# POLITECHNIKA WROCŁAWSKA

# WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI

# PROJEKT Z BAZ DANYCH

## DZIENNIK ELEKTRONICZNY *Gejzer*

Termin zajęć:

- Środa, 11:15-13:00

Autorzy:

- Miłosz Bedryło 272934
- Artur Kręgiel 272989
- Dominik Pokrzywa 272953
- Mateusz Głuchowski 272885

Prowadzący:

- dr inż. Roman Ptak, W4

Wrocław, 2025 r.

# Spis treści

---

# 1. Wstęp

## 1.1 Cel projektu

- Projekt ma na celu stworzenie systemu, który ułatwi zarządzanie danymi, nauczania, komunikacją w szkole. Baza danych będzie centralnym elementem systemu, aplikacja dostępowa oferująca różne funkcje dla uczniów, rodziców, nauczycieli oraz administracji.

---

## 1.2 Zakres projektu

- Zakresem naszego projektu będzie realizacja systemu dziennika, jako aplikacja webowa korzystająca z frameworków: spring boot, react, chakra ui. Jako baza danych dla naszego projektu zostanie użyty PostgreSQL.

---

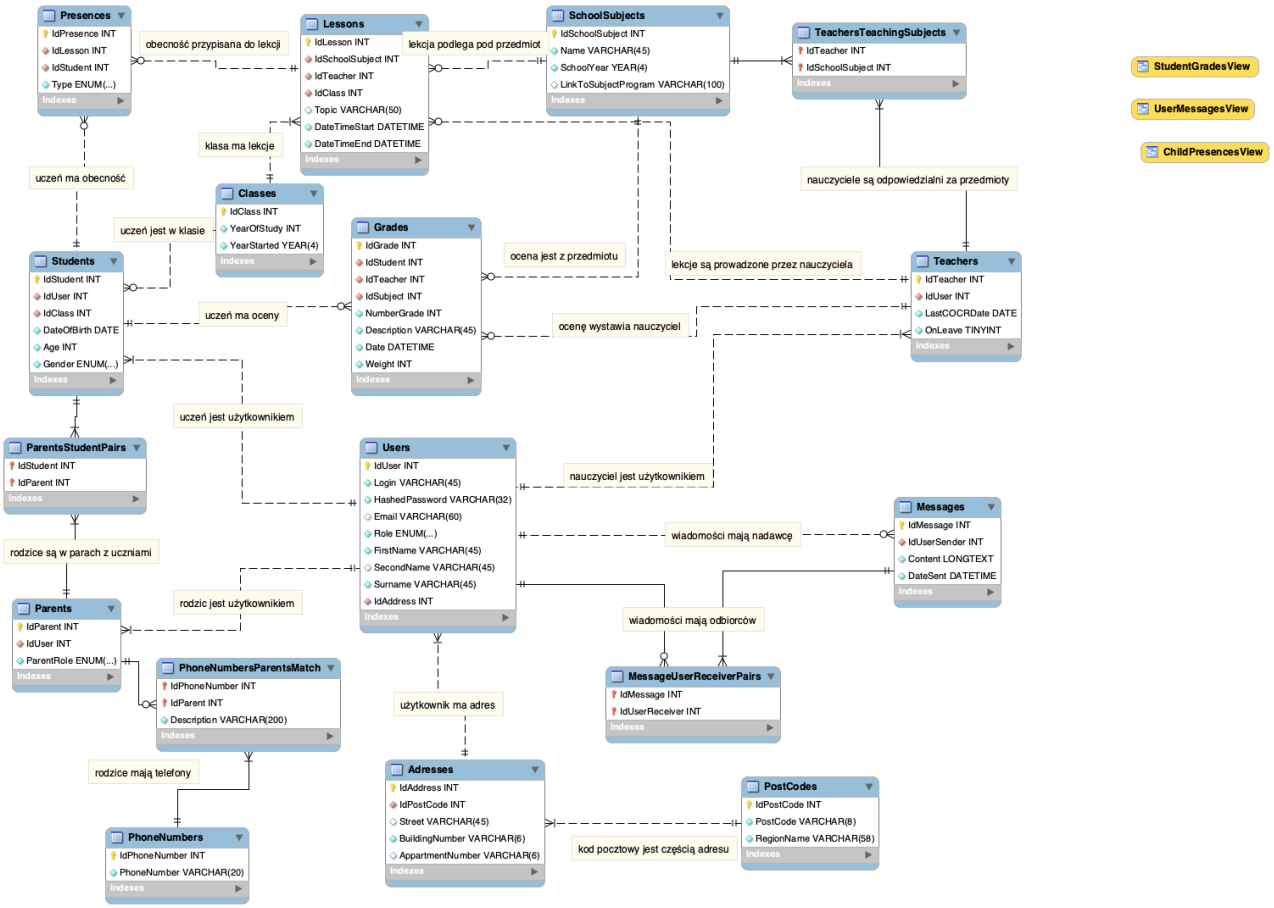
# 2. Analiza wymagań

- W ramach przygotowania do realizacji projektu, przeprowadzona została analiza wymagań użytkowników dziennika online, polegała ona na przeanalizowaniu rozwiązań znanych nam osobiście ze

szkół średnich oraz ustaleniu zakresu naszego projektu.

## 2.1 Opis działania i schemat logiczny systemu

- System będzie działał jako aplikacja webowa składająca się z oddzielnego backendu i frontendu komunikujących się przy pomocy REST API, a bezpośredni dostęp do bazy danych ma tylko backend, który dba też o autentykację użytkowników.




*Rysunek 1. Schemat logiczny systemu.*

## 2.2 Wymagania funkcjonalne

- Panel ucznia
    - możliwość przeglądania planu lekcji
    - możliwość przeglądania zapowiedzianych sprawdzianów, kartkówek
    - możliwość przeglądania ocen oraz średniej
    - możliwość przeglądania obecności na lekcjach
    - otrzymywanie i przeglądanie wiadomości od nauczyciela
- Panel rodzica
    - posiada pełną funkcjonalność ucznia
    - możliwość przełączania między dziećmi
    - mozliwość wysłania usprawiedliwienia
    - możliwość korespondowania z nauczycielem

- Panel nauczyciela
    - możliowść wglądu do planów lekcji
    - uzupełnianie oraz przeglądanie listy obecności
    - wystawianie ocen
    - wysyłanie ogłoszeń
    - prowadzenie korespondecji z rodzicami (CRUD)
- Panel dyrektora (administrator merytoryczny)
    - dodaj usuń edytuj (uczniow nauczycieli rodziców )
    - umieszczanie ogloszenia dla wszystkich (nauczyciel, rodzic, uczeń)
    - resetowanie haseł
    - Panel administratora (administrator techniczny)
    - wgląd do całości systemu

## 2.3 Wymagania niefunkcjonalne

### 2.3.1 Wykorzystywane technologie i narzędzia

Realizacja projektu nastąpiła jako aplikacja webowa podzielona na frontend i backend.

Backend został zrealizowany przy pomocy frameworka spring boot oraz ORMa Hibernate, napisany jest w języku programowania Java.

Frontend jest aplikacją w języku Typescript korzystającą z frameworków React oraz Chakra UI. Frontend komunikuje się z backendem przy pomocy REST API.

Baza danych korzysta z silnika PostgreSQL.

### 2.3.2 Wymagania dotyczące rozmiaru bazy danych

Po wstępnym przeliczeniu dla poniższych założeń co do najliczniejszych tabel w bazie:

- 800 uczniów
- 50 nauczycieli
- 600 rodziców
- 7200 lekcji w jednym semestrze
- 36000 ocen w jednym semestrze
- 180000 wpisów obecności w jednym semestrze
- nieokreślona liczba wiadomości

Otrzymaliśmy około 1/2 GB danych w bazie na jeden semestr.

### 2.3.3 Wymagania dotyczące bezpieczeństwa systemu

W naszej aplikacji zastosowaliśmy mechanizm uwierzytelniania użytkowników przy pomocy JWT. Wszystkie zapytania do serwera muszą być autoryzowane, a użytkownik musi być zalogowany. Wszystkie zapytania do bazy danych są sprawdzane pod kątem uprawnień użytkownika przez logikę biznesową aplikacji po stronie backendu.

## 2.4 Przyjęte założenia projektowe

# 3. Projekt systemu

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

## 3.1 Projekt bazy danych

### 3.1.1 Analiza rzeczywistości i uproszczony model konceptualny

W trakcie projektu analizując rzeczywiste działanie szkoły i biorąc pod uwagę nasze wcześniejsze doświadczenia z systemami dzienników elektronicznych, doszliśmy do wniosku, że nasz system powinien składać się z kilku głównych elementów:

- Użytkownik
- Uczeń
- Nauczyciel
- Rodzic
- Lekcja
- Ocena
- Obecność
- Wiadomość

### 3.1.2 Model logiczny i normalizacja

W trakcie projektowania bazy danych przenieśliśmy elementy z powyższego punktu na encje w bazie, oraz od razu projektowaliśmy je zgodnie z 3 postacią normalną.

### 3.1.3 Model fizyczny i ograniczenia integralności danych

Aby zapewnić poprawność danych w bazie, zdefiniowaliśmy szereg ograniczeń integralności danych:

- Klucze główne
- Klucze obce
- Unikalność
- Poprawnie dobrane typy danych
- Automatyczne akcje przy usunięciu rekordów powiązanych kluczem obcym

### 3.1.4 Inne elementy schematu - mechanizmy przetwarzania danych

Większość mechanizmów przetwarzania danych dokonywana będzie przez logikę aplikacji, z racji na używanie ORMa Hibernate na bazie, której struktura jest z góry określona.

### 3.1.5 Projekt mechanizmów bezpieczeństwa na poziomie bazy danych

W bazie danych są dwa konta dla użytkowników:

- Konto aplikacji mające dostęp do wszystkich tabel i operacji CRUD na nich, jednak nie mogące edytować ich struktury
- Konto administratora mające dostęp do wszystkich operacji na bazie danych, włącznie z edycją struktury oraz tworzenie i przywracanie backupów

## 3.2 Projekt aplikacji użytkownika

### 3.2.1 Architektura aplikacji

- W projekcie została wykorzystana reguła architektura REST. Ma na celu odseparowanie interfejsu użytkownika od operacji na serwerze. Dodatkowo zachowujemy bezstanowość poprzez wymaganie uwierzytelnienia użytkownika przy każdym zapytaniu. Kolejnym elementem wykorzystanym w projekcie jest separacja warstw. Odziela się logikę biznesową oraz prezentacji od warstwy dostępu do danych.

### 3.2.2. Interfejs graficzny i struktura UI

Środowisko graficzne napisane zostało w technologi React w połączeniu z biblioteką Chakra-UI. Interfejs użytkownika został zaimplementowany w minimalnym stopniu, zaimplementowany został system logowania oraz podstawowa funkcjonalność listowania i edycji użytkowników w systemie

### 3.2.3 Projekt wybranych funkcji systemu

### 3.2.4 Metoda podłączania do bazy danych

- W projekcie używamy **ORM** (*Hibernate*) przy założeniu **Database First**.

### 3.2.5 Projekt zabezpieczeń na poziomie aplikacji

Zabezpieczenia zrealizowane zostały za pomocą tokenów JWT dołączanych do każdego zapytania do systemu w nagłówku `authorization`

# 4. Implementacja systemu baz danych

Baza danych została zaimplementowana w PostgreSQL. W projekcie wykorzystaliśmy ORM Hibernate, który pozwala na mapowanie obiektów na relacyjne bazy danych.

## 4.1 Tworzenie tabel i definiowanie ograniczeń

### a. Tabele i Ich Przeznaczenie

#### 1. Users (Użytkownicy)

- **Przeznaczenie:** Przechowuje informacje o wszystkich użytkownikach systemu, takich jak uczniowie, rodzice, nauczyciele, dyrektorzy, administratorzy danych i techniczni.
- **Kluczowe Atrybuty:**
  - `IdUser` (PK): Unikalny identyfikator użytkownika.
  - `Login`: Nazwa użytkownika do logowania.

- `HashedPassword`: Hasło użytkownika w postaci zaszyfrowanej.
- `Email`: Adres e-mail użytkownika (opcjonalny).
- `Role`: Rola użytkownika (ENUM('student','parent','teacher','headmaster','admin') domyślnie 'student').
- `FirstName`, `SecondName`, `Surname`: Imię, drugie imię i nazwisko użytkownika.
- `IdAddress` (FK): Odniesienie do adresu w tabeli `Adresses`.

**2. PostCodes (Kody Pocztowe)**

- **Przeznaczenie:** Zawiera kody pocztowe wraz z nazwami regionów, takich jak miasta, przedmieścia czy wsie.
- **Kluczowe Atrybuty:**
  - `IdPostCode` (PK): Unikalny identyfikator kodu pocztowego.
  - `PostCode` : Kod pocztowy.
  - `RegionName`: Nazwa regionu związana z kodem pocztowym.

**3. Adresses (Adresy)**

- **Przeznaczenie:** Przechowuje szczegółowe informacje adresowe użytkowników.
- **Kluczowe Atrybuty:**
  - `IdAddress` (PK): Unikalny identyfikator adresu.
  - `IdPostCode` (FK): Odniesienie do kodu pocztowego w tabeli `PostCodes`.
  - `Street` (opcjonalne): Ulica.
  - `BuildingNumber`: Numer budynku.
  - `AdditionalBuildingIdentifier` (opcjonalny): Dodatkowy identyfikator budynku (np. litera w numerze budynku).
  - `AppartmentNumber` (opcjonalny): Numer mieszkania.

**4. Classes (Klasy)**

- **Przeznaczenie:** Przechowuje informacje o klasach szkolnych.
- **Kluczowe Atrybuty:**
  - `IdClass` (PK): Unikalny identyfikator klasy.
  - `YearOfStudy`: Rok nauki.
  - `YearStarted`: Rok rozpoczęcia nauki w klasie.

**5. Students (Uczniowie)**

- **Przeznaczenie:** Specjalizacja tabeli `Users` dla uczniów, zawierająca dodatkowe informacje.
- **Kluczowe Atrybuty:**
  - `IdStudent` (PK): Unikalny identyfikator ucznia.
  - `IdUser` (FK): Odniesienie do użytkownika w tabeli `Users`.
  - `IdClass` (FK): Odniesienie do klasy w tabeli `Classes`.
  - `DateOfBirth`: Data urodzenia ucznia.
  - `Age`: Wiek ucznia.
  - `Gender`: Płeć ucznia (`ENUM('male', 'female', 'other')`).

**6. Teachers (Nauczyciele)**

- **Przeznaczenie:** Specjalizacja tabeli `Users` dla nauczycieli, zawierająca dodatkowe informacje.
- **Kluczowe Atrybuty:**
  - `IdTeacher` (PK): Unikalny identyfikator nauczyciela.
  - `IdUser` (FK): Odniesienie do użytkownika w tabeli `Users`.
  - `LastCOCRDate`: Data ostatniego zaświadczenia o niekaralności.
  - `OnLeave`: Flaga wskazująca, czy nauczyciel jest na urlopie.

**7. SchoolSubjects (Przedmioty Szkolne)**

- **Przeznaczenie:** Przechowuje informacje o przedmiotach nauczanych w szkole.
- **Kluczowe Atrybuty:**
  - `IdSchoolSubject` (PK): Unikalny identyfikator przedmiotu.
  - `Name`: Nazwa przedmiotu.
  - `SchoolYear`: Rok szkolny, w którym przedmiot jest nauczany.
  - `LinkToSubjectProgram` (opcjonalny): Link do programu przedmiotu.

**8. Lessons (Lekcje)**

- **Przeznaczenie:** Przechowuje informacje o lekcjach.
- **Kluczowe Atrybuty:**
  - `IdLesson` (PK): Unikalny identyfikator lekcji.
  - `IdSchoolSubject` (FK): Odniesienie do przedmiotu w tabeli `SchoolSubjects`.
  - `IdTeacher` (FK): Odniesienie do nauczyciela w tabeli `Teachers`.
  - `IdClass` (FK): Odniesienie do klasy w tabeli `Classes`.
  - `Topic` (opcjonalny): Temat lekcji.
  - `Date`: Data i godzina lekcji.

**9. TeachersTeachingSubjects (Nauczyciele Nauczający Przedmioty)**

- **Przeznaczenie:** Łączy nauczycieli z przedmiotami, które nauczają.
- **Kluczowe Atrybuty:**
  - `IdTeacher` (FK): Odniesienie do nauczyciela w tabeli `Teachers`.
  - `IdSchoolSubject` (FK): Odniesienie do przedmiotu w tabeli `SchoolSubjects`.
- **Klucz Główny:** `IdTeacher`, `IdSchoolSubject`

**10. Grades (Oceny)**

- **Przeznaczenie:** Przechowuje oceny uczniów.
- **Kluczowe Atrybuty:**
  - `IdGrade` (PK): Unikalny identyfikator oceny.
  - `IdStudent` (FK): Odniesienie do ucznia w tabeli `Students`.
  - `IdTeacher` (FK): Odniesienie do nauczyciela w tabeli `Teachers`.
  - `IdSubject` (FK): Odniesienie do przedmiotu w tabeli `SchoolSubjects`.
  - `NumberGrade`: Numer oceny (domyślnie 1).
  - `Description`: Opis oceny.
  - `Date`: Data wystawienia oceny.
  - `Weight`: Waga oceny (np. 2 dla oceny z testu, 1 dla oceny z pracy klasowej).

### 11. Parents (Rodzice)

- **Przeznaczenie:** Specjalizacja tabeli `Users` dla rodziców.
- **Kluczowe Atrybuty:**
  - `IdParent` (PK): Unikalny identyfikator rodzica.
  - `IdUser` (FK): Odniesienie do użytkownika w tabeli `Users`.
  - `Role`: Rola rodzica (`ENUM('father', 'mother', 'parent', 'legal_guardian')`).

### 12. ParentsStudentPairs (Powiązania Rodziców ze Studentami)

- **Przeznaczenie:** Łączy rodziców ze studentami, umożliwiając przypisanie wielu rodziców do jednego ucznia i odwrotnie.
- **Kluczowe Atrybuty:**
  - `IdStudent` (FK): Odniesienie do ucznia w tabeli `Students`.
  - `IdParent` (FK): Odniesienie do rodzica w tabeli `Parents`.

### 13. Messages (Wiadomości)

- **Przeznaczenie:** Przechowuje wiadomości wysyłane między użytkownikami.
- **Kluczowe Atrybuty:**
  - `IdMessage` (PK): Unikalny identyfikator wiadomości.
  - `IdUserSender` (FK): Odniesienie do użytkownika wysyłającego wiadomość.
  - `Content`: Treść wiadomości.
  - `DateSent`: Data wysłania wiadomości.

### 14. MessageUserReceiverPairs (Powiązania Wiadomości z Odbiorcami)

- **Przeznaczenie:** Łączy wiadomości z ich odbiorcami.
- **Kluczowe Atrybuty:**
  - `IdMessage` (FK): Odniesienie do wiadomości w tabeli `Messages`.
  - `IdUserReceiver` (FK): Odniesienie do odbiorcy w tabeli `Users`.

### 15. Presences (Frekwencja)

- **Przeznaczenie:** Przechowuje informacje o obecności uczniów na lekcjach.
- **Kluczowe Atrybuty:**
  - `IdPresence` (PK): Unikalny identyfikator frekwencji.
  - `IdLesson` (FK): Odniesienie do lekcji w tabeli `Lessons`.
  - `IdStudent` (FK): Odniesienie do ucznia w tabeli `Students`.
  - `Type`: Typ obecności (`ENUM('present', 'excused_absence', 'unexcused_absence')`).

### 16. PhoneNumbers (Numery Telefonów)

- **Przeznaczenie:** Przechowuje numery telefonów użytkowników.
- **Kluczowe Atrybuty:**
  - `IdPhoneNumber` (PK): Unikalny identyfikator numeru telefonu.
  - `PhoneNumber`: Numer telefonu.

#### 17. PhoneNumbersParentsMatch (Powiązania Numerów Telefonów z Rodzicami)

- **Przeznaczenie:** Łączy numery telefonów z rodzicami, umożliwiając dodanie opisu do każdego połączenia.
- **Kluczowe Atrybuty:**
  - `IdPhoneNumber` (FK): Odniesienie do numeru telefonu w tabeli `PhoneNumbers`.
  - `IdParent` (FK): Odniesienie do rodzica w tabeli `Parents`.
  - `Description`: Opis powiązania (np. "telefon domowy", "telefon komórkowy").

### b. Relacje Między Tabelami

#### 1. Relacje Jeden-do-Wielu (1:N)

- **Users → Adresses:**
  - Jeden adres (`Adresses`) może być przypisany do wielu użytkowników (`Users`).
  - **Przykład:** Adres `IdAddress = 1` może być przypisany do użytkowników `IdUser = 1, 2, 3`.
- **Classes → Students:**
  - Jedna klasa (`Classes`) może mieć wielu uczniów (`Students`).
  - **Przykład:** Klasa `IdClass = 1` może mieć uczniów `IdStudent = 1, 2, 3`.
- **SchoolSubjects → Lessons:**
  - Jeden przedmiot (`SchoolSubjects`) może mieć wiele lekcji (`Lessons`).
  - **Przykład:** Przedmiot `IdSchoolSubject = 1` może mieć lekcje `IdLesson = 1, 2, 3`.
- **Teachers → Lessons:**
  - Jeden nauczyciel (`Teachers`) może prowadzić wiele lekcji (`Lessons`).
  - **Przykład:** Nauczyciel `IdTeacher = 1` może prowadzić lekcje `IdLesson = 1, 2`.

#### 2. Relacje Wielu-do-Wielu (M:N)

- **TeachersTeachingSubjects (Nauczyciele Nauczający Przedmioty):**
  - Jeden nauczyciel może uczyć wielu przedmiotów, a jeden przedmiot może być nauczany przez wielu nauczycieli.
  - **Przykład:** Nauczyciel `IdTeacher = 1` może uczyć przedmiotów `IdSchoolSubject = 1, 2`, a przedmiot `IdSchoolSubject = 1` może być nauczany przez nauczycieli `IdTeacher = 1, 2`.
- **ParentsStudentPairs (Rodzice → Uczniowie):**
  - Jeden rodzic może mieć wielu uczniów, a jeden uczeń może mieć wielu rodziców.
  - **Przykład:** Rodzic `IdParent = 1` może mieć uczniów `IdStudent = 1, 2`, a uczeń `IdStudent = 1` może mieć rodziców `IdParent = 1, 2`.

#### 3. Relacja Jeden-do-Zerowego lub Wielu (1:0..N)

- **Presences → Lessons:**

- Lekcja (`Lessons`) może mieć wiele wpisów obecności (`Presences`), ale na początku roku szkolnego mogą nie istnieć żadne wpisy.
- **Przykład:** Lekcja `IdLesson = 1` może mieć obecności `IdPresence = 1, 2, 3`, ale lekcja `IdLesson = 2` może nie mieć żadnych wpisów obecności.

- **Grades → Students:**
  - Uczeń (`Students`) może mieć wiele ocen (`Grades`), ale na początku roku szkolnego może nie mieć żadnych ocen.
  - **Przykład:** Uczeń `IdStudent = 1` może mieć oceny `IdGrade = 1, 2, 3`, ale uczeń `IdStudent = 2` może nie mieć żadnych ocen.

## c. Klucze Obce ograniczające tabele i Spójność Danych

### 1. Przykład Kluczy Obcych Zapewniających Spójność

- **Tabela `Students`:**
  - `IdUser` jest kluczem obcym odnoszącym się do `Users(IdUser)`.
  - `IdClass` jest kluczem obcym odnoszącym się do `Classes(IdClass)`.
  - **Jak to działa:** Zapewnia, że każdy uczeń jest powiązany z istniejącym użytkownikiem i klasą. Nie można dodać ucznia bez istniejącego użytkownika lub klasy, co zapobiega niespójnym danym.

- **Tabela `Presences`:**
  - `IdLesson` jest kluczem obcym odnoszącym się do `Lessons(IdLesson)`.
  - `IdStudent` jest kluczem obcym odnoszącym się do `Students(IdStudent)`.
  - **Jak to działa:** Zapewnia, że każda frekwencja odnosi się do istniejącej lekcji i istniejącego ucznia. Nie można dodać wpisu frekwencji dla nieistniejącej lekcji lub ucznia.

- **Tabela `Grades`:**
  - `IdSubject` jest kluczem obcym odnoszącym się do `SchoolSubjects(IdSchoolSubject)`.
  - `IdStudent` jest kluczem obcym odnoszącym się do `Students(IdStudent)`.
  - `IdTeacher` jest kluczem obcym odnoszącym się do `Teachers(IdTeacher)`.
  - **Jak to działa:** Każda ocena musi być przypisana do istniejącego przedmiotu, ucznia i nauczyciela. Nie można wprowadzić oceny dla przedmiotu, ucznia lub nauczyciela, który nie istnieje w odpowiednich tabelach.

## d. Indeksy Unikalne (`UNIQUE INDEX`) jako ograniczenia

- **Definicja:** Zapewniają, że wartości w określonych kolumnach są unikalne, zapobiegając duplikatom.

- **Przykład:**
  - `Login_UNIQUE` w tabeli `Users` gwarantuje, że każdy login jest unikalny.
  - `Email_UNIQUE` w tabeli `Users` gwarantuje, że każdy adres e-mail jest unikalny.
  - `IdUser_UNIQUE` w tabeli `Students` zapewnia, że dany użytkownik może być przypisany tylko do jednego ucznia.
  - `IdUser_UNIQUE` w tabeli `Teachers` zapewnia, że dany użytkownik może być przypisany tylko do jednego nauczyciela.

- `IdUser_UNIQUE` w tabeli `Parents` zapewnia, że dany użytkownik może być przypisany tylko do jednego rodzica.
- `PhoneNumber_UNIQUE` w tabeli `PhoneNumbers` gwarantuje, że każdy numer telefonu jest unikalny.

## 4.2 Implementowanie machanizmów przetwarzania danych

Jedynymi mechanizmami przetwarzającymi dane bezpośrednio w bazie danych są widoki zdefiniowane i opisane poniżej:

`StudentGradesView` **– Widok Ocen Ucznia**

Pozwala uczniowi zobaczyć swoje oceny z informacjami o przedmiocie, nauczycielu i wadze.

**Definicja widoku:**

```
CREATE VIEW StudentGradesView AS
SELECT
    g.IdGrade,
    g.IdStudent,
    ss.Name AS SubjectName,
    g.NumberGrade,
    g.Weight,
    g.Description,
    g.Date,
    CONCAT(tu.FirstName, ' ', tu.Surname) AS TeacherName
FROM
    Grades g
    INNER JOIN Students s ON g.IdStudent = s.IdStudent
    INNER JOIN Teachers t ON g.IdTeacher = t.IdTeacher
    INNER JOIN Users tu ON t.IdUser = tu.IdUser
    INNER JOIN SchoolSubjects ss ON g.IdSubject = ss.IdSchoolSubject;
```

**Użycie:**

Uczeń wyświetla swoje oceny:

```
SELECT *
FROM StudentGradesView
WHERE IdStudent = [TwojeIdStudenta];
```

---

`UserMessagesView` **– Widok Wiadomości Użytkownika**

Pozwala użytkownikowi odczytać swoje wiadomości wraz z informacjami o nadawcy.

**Definicja widoku:**

```
CREATE VIEW UserMessagesView AS
SELECT
    m.IdMessage,
    murp.IdUserReceiver,
    m.Content,
    m.DateSent,
    CONCAT(su.FirstName, ' ', su.Surname) AS SenderName
FROM
    Messages m
    INNER JOIN MessageUserReceiverPairs murp ON m.IdMessage = murp.IdMessage
    INNER JOIN Users su ON m.IdUserSender = su.IdUser;
```

**Użycie:**

Użytkownik wyświetla swoje wiadomości:

```
SELECT *
FROM UserMessagesView
WHERE IdUserReceiver = [TwojeIdUser];
```

---

`ChildPresencesView` **– Widok Obecności Dziecka**

Pozwala rodzicowi zobaczyć obecności swojego dziecka na lekcjach.

**Definicja widoku:**

```
CREATE VIEW ChildPresencesView AS
SELECT
    p.IdPresence,
    p.IdStudent,
    p.Type,
    l.DateTimeStart,
    l.Topic,
    ss.Name AS SubjectName,
    CONCAT(su.FirstName, ' ', su.Surname) AS StudentName
FROM
    Presences p
    INNER JOIN Lessons l ON p.IdLesson = l.IdLesson
    INNER JOIN SchoolSubjects ss ON l.IdSchoolSubject = ss.IdSchoolSubject
    INNER JOIN Students s ON p.IdStudent = s.IdStudent
    INNER JOIN Users su ON s.IdUser = su.IdUser
    INNER JOIN ParentsStudentPairs psp ON s.IdStudent = psp.IdStudent
WHERE
    psp.IdParent = [TwojeIdParent];
```

**Użycie:**

Rodzic wyświetla obecności dziecka:

```
SELECT *
FROM ChildPresencesView;
```

## 4.3 Implementacja uprawnień i innych zabezpieczeń

W realizacji bazy danych planowane są dwa rodzaje uprawnień:

- dla administratora technicznego (administratora bazy danych) - pełne uprawnienia do zarządzania bazą danych,
- dla serwera aplikacji - typowe uprawnienia CRUD dla bazy, z wyłączeniem edycji struktury tabel.

Powyższe uprawnienia będą nadawane na poziomie użytkownika bazy danych, a nie na poziomie tabeli.

Uważamy że powyższe uprawnienia będą wystarczające dla zapewnienia bezpieczeństwa bazy, ponieważ:

- dostęp do bazy będzie ograniczony tylko do podanych adresów IP, administratora i serwera aplikacji, oraz serwer dbms będzie zabezpieczony odpowienimi zasadami firewalla który będzie blokował cały ruch z poza dwóch podanych adresów IP do bazy.

## 4.4 Testowanie bazy danych na przykładowych danych

# 5. Implementacja i testy aplikacji

- Skrócone sprawozdanie z etapu implementacja i testowania aplikacji.

## 5.1 Instalacja i konfigurowanie systemu

W celu wprowadzenia własnej implementacji należy edytować plik `application.yml` dostępny w podfolderze `resources` w module `client-app`
Szczególną uwagę należy zwrócić na port na którym działa aplikacja oraz parametry połączenia do bazy danych

```
spring:
  datasource:
    url: jdbc:postgresql://<DB_URL>
    username: <DB_USER>
    password: <DB_PASSWORD>
```

Następnym krokiem jest zainstalowanie wymaganych bibliotek i wygenerowanie klas pomocniczych. Możemy to osiągnąć za pomocą polecenia

```
mvn clean install
```

W celu uruchomienia aplikacji należy wykonać polecenie

```
mvn -pl client-app -am spring-boot:run
```

---

## 5.2 Instrukcja użytkowania aplikacji

Aplikacja została zbudowana w oparciu o model architektury REST. Udostępnione zostały kontrolery umożliwiające listowanie oraz edycję danych.

Pełna dokumentacja dostępnych kontrolerów wraz z modelami danych została przygotowana za pomocą narzędzia `Swagger 2 UI`, które jest dostępne pod adresem: `<APP_URL>/swagger-ui/index.html`.

Backend aplikacji jest zabezpieczony mechanizmem autoryzacji opartym na tokenach JWT. Każdy z kontrolerów wymaga dołączenia ważnego tokenu autoryzacyjnego do nagłówka Authorization. Token ten można uzyskać, wysyłając żądanie POST na endpoint <APP_URL>/auth/login z następującymi danymi:

```
{
  "username": "twoja_nazwa_uzytkownika",
  "password": "twoje_haslo"
}
```

Jeśli proces autoryzacji przebiegnie pomyślnie, endpoint zwróci dane w następującym formacie:

```
{
  "token": "twój_token_jwt",
  "expireTime": 1234567890
}
```

Token uzyskany w odpowiedzi należy dołączyć w nagłówku Authorization w postaci Bearer w każdym dalszym żądaniu do kontrolerów zabezpieczonych autoryzacją.

---

## 5.3 Testowanie opracowanych funkcji systemu

Dla wybranych funkcji systemu opracowane zostały testy jednostkowe sprawdzające poprawne działanie poszczególnych komponentów.
Testy opracowane zostały za pomocą narzędzi i bibliotek dostarczanych przez środowisko `spring boot` takich jak `mockito` oraz `JUnit`
Testom poddane zostały fasady, serwisy i mappery. Poniżej znajduje się przykładowy test mappera upewniający się, że pola i klasy zagnieżdżone zostają poprawnie zmapowane

```
@BeforeEach
public void setUp() {
  schoolClass = new SchoolClass();
  schoolClass.setId(SCHOOL_CLASS_ID);

  when(schoolClassMapper.createDestination(any(SchoolClass.class))).thenReturn(new
SchoolClassDto());

  doAnswer(invocation -> {
  var entity = invocation.getArgument(0, SchoolClass.class);
  var dto = invocation.getArgument(1, SchoolClassDto.class);
  dto.setId(entity.getId());
  return null;
  }).when(schoolClassMapper).transform(any(SchoolClass.class),
any(SchoolClassDto.class), any(SchoolClassDto.Properties.class));
}

@Test
public void givenLesson_shouldMapToLessonDto() {
  var lessonId = UUID.randomUUID();
  var lesson = new Lesson();
  lesson.setId(lessonId);
  lesson.setSchoolSubject(schoolSubject);

  var properties = mappingService.createProperties(LessonDto.Properties.class)
    .setIncludeSchoolSubject(true);

  var lessonDto = lessonMapper.map(lesson, properties);

  assertEquals(lessonDto.getId(), lessonId);

  assertThat(lessonDto.getStudentClass()).isNotNull();
  assertEquals(lessonDto.getStudentClass().getId(), SCHOOL_CLASS_ID);
}
```

## 5.4 Omówienie wybranych rozwiązań programistycznych

### 5.4.1 Implementacja interfejsu dostępu do bazy danych

W projekcie wykorzystano w pełni możliwości oferowane przez środowisko **Spring Boot**, w tym funkcjonalności dostarczane przez **Spring Data JPA**. Dzięki temu implementacja interfejsu dostępu do bazy danych została znacząco uproszczona i zautomatyzowana. Spring Data JPA generuje niezbędny kod odpowiedzialny za komunikację z bazą danych na podstawie definicji encji i repozytoriów, co pozwala skupić się na logice biznesowej aplikacji. \

**Konfiguracja połączenia z bazą danych**
Połączenie z bazą danych zostało skonfigurowane za pomocą pliku konfiguracyjnego `application.yml`. W pliku tym określono kluczowe parametry, takie jak:

- adres bazy danych(`spring.datasource.url`)
- dane uwierzytelniające użytkownika (`spring.datasource.username` i `spring.datasource.password`)
- sposób zarządzania schematem bazy danych (`spring.jpa.hibernate.ddl-auto`)
- mechanizm ORM (Hibernate) oraz jego parametry

**Definicja encji**
Każda tabela w bazie danych została odwzorowana za pomocą klasy encji oznaczonej odpowiednimi adnotacjami JPA, m.in.:

- `@Entity` – określa, że dana klasa jest encją,
- `@Table` – umożliwia dostosowanie nazwy tabeli w bazie danych (opcjonalne),
- `@Id` oraz `@GeneratedValue` – definiują klucz główny i strategię jego generowania.

**Repozytoria**
Dostęp do danych został zrealizowany poprzez repozytoria dziedziczące po interfejsie `JpaRepository`. Dzięki tej implementacji możliwe jest wykorzystanie gotowych metod CRUD (m.in. `save()`, `findById()`, `deleteById()`) bez konieczności pisania dodatkowego kodu.

### 5.4.2 Implementacja wybranych funkcjonalności systemu

**Obsługa wyjątków i błędów** W celu ułatwienie konkretnych błędów zgłaszanych przez aplikację stworzony został specjalny wyjątek obsługujący własne błędy

```
@Getter
public class ApplicationException extends RuntimeException {

    private final ErrorDescriptor descriptor;
    private final HttpStatus status;
    private final String code;
    private final Map<String, Object> additionalInfo;
    private final BindingResult bindingResult;

    // custom constructors...
}
```

Wyjątek ten jest następnie wyłapywany i analizowany przez klasę `ExceptionMapper` która na podstawie argumentów wyjątku zwraca szczegóły błedu poprzez odpowiednią odpowiedź w formie `ErrorDto`

```
@ControllerAdvice
public class ExceptionMapper extends ResponseEntityExceptionHandler implements
InitializingBean {

    // ...
public ResponseEntity<Object> handleAnyException(Exception exception, WebRequest
request, Function<ErrorDto, Object> errorWrapper) {
    ResolvedError resolvedError = resolveError(exception);
    ErrorDto errorDto = resolvedError.getError();
    HttpStatus status = resolvedError.getStatus();
```

```
    Object body = errorWrapper != null ? errorWrapper.apply(errorDto) : errorDto;
    HttpHeaders headers = new HttpHeaders();
    headers.add("Content-Entity", "Error");
    return handleExceptionInternal(exception, body, headers, status, request);
  }
  // ...
}
```

Klasa `ErrorDto`

```
@Data
@Accessors(chain = true)
public class ErrorDto {

    private String requestId;

    private String code;

    private String message;

    private Map<String, Object> additionalInfo;

    private boolean accessRevoked;

    private FieldErrorDto[] fieldErrors;
}
```

**CRUD dla zasobów (na przykładzie kontrolera `LessonController`**

Kontroler umożliwia listowanie, tworzenie i edycję danych

```
@RestController
@RequestMapping("/lesson")
@RequiredArgsConstructor
public class LessonController implements InitializingBean {
    private final LessonFacade lessonFacade;

    @GetMapping
    public ResponseEntity<Collection<LessonDto>> getAll(LessonFilter filter) {
        return ResponseEntity.ok(lessonFacade.getList(filter,
defaultListProperties));
    }

    @GetMapping("/{id}")
    public ResponseEntity<LessonDto> getById(@PathVariable UUID id) {
        return ResponseEntity.ok(lessonFacade.getById(id,
defaultSingleProperties));
    }

    @PostMapping
```

```
    public ResponseEntity<LessonDto> create(@RequestBody LessonDto dto) {
        var lesson = lessonFacade.create(dto);
        return ResponseEntity.ok(lessonFacade.map(lesson,
defaultSingleProperties));
    }

    @PutMapping("/{id}") // zmienna oznaczona @PathVariable musi mieć dokładnie
taką samą nazwę jak w stringu w @PutMapping
    public ResponseEntity<LessonDto> update(@PathVariable UUID id, @RequestBody
LessonDto dto) {
        var lesson = lessonFacade.update(id, dto);
        return ResponseEntity.ok(lessonFacade.map(lesson,
defaultSingleProperties));
    }
}
```

**Properties**

W kontrolerach wykorzystywany jest autorski mechanizm `MappingProperties` które sterują zagnieżdżanym mapowaniem encji w maperach. Tworzone są za pomocą serwisu `MappingService`, w każdym kontrolerze w dostarczonej przez Spring Boot metodzie `afterPropertiesSet()`. Serwis ten używa refleksji więc w celu braku wpływy na czas odpowiedzi, wszystkie metody wykonywane są jednorazowo tuż po starcie aplikacji.

```
@RestController
@RequestMapping("/lesson")
@RequiredArgsConstructor
public class LessonController implements InitializingBean {
    private final MappingService mappingService;

    private MappingProperties defaultSingleProperties;
    private MappingProperties defaultListProperties;

    @Override
    public void afterPropertiesSet() throws Exception {
        defaultSingleProperties =
mappingService.createProperties(LessonDto.Properties.class)
                .setIncludeSchoolSubject(true)
                .setIncludeTeacher(true)
                .setIncludeStudentClass(true);
        defaultListProperties =
mappingService.createProperties(LessonDto.Properties.class)
                .setIncludeSchoolSubject(true)
                .setIncludeTeacher(true)
                .setIncludeStudentClass(true);
    }
    // metody kontrolera...
}
```

**Logowanie i autoryzacja**

System autoryzacji oparty jest o tokeny JWT, uzyskiwany poprzez odpytanie endpointu `/auth/login` z

zawartymi parametrami `email` oraz `password`. Po sprawdzeniu poprawności danych system zwraca wygenerowany token i czas jego żywotności.

```
@Transactional
    public JwtDto authenticate(CredentialsDto dto) {
        try {
            var token = new UsernamePasswordAuthenticationToken(dto.getEmail(),
dto.getPassword());
            var auth = (UsernamePasswordAuthenticationToken)
authenticationManager.authenticate(token);
            SecurityContextHolder.getContext().setAuthentication(auth);

            var accessToken = jwtService.generateToken((AuthenticatedUser)
auth.getPrincipal());
            var jwtDto = new JwtDto();
            jwtDto.setToken(accessToken);
            jwtDto.setExpiresIn(jwtService.getExpirationTime());
            return jwtDto;
        } catch (AuthenticationException ex) {
            throw new IllegalStateException(ex);
        }
    }

@Data
public class CredentialsDto {

  private String email;

  private String password;
}

@Data
public class JwtDto {

  private String token;

  private Long expiresIn;
}
```

Każde zapytanie do aplikacji weryfikowane jest przez filtr `JwtAuthorizationFilter` sprawdzający poprawność tokenu. W razie jego niepoprawności system zwraca kod błędu 403 (FORBIDDEN).

```
@Override
    protected void doFilterInternal(@NonNull HttpServletRequest request, @NonNull
HttpServletResponse response, @NonNull FilterChain filterChain) throws
ServletException, IOException {
        String accessToken = resolveToken(request);
        if(accessToken == null || accessToken.equals("null")) {
            filterChain.doFilter(request, response);
            return;
```

```
        }
        try {
            var userEmail = jwtService.extractUsername(accessToken);
            var authentication =
SecurityContextHolder.getContext().getAuthentication();

            if(userEmail != null && authentication == null) {
                var authenticatedUser = (AuthenticatedUser)
userService.loadUserByUsername(userEmail);
                if(jwtService.isTokenValid(accessToken, authenticatedUser)) {
                    var user = userService.findByEmail(userEmail);
                    var authorities = new HashSet<GrantedAuthority>();
                    userService.collectAuthorities(user, authorities);
                    var authToken = new
UsernamePasswordAuthenticationToken(authenticatedUser, null, authorities);
                    authToken.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(authToken);
                }
            }
            filterChain.doFilter(request, response);
        } catch (JwtException ex) {
            handlerExceptionResolver.resolveException(request, response, null, new
ApplicationException(ApplicationError.InvalidJwtToken));
        } catch (Exception ex) {
            handlerExceptionResolver.resolveException(request, response, null,
ex);
        }
    }
```

# 6. Podsumowanie i wnioski

W ramach projektu opracowano system dziennika elektronicznego, obejmujący zarówno jego projekt, jak i implementację kluczowych funkcjonalności. Centralnym elementem systemu jest baza danych, zaprojektowana zgodnie z zasadami trzeciej postaci normalnej, co gwarantuje jej przejrzystość, spójność oraz minimalizację redundancji danych. Aplikacja dostępowa została stworzona jako aplikacja webowa w architekturze REST, co zapewnia elastyczność oraz możliwość łatwej integracji z innymi systemami.

Backend systemu jest wysoce skalowalny, co umożliwia szybkie i sprawne dodawanie nowych funkcjonalności w przyszłości. Proces uwierzytelniania i autoryzacji użytkowników został zrealizowany z użyciem tokenów JWT (JSON Web Token), co zapewnia wysoki poziom bezpieczeństwa oraz zgodność z nowoczesnymi standardami.

Realizacja projektu okazała się bardziej wymagająca, niż pierwotnie zakładano. Stworzenie aplikacji dostępowej zgodnie z najlepszymi praktykami programistycznymi oraz najnowszymi trendami w tworzeniu aplikacji webowych wymagało przygotowania ponad 100 plików zawierających klasy i interfejsy w języku Java, co przełożyło się na znaczną ilość godzin pracy. Pomimo to, backend systemu oraz REST API są w pełni funkcjonalne, umożliwiając pełną obsługę operacji CRUD (Create, Read, Update, Delete) na bazie danych.

Ze względu na ograniczenia czasowe nie udało się zrealizować w pełni interfejsu użytkownika. Opracowano jedynie podstawową wersję panelu administratora, która pozwala na zarządzanie kluczowymi elementami systemu. Niemniej jednak, solidna implementacja warstwy backendowej oraz zgodność z architekturą REST daje solidne podstawy do dalszego rozwijania systemu i rozbudowy interfejsu w przyszłości.

# 7. Źródła


- https://github.com/Ite-2022-pwr
- https://github.com/Ite-2022-pwr/sem5-bd2-proj-mg-ak-mb-dp
- https://devszczepaniak.pl/wprowadzenie-do-rest-api/


# 8. Spis rysunków