULL, city varchar(255) NOT NULL, postal_code char(6) NOT NULL, CONSTRAINT id PRIMARY KEY (id));
```

4.1.2. Tworzenie ograniczeń tabel

- Stworzenie ograniczenia dla tabeli „Application” na bazie tabeli „Application Type”:

```
ALTER TABLE Application ADD CONSTRAINT FKApplication410864 FOREIGN KEY (application_type_id) REFERENCES `Application Type` (id);
```

- Stworzenie ograniczenia dla tabeli „Applicant” na bazie tabeli „Person”:

```
ALTER TABLE Applicant ADD CONSTRAINT FKApplicant692669 FOREIGN KEY (person_id) REFERENCES Person (id);
```

- Stworzenie ograniczenia dla tabeli „Application” na bazie tabeli „Applicant”:

```
ALTER TABLE Application ADD CONSTRAINT FKApplication896957 FOREIGN KEY (applicant_id) REFERENCES Applicant (id);
```

- Stworzenie ograniczenia dla tabeli „Employee” na bazie tabeli „User”:

```
ALTER TABLE Employee ADD CONSTRAINT FKEmployee631292 FOREIGN KEY (user_id) REFERENCES `User` (id);
```

- Stworzenie ograniczenia dla tabeli „Resident” na bazie tabeli „Person”:

```
ALTER TABLE Resident ADD CONSTRAINT FKResident140468 FOREIGN KEY (person_id) REFERENCES Person (id);
```

- Stworzenie ograniczenia dla tabeli „Resident” na bazie tabeli „User”:

```
ALTER TABLE Resident ADD CONSTRAINT FKResident964261 FOREIGN KEY (user_id) REFERENCES `User` (id);
```

- Stworzenie ograniczenia dla tabeli „Application” na bazie tabeli „Employee”:

```
ALTER TABLE Application ADD CONSTRAINT FKApplication361712 FOREIGN KEY (employee_id) REFERENCES Employee (id);
```

- Stworzenie ograniczenia dla tabeli „Person” na bazie tabeli „Address”:

```
ALTER TABLE Person ADD CONSTRAINT FKPerson829511 FOREIGN KEY (address_id) REFERENCES Address (id);
```

- Stworzenie ograniczenia dla tabeli „User” na bazie tabeli „Applicant”:

```
ALTER TABLE `User` ADD CONSTRAINT FKUser595632 FOREIGN KEY (applicant_id) REFERENCES Applicant (id);
```

4.2 Implementacja mechanizmów przetwarzania danych

Mechanizmy zostały napisane w języku SQL bez użycia generatorów kodu.

4.2.1. Tworzenie indeksów

- Index 1: Tabela „Person” po „PESEL”

```
CREATE INDEX idx_pesel ON Person (PESEL);
```

- Index 2: Tabela „Person” po „last_name” + „first_name”

```
CREATE INDEX idx_last_name_first_name ON Person (last_name, first_name);
```

- Index 3: Tabela „Application” po „applicant_id”

```
CREATE INDEX idx_applicant_id ON Application (applicant_id);
```

- Index 4: Tabela „Application” po „application_type_id”

```
CREATE INDEX idx_application_type_id ON Application (application_type_id);
```

- Index 5: Tabela „Application” po „date_of_submission” malejąco

```
CREATE INDEX idx_date_of_submission_desc ON Application (date_of_submission DESC);
```

4.2.2. Tworzenie widoków

- Widok 1: Informacje o wnioskach i pracownikach

```
CREATE VIEW ApplicationEmployeeInfo AS
SELECT
  a.id AS ApplicationID,
  a.status AS ApplicationStatus,
  a.date_of_submission AS SubmissionDate,
  a.date_of_verification AS VerificationDate,
  a.note AS ApplicationNote,
  e.id AS EmployeeID,
  u.username AS EmployeeUsername,
  p.first_name AS EmployeeFirstName,
  p.last_name AS EmployeeLastName
FROM Application a
JOIN Employee e ON a.employee_id = e.id
JOIN `User` u ON e.user_id = u.id
JOIN Person p ON u.applicant_id = p.id;
```

- Widok 2: Informacje o użytkownikach i ich wnioskach

```
CREATE VIEW UserApplicationInfo AS
SELECT
  u.id AS UserID,
  u.username AS Username,
  a.id AS ApplicationID,
  a.status AS ApplicationStatus,
  a.date_of_submission AS SubmissionDate,
  a.date_of_verification AS VerificationDate,
  a.note AS ApplicationNote,
  at.type_name AS ApplicationTypeName
FROM `User` u
JOIN Applicant app ON u.applicant_id = app.id
JOIN Application a ON app.id = a.applicant_id
JOIN `Application Type` at ON a.application_type_id = at.id;
```

- Widok 3: Liczba aplikacji dla każdego typu aplikacji

```
CREATE VIEW ApplicationTypeSummary AS
SELECT
    at.type_name,
    COUNT(a.id) AS NumberOfApplications
FROM `Application Type` at
JOIN Application a ON at.id = a.application_type_id
GROUP BY at.type_name;
```

- Widok 4: Informacje kontaktowe mieszkańców

```
CREATE VIEW ResidentContactInfo AS
SELECT
    r.id AS ResidentID,
    p.first_name,
    p.last_name,
    p.PESEL,
    a.street,
    a.apartment_number,
    a.city,
    a.postal_code,
    app.email_address,
    app.phone_number
FROM Resident r
JOIN Person p ON r.person_id = p.id
JOIN Address a ON p.address_id = a.id
JOIN Applicant app ON p.id = app.person_id;
```

- Widok 5: Role i poziomy dostępu każdego użytkownika

```
CREATE VIEW UserRoleAccess AS
SELECT
    u.id AS UserID,
    u.username,
    CASE
        WHEN e.id IS NOT NULL THEN 'Employee'
        WHEN app.id IS NOT NULL THEN 'Applicant'
        WHEN r.id IS NOT NULL THEN 'Resident'
        ELSE 'Unknown'
    END AS UserRole
FROM `User` u
LEFT JOIN Employee e ON u.id = e.user_id
LEFT JOIN Applicant app ON u.applicant_id = app.id
LEFT JOIN Resident r ON u.id = r.user_id;
```

- Widok 6: Pracownicy i ich przypisania do aplikacji

```
CREATE VIEW EmployeeTaskAssignment AS
SELECT
    e.id AS EmployeeID,
    p.first_name AS EmployeeFirstName,
    p.last_name AS EmployeeLastName,
    a.id AS AssignedApplicationID,
    a.status AS ApplicationStatus
FROM Employee e
JOIN Application a ON e.id = a.employee_id
JOIN `User` u ON e.user_id = u.id
JOIN Person p ON u.applicant_id = p.id;
```

- Widok 7: Pracownicy i ilości aplikacji do nich przypisanych

```
CREATE VIEW EmployeeApplicationCount AS
SELECT
    E.id AS employee_id,
    U.username AS employee_username,
    COUNT(A.id) AS application_count
FROM
    Employee E
JOIN
    `User` U ON E.user_id = U.id
LEFT JOIN
    Application A ON E.id = A.employee_id
GROUP BY
    E.id, U.username;
```

4.2.3. Tworzenie procedur

- Procedura 1: Walidacja numeru PESEL

```
CREATE PROCEDURE ValidatePesel(IN p_PESSEL CHAR(11))
BEGIN
    DECLARE error_message VARCHAR(255);

    IF LENGTH(p_PESSEL) <> 11 OR NOT p_PESSEL REGEXP '^[0-9]+$' THEN
        SET error_message = 'PESEL is incorrect';
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
    END IF;
END ;
```

- Procedura 2: Walidacja adresu e-mail

```
CREATE PROCEDURE ValidateEmail(IN p_email_address VARCHAR(255))
BEGIN
    DECLARE error_message VARCHAR(255);

    IF NOT p_email_address REGEXP '^([A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,})$' THEN
        SET error_message = 'Email address is incorrect';
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
    END IF;
END;
```

- Procedura 3: Walidacja numeru telefonu

```
CREATE PROCEDURE ValidatePhone(IN p_phone_number INT)
BEGIN
    DECLARE error_message VARCHAR(255);

    IF LENGTH(p_phone_number) <> 9 OR NOT p_phone_number REGEXP '^([0-9])+$' THEN
        SET error_message = 'Phone number is incorrect';
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
    END IF;
END ;
```

- Procedura 4: Walidacja danych tekstowych

```
CREATE PROCEDURE ValidateTextData(IN p_text_data VARCHAR(255), IN fieldName VARCHAR(255))
BEGIN
    DECLARE error_message VARCHAR(255);

    -- Walidacja czy dane tekstowe nie zawierają znaków specjalnych i liczb
    IF p_text_data REGEXP '^[^A-Za-z ]+$' THEN
        SET error_message = CONCAT('Field ', fieldName, ' is incorrect. It should contain only letters.');
```

```
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
    END IF;
END;
```

- Procedura 5: Walidacja kodu pocztowego

```
CREATE PROCEDURE ValidatePostalCode(IN p_postal_code VARCHAR(6))
BEGIN
    DECLARE error_message VARCHAR(255);

    IF NOT p_postal_code REGEXP '^[0-9]{2}-[0-9]{3}$' THEN
        SET error_message = 'Postal code is incorrect. It should be in the format xx-xxx where x
is a digit.';
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
    END IF;
END;
```

- Procedura 6: Dodawanie nowego pracownika

```
CREATE PROCEDURE AddEmployee(IN p_user_id INT)
BEGIN
    DECLARE error_message VARCHAR(255);

    -- Sprawdzenie, czy użytkownik istnieje
    IF NOT EXISTS (SELECT 1 FROM `User` WHERE id = p_user_id) THEN
        SET error_message = 'User with that id doesnt exists';
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
    END IF;

    -- Dodanie nowego pracownika
    INSERT INTO Employee (user_id) VALUES (p_user_id);
END;
```


4.2.4. Tworzenie triggerów

- Trigger 1: Automatyczne przypisanie pracownika do wniosku. Trigger wybiera pracownika, który ma najmniej wniosków na swoim koncie. Dodatkowo, jeśli aplikant sam jest pracownikiem, wybiera innego pracownika. Trigger również sprawdza, czy to pierwsza aplikacja danego aplikanta - jeśli tak, tworzy dla niego nowe konto użytkownika w tabeli User, generując losowe hasło.

```
CREATE TRIGGER assign_employee_before_insert_application
BEFORE INSERT ON Application
FOR EACH ROW
BEGIN
    DECLARE available_employee_id INT;
    DECLARE applicant_pesel CHAR(11);
    DECLARE random_password VARCHAR(255);

    -- Znajduje pracownika z najmniejszą ilością wniosków
    SELECT e.id INTO available_employee_id
    FROM Employee e
    LEFT JOIN (SELECT employee_id, COUNT(*) AS application_count
               FROM Application
               GROUP BY employee_id) AS app_count ON e.id = app_count.employee_id
    ORDER BY app_count.application_count ASC, RAND()
    LIMIT 1;

    -- Sprawdza czy pracownik nie jest aplikantem tego wniosku
    IF NEW.applicant_id = available_employee_id THEN
        SELECT e.id INTO available_employee_id
        FROM Employee e
        WHERE e.id <> NEW.applicant_id
        ORDER BY RAND()
        LIMIT 1;
    END IF;

    -- Przypisuje pracownika do wniosku
    SET NEW.employee_id = available_employee_id;

    -- Sprawdza czy jest to pierwszy wniosek aplikanta
    SELECT COUNT(*) INTO @existing_applications
    FROM Application
    WHERE applicant_id = NEW.applicant_id;

    IF @existing_applications = 0 THEN
        -- Generowanie hasła
        SELECT SUBSTRING(MD5(RAND()), 1, 10) INTO random_password;

        -- Pobieranie numeru PESEL
        SELECT p.PESEL INTO applicant_pesel
        FROM Applicant
        JOIN Person p ON Applicant.person_id = p.id
        WHERE Applicant.id = NEW.applicant_id;

        -- Tworzenie nowego użytkownika
        INSERT INTO `User` (username, password, applicant_id)
        VALUES (applicant_pesel, random_password, NEW.applicant_id);
    END IF;
END;
```

- Trigger 2: Dodanie nowego mieszkańca do rejestru. Jeśli status aplikacji został zmieniony na „approved” i jest to aplikacja o wpis do rejestru to tworzony jest nowy mieszkaniec.

```
CREATE TRIGGER after_application_update
AFTER UPDATE ON Application
FOR EACH ROW
BEGIN
    DECLARE application_type INT;
    DECLARE person_id_for_resident INT;
    DECLARE user_id_for_resident INT;

    SELECT type INTO application_type FROM `Application Type` WHERE id =
NEW.application_type_id;

    -- Sprawdza czy status wniosku jest zatwierdzony i czy jest to wniosek o wpis do rejestru
    IF OLD.status <> 'approved' AND NEW.status = 'approved' AND application_type = 1 THEN

        -- Pobiera person_id z tabeli Applicant powiązanej z wnioskiem
        SELECT Applicant.person_id INTO person_id_for_resident
        FROM Applicant
        WHERE Applicant.id = NEW.applicant_id;

        -- Pobiera user_id z tabeli User powiązanej z wnioskiem
        SELECT id INTO user_id_for_resident
        FROM `User`
        WHERE applicant_id = NEW.applicant_id;

        -- Dodaje nowego mieszkańca
        INSERT INTO Resident (person_id, user_id)
        VALUES (person_id_for_resident, user_id_for_resident);
    END IF;
END;
```

- Trigger 3: Walidacja danych przez system przed dodaniem danych osobowych.

```
CREATE TRIGGER before_insert_person
BEFORE INSERT ON Person
FOR EACH ROW
BEGIN
    IF NEW.PESEL IS NOT NULL THEN
        CALL ValidatePesel(NEW.PESEL);
    END IF;

    CALL ValidateTextData(NEW.first_name, "first_name");

    IF NEW.second_name IS NOT NULL THEN
        CALL ValidateTextData(NEW.second_name, "second_name");
    END IF;

    CALL ValidateTextData(NEW.last_name, "last_name");

    IF NEW.place_of_birth IS NOT NULL THEN
        CALL ValidateTextData(NEW.place_of_birth, "place_of_birth");
    END IF;

    IF NEW.father_name IS NOT NULL THEN
        CALL ValidateTextData(NEW.father_name, "father_name");
    END IF;

    IF NEW.mother_name IS NOT NULL THEN
        CALL ValidateTextData(NEW.mother_name, "mother_name");
    END IF;

    IF NEW.mother_maiden_name IS NOT NULL THEN
        CALL ValidateTextData(NEW.mother_maiden_name, "mother_maiden_name");
    END IF;
END;
```

- Trigger 4: Walidacja danych przez system przed dodaniem aplikanta.

```
CREATE TRIGGER before_insert_applicant
BEFORE INSERT ON Applicant
FOR EACH ROW
BEGIN
    -- Walidacja e-mail
    CALL ValidateEmail(NEW.email_address);

    -- Walidacja numeru telefonu
    CALL ValidatePhone(NEW.phone_number);
END;
```

- Trigger 5: Walidacja danych przez system przed dodaniem danych adresowych.

```
CREATE TRIGGER before_insert_address
BEFORE INSERT ON Address
FOR EACH ROW
BEGIN
    CALL ValidatePostalCode(NEW.postal_code);
    CALL ValidateTextData(NEW.street, "street");
    CALL ValidateTextData(NEW.city, "city");
END;
```

4.3. Implementacja uprawnień i innych zabezpieczeń

4.3.1. Utworzenie ról

```
CREATE ROLE 'administrator_role';
CREATE ROLE 'manager_role';
CREATE ROLE 'employee_role';
```

4.3.2. Nadanie uprawnień

```
GRANT ALL PRIVILEGES ON del.* TO administrator_role;

GRANT SELECT, INSERT, UPDATE ON del.Application TO manager_role, employee_role;

GRANT SELECT, INSERT, UPDATE, DELETE ON del.`Application Type` TO manager_role;

GRANT SELECT ON del.`Application Type` TO employee_role;

GRANT SELECT, INSERT, UPDATE, DELETE ON del.`User` TO manager_role, employee_role;

GRANT SELECT, INSERT, UPDATE, DELETE ON del.Applicant TO manager_role, employee_role;

GRANT SELECT, INSERT, UPDATE, DELETE ON del.Resident TO manager_role, employee_role;

GRANT SELECT, INSERT, UPDATE, DELETE ON del.Person TO manager_role, employee_role;

GRANT SELECT, INSERT, UPDATE, DELETE ON del.Address TO manager_role, employee_role;

GRANT SELECT, INSERT, UPDATE, DELETE ON del.Employee TO manager_role;
```

4.4. Testowanie bazy danych na przykładowych danych

Testy zostały przeprowadzone w programie Visual Studio Code z pomocą dodatku MySQL

4.4.1. Testy triggerów

Zdjęcie 5: Test triggera „before_insert_address”

```
Execute
INSERT INTO Address (street, apartment_number, city, postal_code) VALUES ('Oak Ave2lnue', '404', 'Rivertown', '34-567') 4ms
Field street is incorrect. It should contain only letters.
```

Zdjęcie 6: Test triggera „before_insert_address”

```
Execute
INSERT INTO Address (street, apartment_number, city, postal_code) VALUES ('Oak Avenue', '404', 'Rivertown', '34-557'); 13ms
Postal code is incorrect. It should be in the format xx-xxx where x is a digit.
```

Zdjęcie 7: Test triggera „before_insert_person”

```
Execute
INSERT INTO Person (first_name, second_name, last_name, gender, PESEL, date_of_birth, place_of_birth, birth_certificate, death_certificate, civil_status_certificate, father_name, mother_name, mother_maiden_name, address_id) VALUES ('Frank', 'Lynn', 'Johnson', 'F', '5223069332', '1962-03-03', 'Lakeside', 'BirthCert#55544', 'DeathCert#12488', 'CivilCert#23642', 'Thomas', 'Jennifer', 'Smith', 1); 6ms
PESEL is incorrect
```

Zdjęcie 8: Test triggera „before_insert_applicant”

```
Execute
INSERT INTO Applicant (email_address, phone_number, person_id) VALUES ('user1inbox.com', 448905718, 1); 3ms
Email address is incorrect
```

Zdjęcie 9: Test triggera „before_insert_applicant”

```
Execute
INSERT INTO Applicant (email_address, phone_number, person_id) VALUES ('user1@inbox.com', -448905718, 1); 5ms
Phone number is incorrect
```

Zdjęcia 10 i 11: Test triggera „assign_employee_before_insert_application”

```
Execute
INSERT INTO Application (application_type_id, applicant_id, employee_id) VALUES (10, 1, NULL); 4ms
```

84	873	in review	2023-12-22 01:09:44	(NULL)	(NULL)	10	1	22
----	-----	-----------	---------------------	--------	--------	----	---	----

Zdjęcia 11, 12, 13 i 14: Test triggera „after_application_update”

Q	id int(10)	status varchar(2)	date_of_submission datetime	date_of_verification datetime	note varchar(1)	application_type_id int(10)	applicant_id int(10)	employee_id int(10)
64	853	in review	2023-12-22 00:42:32	(NULL)	(NULL)	1	81	4

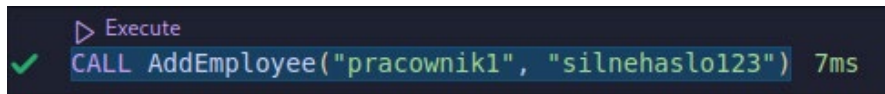
```
Execute
UPDATE Application SET status = "approved" where id = 853; 4ms
```

64	853	approved	2023-12-22 00:42:32	2023-12-22 01:30:49	(NULL)	1	81	4
----	-----	----------	---------------------	---------------------	--------	---	----	---

Q	id int(10)	person_id int(10)	user_id int(10)
1	2	81	244

4.4.2. Testy procedur

Zdjęcie 15: Test procedury „AddEmployee”



4.4.3. Testy widoków

Zdjęcie 16: Test widoku „UserApplicationInfo”

	* UserID int(10)	* Username varchar(255)	* ApplicationID int(10)	* ApplicationStatus varchar(255)	* SubmissionDate datetime	VerificationDate datetime	ApplicationNote varchar(1000)	* ApplicationTypeName varchar(255)
1	208	51	817	in review	2023-12-22 00:42:32	(NULL)	(NULL)	wpis_do_rejestru
2	213	57993435265	822	in review	2023-12-22 00:42:32	(NULL)	(NULL)	wpis_do_rejestru
3	229	18496398799	838	in review	2023-12-22 00:42:32	(NULL)	(NULL)	wpis_do_rejestru
4	237	73811869790	846	in review	2023-12-22 00:42:32	(NULL)	(NULL)	wpis_do_rejestru
5	238	83265141758	847	in review	2023-12-22 00:42:32	(NULL)	(NULL)	wpis_do_rejestru
6	244	49782625797	853	approved	2023-12-22 00:42:32	2023-12-22 01:30:49	(NULL)	wpis_do_rejestru

Zdjęcie 17: Test widoku „EmployeeTaskAssignment”

	* Employee int(10)	* EmployeeFirstName varchar(255)	* EmployeeLastName varchar(255)	* AssignedApplicationID int(10)	* ApplicationStatus varchar(255)
1	22	Frank	Johnson	873	approved

Zdjęcie 18: Test widoku „ResidentContactInfo”

	* ResidentID int(10)	* first_name varchar(255)	* last_name varchar(255)	PESEL char(11)	* street varchar(255)	* apartment_number varchar(20)	* city varchar(255)	* postal_code char(6)	* email_address varchar(255)	* phone_number int(9)
1	2	Henry	Williams	49782625797	Cedar Drive	909	Hillcrest	56-789	user81@example.com	120678072
2	1	Grace	Williams	81437518189	Oak Avenue	303	Springfield	78-901	user99@service.com	971000201

Zdjęcie 19: Test widoku „UserRoleAccess”

	* UserID int(10)	* username varchar(255)	* UserRole varchar(9)
19	19	nathanperez	Employee
20	20	avaedwards	Employee
21	129	52290569332	Employee
22	130	83524897814	Applicant
23	131	24972167213	Applicant
24	132	45676008916	Applicant

Zdjęcie 20: Test widoku „EmployeeApplicationCount”

	* employee_id int(10)	* employee_username varchar(255)	* application_count bigint(21)
1	1	janedoe	4
2	2	johnsmith	4
3	3	alicejones	4
4	4	bobross	5
5	5	emilybrown	4
6	6	laurasmith	4
7	7	williamjones	4

Zdjęcie 21: Test widoku „ApplicationTypeSummary”

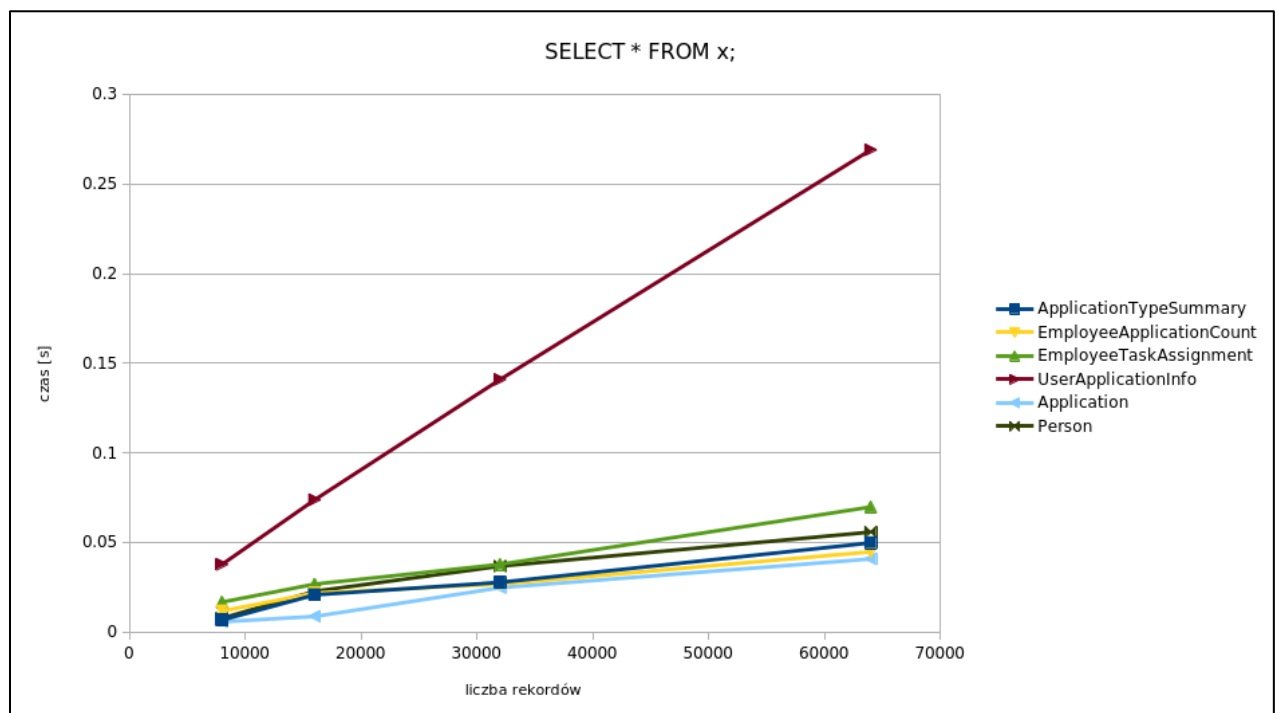
	* type_name varchar(255)	* NumberOfApplicator bigint(21)
1	info_o_smierci	842
2	wpis_do_rejestru	787
3	wpis_dziecka_do_rejestr	740
4	wypis_z_rejestru	787
5	wyrobiecie_pesel	834
6	zmiana_imienia_nazwisk	795
7	zmiana_pesel	793
8	zmiana_plci	826
9	zmiana_stanu_cywilneg	828
10	zmiana_zameldowania	768

4.4.3. Testy wydajności zapytań

Tabela 1: Test wydajności zapytań SELECT * FROM x;

	liczba rekordów			
	8000	16000	32000	64000
ApplicationTypeSummary [s]	0.007	0.021	0.028	0.05
EmployeeApplicationCount [s]	0.012	0.022	0.027	0.045
EmployeeTaskAssignment [s]	0.017	0.027	0.038	0.07
UserApplicationInfo [s]	0.038	0.074	0.141	0.269
Application [s]	0.006	0.009	0.025	0.041
Person [s]	0.008	0.023	0.037	0.056

Wykres 1: Test wydajności zapytań



5. Implementacja i testy aplikacji

5.1. Instalacja i konfigurowanie systemu

- Sklonuj repozytorium z githuba: <https://github.com/Pjurkowski/PRD>
- W katalogu "PRD\backend>" zmodyfikuj plik „database.py”, wpisując w pole „host” swój adres ip:

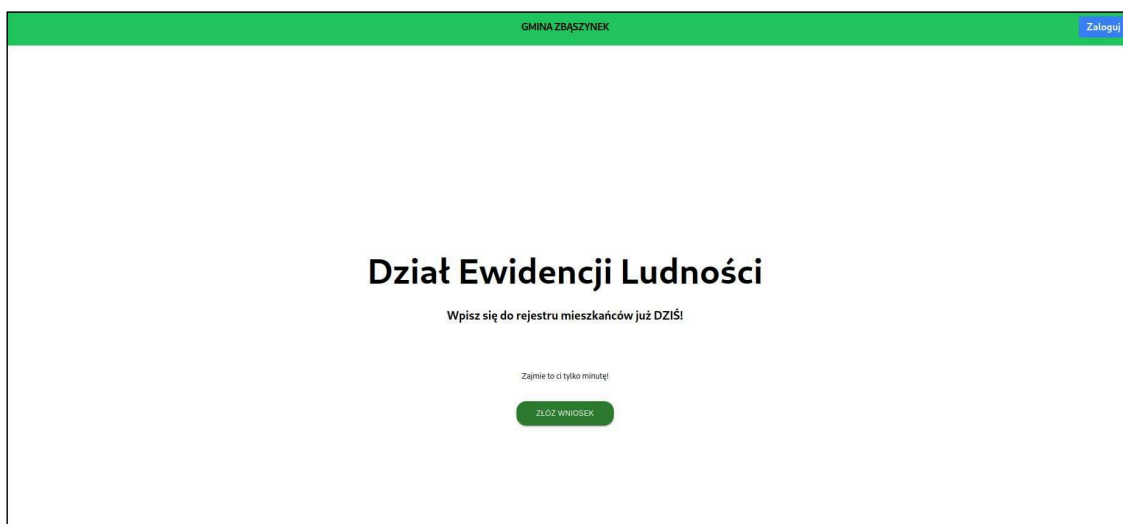
```
def connect():  
    connection = mysql.connector.connect(  
        host='192.168.0.178',  
        user='root',  
        password='test123',  
        connect_timeout=10,  
        database='del'  
    )  
    return connection
```

Zdjęcie 22: Edycja ip hosta

- Będąc w katalogu „PRD\frontend>” uruchom komendę: npm install
- W katalogu "PRD\backend>" utwórz plik „.env” zawierający następujące kluczowe zmienne środowiskowe:
 - 1) secret - Tajny klucz wykorzystywany do kodowania tokenów JWT
 - 2) algorithm - Algorytm wykorzystywany do kodowania tokenów JWT, np. "HS256"
- Będąc w katalogu „PRD>” uruchom komendy:
docker-compose -f docker-compose.yml build
docker-compose -f docker-compose.yml up
- W przeglądarce wejdź na adres: <http://localhost:5173/>

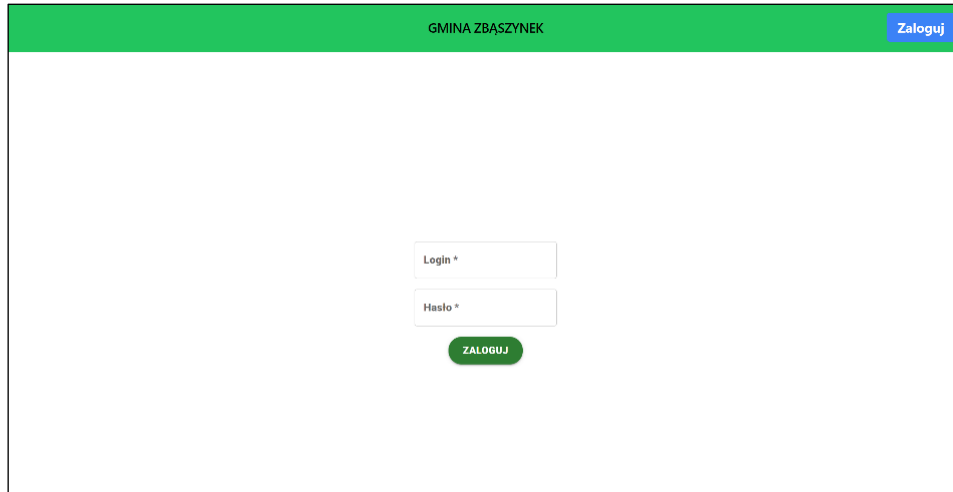
5.2. Instrukcja użytkowania aplikacji

- Widok po wejściu na stronę główną:



Zdjęcie 22: Strona główna

- Będąc na stronie główne są dwie możliwości:
 1. Osoba widnieje już w rejestrze mieszkańców lub wcześniej ubiegała się o wpis do niego, może się zalogować, klikając w prawym górnym rogu przycisk „Zaloguj”, który przeniesie ją do strony logowania:



The screenshot shows a web page for Gmina Zbąszynek. At the top, there is a green navigation bar containing the text 'GMINA ZBĄSZYNEK' on the left and a blue button labeled 'Zaloguj' on the right. The main body of the page is white and contains a centered login form. This form consists of two input fields: the first is labeled 'Login *' and the second is labeled 'Hasło *'. Below these fields is a green button with the text 'ZALOGUJ' in white capital letters.

Zdjęcie 23: Strona logowania

2. Osoba chce złożyć wniosek o wpis do rejestru, klikając przycisk „Złóż wniosek”, który przeniesie ją do strony z formularzem:



The screenshot shows a web page for Gmina Zbąszynek. At the top, there is a green navigation bar containing the text 'GMINA ZBĄSZYNEK' on the left and a blue button labeled 'Zaloguj' on the right. The main body of the page is white and features the title 'Formularz składania wniosku' in bold black text. Below the title is a dropdown menu with the placeholder text 'Typ wniosku' and a small downward arrow on the right side.

Zdjęcie 24: Strona z formularzem składania wniosku

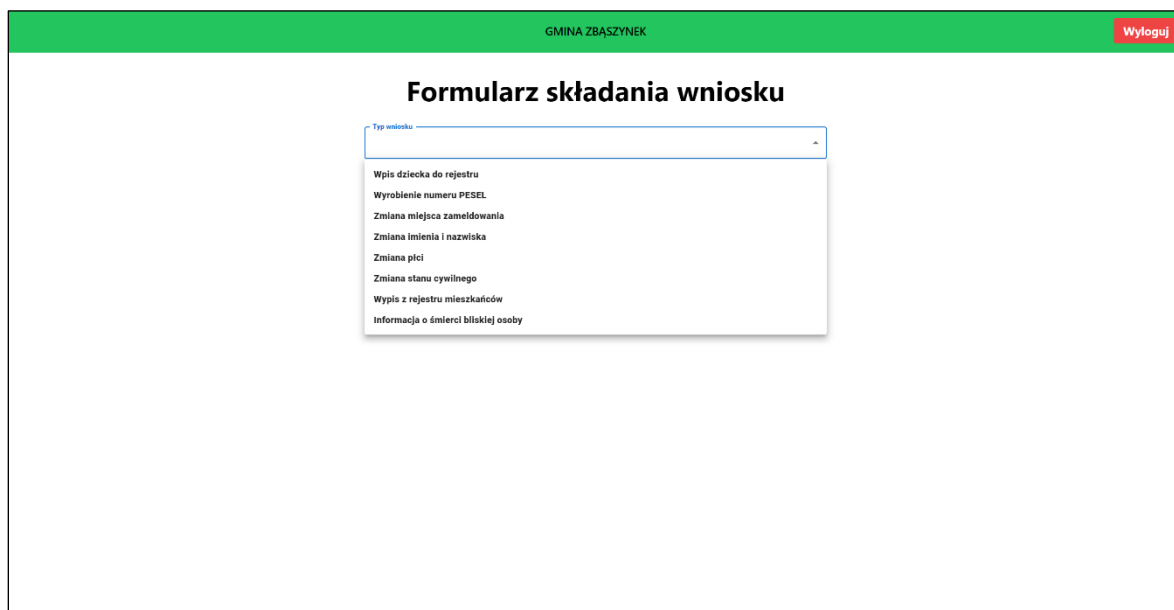
W polu „Typ wniosku” musi wybrać jedyną dostępną na ten moment opcję „Wpis do rejestru” i wypełnić wymagane pole odpowiednimi danymi:

Zdjęcie 24: Strona z formularzem składania wniosku o wpis do rejestru

- Po złożeniu swojego pierwszego wniosku osoba może już się zalogować do systemu, klikając w prawym górnym rogu przycisk „Zaloguj”, jako login wpisując swój PESEL i jako hasło wpisując wygenerowany przez system ciąg znaków, który został dostarczony na adres e-mail podany w formularzu przez tą osobę. Po zalogowaniu osoba może przeglądać swoje wnioski w panelu użytkownika jak i również składać nowe:

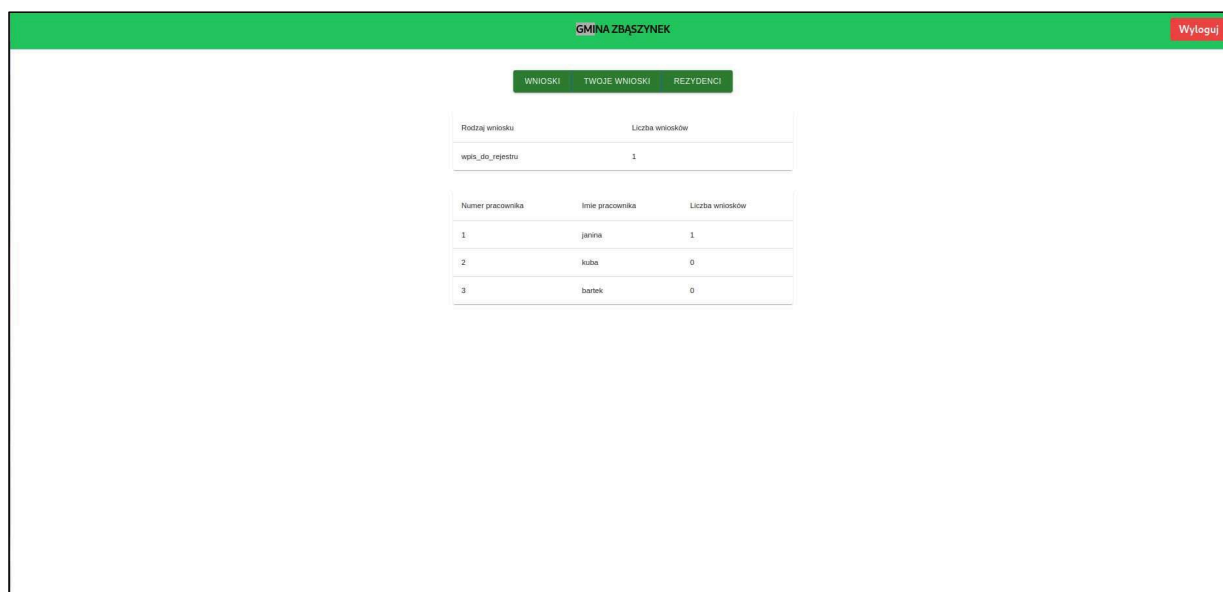
Zdjęcie 25: Strona z panelem użytkownika

Zalogowany użytkownik po kliknięciu przycisku „Złóż nowy wniosek” ma już możliwość złożenia wniosku każdego typu:



Zdjęcie 26: Strona z formularzami składania wniosków

- Jeżeli osoba logująca się na naszej stronie jest pracownikiem, przejdzie do panelu pracownika, w którym widnieje lista wszystkich pracowników, rodzaj i ilość wniosków rozpatrzona przez tego pracownika oraz możliwość sprawdzenia swoich własnych wniosków, rozpatrzonych przez siebie wniosków oraz listę mieszkańców gminy:



Rodzaj wniosku	Liczba wniosków
wpis_dla_rejestru	1

Numer pracownika	Imię pracownika	Liczba wniosków
1	janina	1
2	karla	0
3	bartek	0

Zdjęcie 27: Strona z panelem pracownika

Zalogowanemu pracownikowi po kliknięciu przycisku „Wnioski” pojawi się lista rozpatrzonych przez siebie wniosków:

GMINA ZBĄSZYNEK						Wyloguj
Panel Pracownika						
ID	Status	Data złożenia	Data weryfikacji	Rodzaj wniosku		
1	approved	2024-01-19T10:00:31	2024-01-19T10:02:12	wpis_do_rejestru	PRZEGLĄDAJ	
Rows per page: 100 1-1 of 1 < >						

Zdjęcie 28: Strona z rozpatrzonymi wnioskami przez pracownika

Zalogowanemu pracownikowi po kliknięciu przycisku „Rezydenci” pojawi się lista mieszkańców gminy widniejących w rejestrze:

GMINA ZBĄSZYNEK						Wyloguj
Lista Mieszkańców						
ID	Imię	Nazwisko	PESEL	Email	Numer telefonu	
1	Jasmina	Brown	23456789012	jasmina@example.com	987654321	
2	Kuba	Zakubowski	23456789013	kuba@example.com	123456789	
3	Bartek	Straut	23456789014	bartek@example.com	123456788	
4	Lukasz	Widwiał	12312312312	lukaszwidwial01@gmail.com	532080942	
Rows per page: 100 1-4 of 4 < >						

5.3. Testowanie opracowanych funkcji systemu

- Nieprawidłowe wprowadzenie danych podczas składania wniosku o wpis do rejestru:

- Testowanie API bezpośrednio z poziomu przeglądarki:

Jedną z głównych zalet FastAPI wykorzystanego w tym projekcie jest to, że automatycznie generuje dokumentację dla API przy użyciu Swagger UI. Dzięki temu użytkownicy mogą łatwo zrozumieć, jak z niego korzystać, jakie są dostępne endpointy, jakie parametry przyjmują, jakie są oczekiwane formaty danych wejściowych i wyjściowych, a także przetestować API bezpośrednio w przeglądarce.

Aby się tam dostać należy wpisać adres: <http://localhost:8000/docs>

default			^
GET	/api/applications	Get Applications	🔒
GET	/api/application/{id}	Get Application	🔒
GET	/api/employee_applications	Get Employee Applications	🔒
GET	/api/user_applications	Get User Applications	🔒
GET	/api/user	Get User	🔒
GET	/api/is_employee	Get Is Employee	🔒
GET	/api/application_types	Get Application Types	🔒
POST	/api/address	Post Address	🔒
POST	/api/person	Post Person	🔒
POST	/api/applicant	Post Applicant	🔒
POST	/api/full_application	Post Full Application	🔒
POST	/api/application_verification	Post Application Verification	🔒
POST	/api/application	Post Application	🔒
POST	/api/login	Login	🔒

Schemas			^
AddressSchema	>	Expand all	object
ApplicantSchema	>	Expand all	object
ApplicationSchema	>	Expand all	object
ApplicationVerificationSchema	>	Expand all	object
FullApplicationSchema	>	Expand all	object
HTTPValidationError	>	Expand all	object
PersonSchema	>	Expand all	object
UserLoginSchema	>	Expand all	object
ValidationError	>	Expand all	object

Przykładowy schemat dla przesłania aplikacji:

FullApplicationSchema  Collapse all object

```
street* string
apartment_number* string
city* string
postal_code* string
first_name* string
second_name > Expand all (string | null)
last_name* string
gender* string
pesel > Expand all (string | null)
date_of_birth > Expand all (string | null)
place_of_birth > Expand all (string | null)
birth_certificate > Expand all (string | null)
death_certificate > Expand all (string | null)
civil_status_certificate > Expand all (string | null)
father_name > Expand all (string | null)
mother_name > Expand all (string | null)
mother_maiden_name > Expand all (string | null)
email_address* string
phone_number* string
application_type_id* integer
```

Example

```
{
  "apartment_number": "69/1d",
  "application_type_id": 1,
  "birth_certificate": "12345678901",
  "city": "wroclaw",
  "civil_status_certificate": "12345678901",
  "date_of_birth": "1999-01-01",
  "death_certificate": "12345678901",
  "email_address": "essatron@essa.pl",
  "father_name": "Jan",
  "first_name": "Jan",
  "gender": "M",
  "last_name": "Kowalski",
  "mother_maiden_name": "Kowalska",
  "mother_name": "Anna",
  "pesel": "12345678901",
  "phone_number": "123456789",
  "place_of_birth": "wroclaw",
  "postal_code": "53-530",
  "second_name": "Kazimierz",
  "street": "Nikolaja reja"
}
```

Przykładowy test dla przesłania aplikacji:

POST /api/full_application Post Full Application

Parameters

No parameters

Request body **required** application/json

```
{
  "apartment_number": "69/1d",
  "application_type_id": 1,
  "birth_certificate": "12345678901",
  "city": "wroclaw",
  "civil_status_certificate": "12345678901",
  "date_of_birth": "1999-01-01",
  "death_certificate": "12345678901",
  "email_address": "essatron@essa.pl",
  "father_name": "Jan",
  "first_name": "Jan",
  "gender": "M",
  "last_name": "Kowalski",
  "mother_maiden_name": "Kowalska",
  "mother_name": "Anna",
  "pesel": "12345678901",
  "phone_number": "123456789",
  "place_of_birth": "wroclaw",
  "postal_code": "53-530",
  "second_name": "Kazimierz",
  "street": "Nikolaja reja"
}
```

Execute Clear

Responses

Curl

```
curl -X POST \
  http://localhost:8000/api/full_application \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "apartment_number": "69/1d",
    "application_type_id": 1,
    "birth_certificate": "12345678901",
    "city": "wroclaw",
    "civil_status_certificate": "12345678901",
    "date_of_birth": "1999-01-01",
    "death_certificate": "12345678901",
    "email_address": "essatron@essa.pl",
    "father_name": "Jan",
    "first_name": "Jan",
    "gender": "M",
    "last_name": "Kowalski",
    "mother_maiden_name": "Kowalska",
    "mother_name": "Anna",
    "pesel": "12345678901",
    "phone_number": "123456789",
    "place_of_birth": "wroclaw",
    "postal_code": "53-530",
    "second_name": "Kazimierz",
    "street": "Nikolaja reja"
  }'
```

Request URL

http://localhost:8000/api/full_application

Server response

Code Details

501 Error: Not Implemented

Response body

```
{
  "detail": "Invalid data"
}
```

Response headers

```
access-control-allow-credentials: true
access-control-allow-origin: *
content-length: 25
content-type: application/json
date: Thu, 18 Jan 2024 18:06:53 GMT
server: uvicorn
```

Responses

Code Description

200 Successful Response

Media type

application/json

Controls Accept header

Example Value Schema

```
"string"
```

Code Description

422 Validation Error

Media type

application/json

Controls Accept header

Example Value Schema

```
{
  "detail": [
    {
      "loc": [
        "string",
      ],
      "msg": "string",
      "type": "string"
    }
  ]
}
```

5.4. Omówienie wybranych rozwiązań programistycznych

5.4.1. Implementacja interfejsu dostępu do bazy danych

Do połączenia z bazą danych MySQL została użyta biblioteka „mysql.connector”. Połączenie jest konfigurowane za pomocą adresu IP hosta, nazwy użytkownika, hasła i nazwa bazy danych w pliku „database.py”:

```
import mysql.connector

def connect():
    connection = mysql.connector.connect(
        host='192.168.0.178',
        user='root',
        password='test123',
        connect_timeout=10,
        database='del'
    )
    return connection
```

Konfiguracja serwisu bazy danych („db”) zawarta jest w pliku „docker-compose.yml”. Dla kontenera bazy danych użyto obrazu „mysql:8”, a zmienne środowiskowe takie jak „MYSQL_USER”, „MYSQL_ROOT_PASSWORD”, „MYSQL_DATABASE” są użyte do konfiguracji bazy danych. Port „3306” jest mapowany, co pozwala na dostęp do bazy danych z poziomu hosta. Serwisy „web” i „api” są skonfigurowane jako zależne od serwisu „db”, co zapewnia, że baza danych jest dostępna, zanim te serwisy zostaną uruchomione. Z kolei „my-db:/var/lib/mysql” jest używany do przechowywania danych bazy danych, co zapewnia trwałość danych poza cyklem życia kontenera. Wzorzec ten umożliwia elastyczne zarządzanie połączeniami z bazą danych, ponieważ wszystkie parametry połączenia są skoncentrowane w jednym miejscu, co ułatwia ich modyfikację i zarządzanie.

```
version: "3.8"
services:
  db:
    image: mysql:8
    environment:
      MYSQL_USER: user
      MYSQL_ROOT_PASSWORD: test123
      MYSQL_DATABASE: del
    ports:
      - "3306:3306"
    volumes:
      - my-db:/var/lib/mysql
  web:
    build: ./frontend
    ports:
      - "5173:5173"
    volumes:
      - ./frontend:/app
  api:
    build: ./backend
    ports:
      - "8000:8000"
    depends_on:
      - db
    volumes:
      - ./backend:/app
    restart: on-failure
volumes:
  my-db:
```

5.4.2. Implementacja wybranych funkcjonalności systemu

- Router w FastAPI:

Mechanizm pozwalający na organizację grup endpointów. Pozwala na lepsze zarządzanie kodem i strukturę projektu, szczególnie gdy aplikacja zawiera wiele różnych ścieżek i funkcji.

Przykłady zaimplementowanych funkcji z użyciem routera:

```
@router.get("/application/{id}", dependencies=[Depends(JWTBearer())])
async def get_application(id: int):
    query = "SELECT * FROM FullApplicationInfo WHERE application_id=%s"
    cursor.execute(query, (id,))
    result = cursor.fetchone()

    if result is None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
                             detail="Application not found")

    return result
```

Funkcja „get_application” jest endpointem API do pobierania szczegółów pojedynczej aplikacji na podstawie jej ID i wymaga JWT dla dostępu (używa „Depends(JWTBearer())”). Obsługuje żądanie „HTTP GET”. Na samym początku wykonuje zapytanie SQL do bazy danych, aby pobrać wszystkie informacje o aplikacji o określonym ID („application_id=%s”). Następnie używa „cursor.execute” do wykonania zapytania z podanym ID. Wynik przechowuje w zmiennej „result”, a pobiera go z bazy danych za pomocą „cursor.fetchone()”. Jeśli aplikacja zostanie znaleziona zwraca dane, w przeciwnym razie (wynik to „None”), zwraca błąd 404 z odpowiednim komunikatem.

```
@router.post("/address")
async def post_address(address: AddressSchema):
    query = "INSERT INTO Address (street, apartment_number, city, postal_code) VALUES (%s, %s, %s, %s)"
    cursor.execute(query, (address.street, address.apartment_number,
                           address.city, address.postal_code))
    conn.commit()

    item_id = cursor.lastrowid

    cursor.execute("SELECT * FROM Address WHERE id = %s", (item_id,))
    result = cursor.fetchone()

    if result is None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
                             detail="Address not found after insertion")

    return result
```

Funkcja „post_address” jest endpointem API do dodawania nowego adresu do bazy danych. Obsługuje żądanie „HTTP POST”. Przyjmuje obiekt „address”, który jest instancją „AddressSchema”. Wykonuje zapytanie SQL, aby dodać nowy adres do tabeli „Address” w bazie danych. Po jego dodaniu, funkcja pobiera go za pomocą „cursor.lastrowid” i zwraca jako odpowiedź. Jeśli adres nie zostanie znaleziony po dodaniu, funkcja zwraca błąd HTTP 404.

- Definiowanie klas modelowych w Pydantic (plik „model.py”):

Służą do strukturyzowania i walidacji danych wejściowych w aplikacji FastAPI. Każda klasa reprezentuje schemat danych dla różnych typów obiektów, takich jak użytkownicy, adresy, osoby, wnioskodawcy, aplikacje, pełne aplikacje i weryfikacja aplikacji. Klasa wykorzystuje „BaseModel” z Pydantic, co zapewnia automatyczną walidację danych wejściowych i ułatwia błyskawiczną serializację/deserializację danych JSON.

Przykładowa implementacja:

```
class AddressSchema(BaseModel):
    street: str = Field(...)
    apartment_number: str = Field(...)
    city: str = Field(...)
    postal_code: str = Field(...)
    class Config:
        json_schema_extra = {
            "example": {
                "street": "Mikolaja reja",
                "apartment_number": "69/1d",
                "city": "Wroclaw",
                "postal_code": "53-530"
            }
        }
```

"AddressSchema" dziedziczy po "BaseModel" z Pydantic, co oznacza, że korzysta z funkcji tej biblioteki do walidacji i serializacji danych. Definiuje pola tekstowe odpowiadające komponentom adresu, takie jak "street", "apartment_number", "city", "postal_code". Użycie "Field(...)" w definicji każdego pola oznacza, że pole to jest wymagane. Pydantic automatycznie sprawdza, czy dane przekazane do modelu "AddressSchema" są zgodne z określonymi typami danych i czy wymagane pola nie są puste. W klasie "Config" wewnątrz "AddressSchema", zdefiniowano "json_schema_extra", zawierający przykładowe dane wykorzystywane do dokumentacji i testów, pomagając innym deweloperom lub narzędziom zrozumieć, jakie dane są oczekiwane przez model.

- Prywatne trasy (plik „PrivateRoutes.jsx”):

W aplikacjach internetowych są mechanizmem kontrolującym dostęp do określonych sekcji lub stron aplikacji. Ich głównym celem jest zapewnienie, że tylko uwierzytelnieni i upoważnieni użytkownicy mogą uzyskać dostęp do pewnych zasobów lub funkcjonalności.

```
import { Outlet, Navigate } from "react-router-dom";

const PrivateRoutes = () => {
    function getToken() {
        const tokenString = sessionStorage.getItem("token");
        const userToken = JSON.parse(tokenString);
        return userToken?.access_token;
    }
    return getToken() ? <Outlet /> : <Navigate to="/login" />;
};

export default PrivateRoutes;
```

Funkcja "getToken" próbuje pobrać token uwierzytelniający z "sessionStorage" przeglądarki. Jeśli token jest obecny (co wskazuje, że użytkownik jest zalogowany), to "PrivateRoutes" renderuje komponent "Outlet" pozwalający na dalsze renderowanie zagnieżdżonych tras zdefiniowanych w aplikacji. Jeśli token nie istnieje (użytkownik nie jest zalogowany), komponent "Navigate" przekierowuje użytkownika na stronę logowania ("/login").

5.4.3. Implementacja mechanizmów bezpieczeństwa

- Implementacja CORS (plik „main.py”):

CORS (Cross-Origin Resource Sharing) jest mechanizmem, który pozwala lub odrzuca żądania zasobów na stronie internetowej z innych domen niż domena, z której pochodzi pierwszy zasób. Jest to kluczowy aspekt bezpieczeństwa i kontroli dostępu w aplikacjach internetowych.

```
from fastapi.middleware.cors import CORSMiddleware

app = FastAPI()

origins = ["*"]

app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

Lista „origins” określa, które źródła są dozwolone do dostępu do API. W tym przypadku, „*” oznacza akceptację żądań ze wszystkich źródeł.

- Walidacja danych na frontendzie (plik „ApplicationForm.jsx”):

Jest kluczowym aspektem każdego formularza w aplikacji internetowej. Zapewnia, że dane wprowadzane przez użytkowników są w odpowiednim formacie i spełniają określone kryteria, zanim zostaną przesłane do serwera. Jest to ważne zarówno dla użyteczności aplikacji, jak i dla bezpieczeństwa, ponieważ pomaga zapobiegać błędom i potencjalnym atakom.

Do śledzenia błędów w formularzu został użyty hook „useState” (przechowuje błędy walidacji dla poszczególnych pól):

```
const [errors, setErrors] = useState({});
```

Przykładem funkcji walidującej dane jest funkcja walidująca wprowadzony adres e-mail:

```
const validateEmail = () => {
  const regex = /^[a-zA-Z0-9]+@[a-zA-Z0-9]+\.[A-Za-z]+$/;
  validateInput(email_address.current, regex);
};
```

Zostało w niej wykorzystane wyrażenie regularne „regex”, które sprawdza, czy adres e-mail składa się z: jednej lub więcej liter alfabetu łacińskiego lub cyfr przed znakiem „@”, po znaku „@”, kolejny ciąg liter i cyfr, kropki „.” po której następuje jeden lub więcej liter alfabetu.

Natomiast w polach „TextField” wyświetlane są komunikaty o błędach, dzięki którym użytkownik od razu po złożeniu wniosku otrzymuje komunikat o niepoprawności wprowadzonych przez niego danych:

```
<TextField
    id="phone_number"
    required
    label="Numer telefonu"
    inputRef={phone_number}
    error={errors.phone_number}
    helperText={
        errors.phone_number &&
        "Numer telefonu może zawierać tylko cyfry"
    }
/>
```

- Tokeny JWT (JSON Web Token):

Są szeroko stosowane w aplikacjach internetowych do bezpiecznego przekazywania informacji między klientem a serwerem. Są one szczególnie użyteczne do uwierzytelniania i zarządzania sesjami użytkowników.

Generacja tokenów w projekcie (plik „auth.py”):

```
import jwt
from decouple import config

JWT_SECRET = config("secret")
JWT_ALGORITHM = config("algorithm")

def sign_jwt(data: str):
    payload = {"data": data, "expires": time.time() + 300}
    token = jwt.encode(payload, JWT_SECRET, algorithm=JWT_ALGORITHM)
    return token_response(token)
```

Token JWT jest generowany przy użyciu biblioteki „jwt”. Token zawiera „payload” zawierający dane użytkownika oraz informacje o wygaśnięciu tokena, który jest szyfrowany przy użyciu tajnego klucza (JWT_SECRET) i algorytmu (JWT_ALGORITHM).

Zastosowanie Tokenów JWT w projekcie:

- 1) Uwierzytelnianie Użytkowników: Po pomyślnym zalogowaniu się, użytkownik otrzymuje token JWT, który następnie używany jest do dostępu do chronionych zasobów.
- 2) Zarządzanie Sesjami: Tokeny JWT mogą zawierać informacje o czasie wygaśnięcia, co pozwala na kontrolę aktywności sesji użytkownika.
- 3) Bezpieczeństwo: JWT oferuje bezpieczny sposób na przechowywanie i weryfikację tożsamości użytkownika, ponieważ jest trudny do podrobienia ze względu na użycie algorytmu szyfrującego i tajnego klucza.

6. Podsumowanie i wnioski

Skończony projekt oparty o zbudowanie bazy danych dla Działu Ewidencji Ludności oraz strony internetowej pokazuje, że wdrożenie automatyzacji może znacznie zwiększyć wydajność pracy urzędników, zmniejszając czas poświęcony na rutynowe zadania i pozwalając na lepsze wykorzystanie zasobów ludzkich. Dodatkowo dostarcza on wygodę użytkowania zarówno dla pracowników jak i mieszkańców (lub potencjalnie starających się osób o zameldowanie w gminie), co może przełożyć się na zwiększenie atrakcyjności zatrudnienia w urzędzie oraz wzmożonej migracji nowych osób na teren gminy. Interfejs użytkownika jak i pracownika jest prosty i przejrzysty, stąd osoba w każdym wieku powinna dać sobie radę w jego obsłudze. W Dziale Ewidencji Ludności przetwarzanych jest wiele wrażliwych informacji, stąd duży nacisk został położony na bezpieczeństwo danych, implementując mechanizmy szyfrowania, uwierzytelniania i ochrony przed nieautoryzowanym dostępem. Użycie nowoczesnych technologii, takich jak Python FastAPI, JavaScript React i MySQL zapewnia wydajność, stabilność oraz możliwość rozbudowy systemu w przyszłości o nowe funkcjonalności, jak i również przygotowanie go na znaczący wzrost liczby ludności i przetwarzanych danych dla gminy. Zaimplementowany system dodatkowo zapewnia jakość i spójność danych, dzięki zastosowaniu procedur walidacji i ograniczeń integralności danych, co jest niezbędne aby dostarczać rzetelnych i dokładnych informacji. Wnikliwe testowanie pod kątem różnorodnych scenariuszy użycia potwierdziło niezawodność systemu i jego gotowość do wdrożenia.

7. Literatura

1) Dokumentacja używanych technologii:

<https://dev.mysql.com/doc/>

<https://fastapi.tiangolo.com/learn/>

<https://react.dev/learn>

2) Wsparcie merytoryczne:

<https://www.wroclaw.pl/urząd/zameldowanie>

<https://bip.um.wroc.pl/artykul/227/3174/wydzial-spraw-obywatelskich>

<https://www.gov.pl/web/gov/zalatwiaj-sprawy-urzedowe-przez-internet-na-epuap>