Politechnika Wrocławska

Bazy danych- Projekt

Temat: Elektroniczny indeks wyższej uczelni

Termin zajęć: Piątek 11:15

Prowadzący: dr inż. Roman Ptak
Data: 19.01.2024 r.

Skład grupy:

Filip Murawski 263894,
Olaf Szajda, 263906,
Maciej Padula 263919,
Adam Czekalski 264488

Spis treści

1. Wstęp

Celem projektu jest stworzenie aplikacji bazodanowej pełniącej funkcje elektronicznego indeksu uczelni wyższej – Politechniki Wrocławskiej. Aplikacja obsługiwać będzie proces nauczania, czyli zarządzanie kursami, zapisami studentów na zajęcia w ramach kursów oraz ocenianiem. Użytkownikami systemu będą pracownicy dziekanatu, prowadzący zajęcia oraz studenci.

2. Analiza wymagań

2.1. Wymagania funkcjonalne

Pracownik administracji może

- zarządzać semestrami (tworzyć, modyfikować, przeglądać oraz usuwać)
- zarządzać wydziałami
- zarządzać kursami
- zarządzać grupami zajęciowymi
- dodawać studentów do grup zajęciowych
- wypisać administracyjnie studenta z grupy zajęciowej
- zarządzać studentami
- zarządzać prowadzącymi zajęcia
- zarządzać pracownikami administracji

Prowadzący zajęcia może

- przeglądać listę grup, które prowadzi
- przeglądać listę studentów zapisanych do jego poszczególnych grup
- dodawać oraz przeglądać oceny studentów, którzy należą do jego grup zajęciowych
- wystawiać oceny końcowe studentom z grup, które prowadzi
- modyfikować swoje dane kontaktowe
- zaznaczać nieobecności studentów na konkretnych terminach zajęć

Student może

- zapisać się na studia na wybranych kierunkach
- przeglądać listę grup, do których należy
- przeglądać listę swoich ocen
- akceptować lub reklamować nowo wprowadzone oceny
- sprawdzić swoją średnią ważoną z wybranego semestru
- modyfikować swoje dane (hasło)
- przeglądać plan zajęć na dany semestr/tydzień/dzień

2.2. Wymagania niefunkcjonalne

2.2.1. Wykorzystywane technologie i narzędzia

Wynikiem projektu będzie aplikacja webowa napisana przy pomocy ASP.NET MVC (Razor Pages). Jako bazę danych wykorzystamy Microsoft SQL Server korzystający z języka T-SQL. Aplikacja webowa wraz z infrastrukturą bazodanową zostanie upubliczniona w Microsoftowej chmurze Azure.

2.2.2. Wymagania dotyczące rozmiaru bazy danych

Opis encji systemu

- Student {numer indeksu, podstawowe dane o studencie}
- Prowadzący zajęcia {id, podstawowe dane o prowadzącym zajęcia}
- Pracownik administracji {id, podstawowe dane o pracowniku administracji}
- Wydział: {numer, nazwa}
- Kierunek {wydział, nazwa, stopień}
- Semestr {początek, koniec, typ(letni/zimowy)}
- Kurs {nazwa, opiekun, ECTS}
- Grupa zajęciowa {kurs, termin, prowadzący_grupę}
- Ocena {grupa_zajęciowa, student, ocena}
- Nieobecność {grupa_zajęciowa, data, student}

Analiza liczności encji

- Użytkownik - 21 881 studentów, 2 287 nauczycieli akademickich, 85 pracowników administracji, łącznie: 24253 [21 881 studentów + 2 287 nauczycieli + 85 pracowników]
- Wydział - 16 wydziałów
- Kierunek - 69 kierunków rozłożonych na 16stu wydziałach
- Semestr - 7 semestrów I stopień + 3 semestry II stopień
- Kurs - 10 kursów na semestr dla każdego z 69 kierunków, łącznie: 690 [10 kursów * 69 kierunków]
- Grupa zajęciowa - 7 grup zajęciowych (1 grupa wykładowa + 6 grup dla form towarzyszących np.: laboratoria, ćwiczenia, seminaria, projekty) dla każdego kursu, łącznie 4830 [690 kursów * 7 grup zajęciowych]
- Ocena – 2 oceny na każdy kurs studenta zakładając, że średnio zapisany jest on do dwóch grup zajęciowych w kursie (wykład + forma towarzysząca), łącznie: 4 376 200 [21 881 studentów * 2 oceny * 10 kursów * 10 semestrów]
- Nieobecność - 2 dopuszczone nieobecności dla każdego studenta w każdej grupie zajęciowej do której jest zapisany (zakładając ponownie dwie na kurs), łącznie: 8 752 400 [21 881 studentów * 2 nieobecności * 2 grupy * 10 kursów * 10 semestrów]

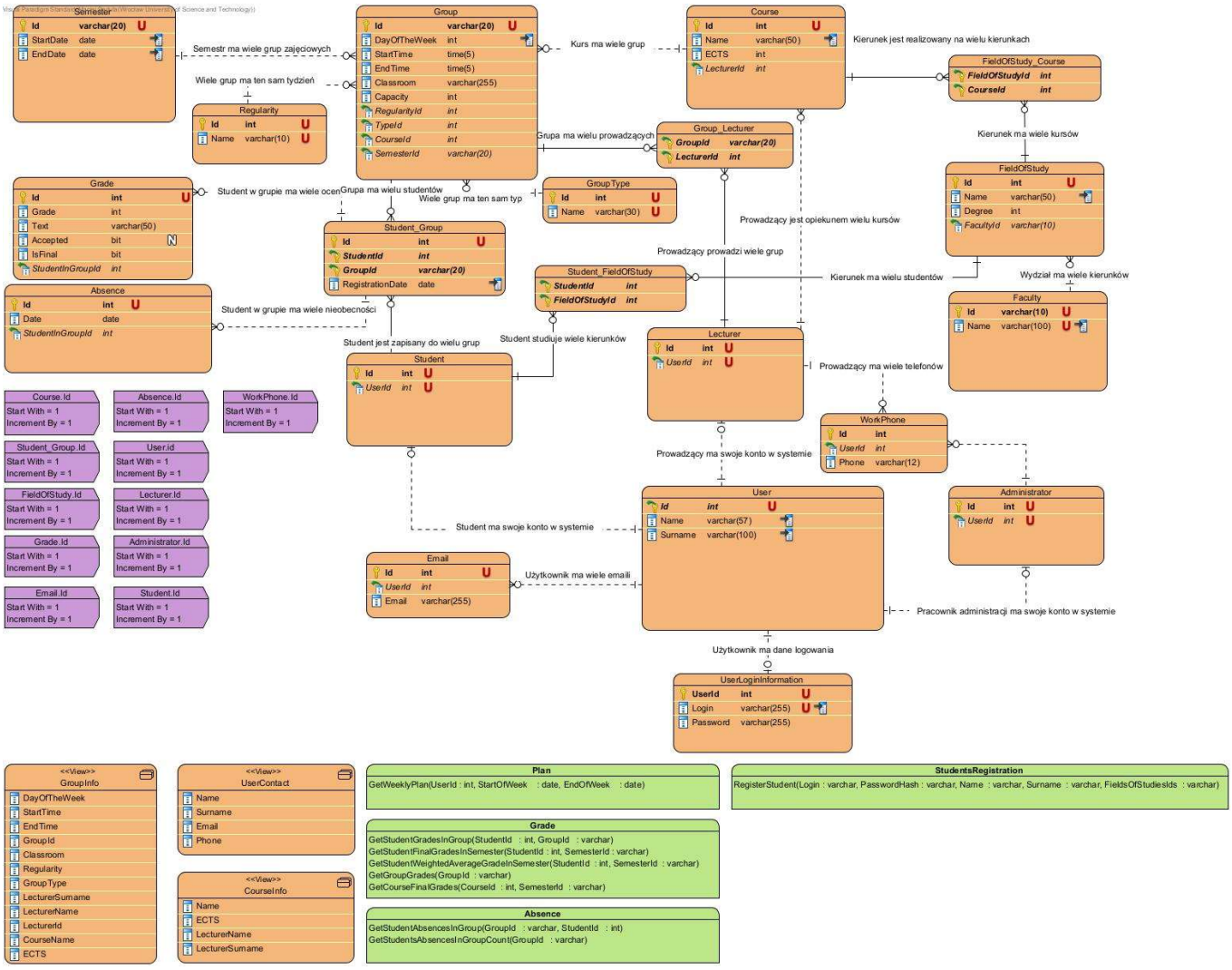
2.2.3. Wymagania dotyczące bezpieczeństwa systemu

Ze względu na wrażliwy charakter danych przetwarzanych przez aplikację oraz jej ogólną dostępność w publicznej sieci wymagane są dodatkowe środki bezpieczeństwa. Aby uniemożliwić dostęp osobom niepowołanym poza wymaganiem nazwy użytkownika oraz hasła do zalogowania będzie możliwość skonfigurowania weryfikacji dwuetapowej (np.: login/hasło + email z kodem).

3. Projekt systemu

3.1. Projekt Bazy danych

3.1.1. Model fizyczny bazy danych



Rysunek 1 Struktura bazy danych stworzona za pomocą programu Visual Paradigm

Pola imię i Nazwisko w tabeli User tworzą indeks złożony.

Daty startu i końca semestru tworzą indeks złożony, sortowanym malejąco.

Indeks registration date w tabeli student jest sortowany malejąco.

3.1.2. Modelowanie pozostałych elementów bazy danych

3.1.2.1. Zabezpieczenia

Dostęp do bazy danych zostanie ograniczony dla trzech użytkowników:

- Pracownik administracji
    - SELECT, INSERT, UPDATE, DELETE, EXEC na wszystkie obiekty w bazie
    - SELECT na GroupInfo, UserContact, CourseInfo
- Student
    - SELECT na wszystko
    - UPDATE na oceny – Accepted, User – Password

- EXEC na RegisterStudent, GetStudentWeeklyPlan, GetStudentGradesInGroup, GetStudentFinalGradesInSemester, GetStudentWeightedAverageGradeInSemester, GetStudentAbsencesInGroup
- SELECT na GroupInfo, UserContact, CourseInfo

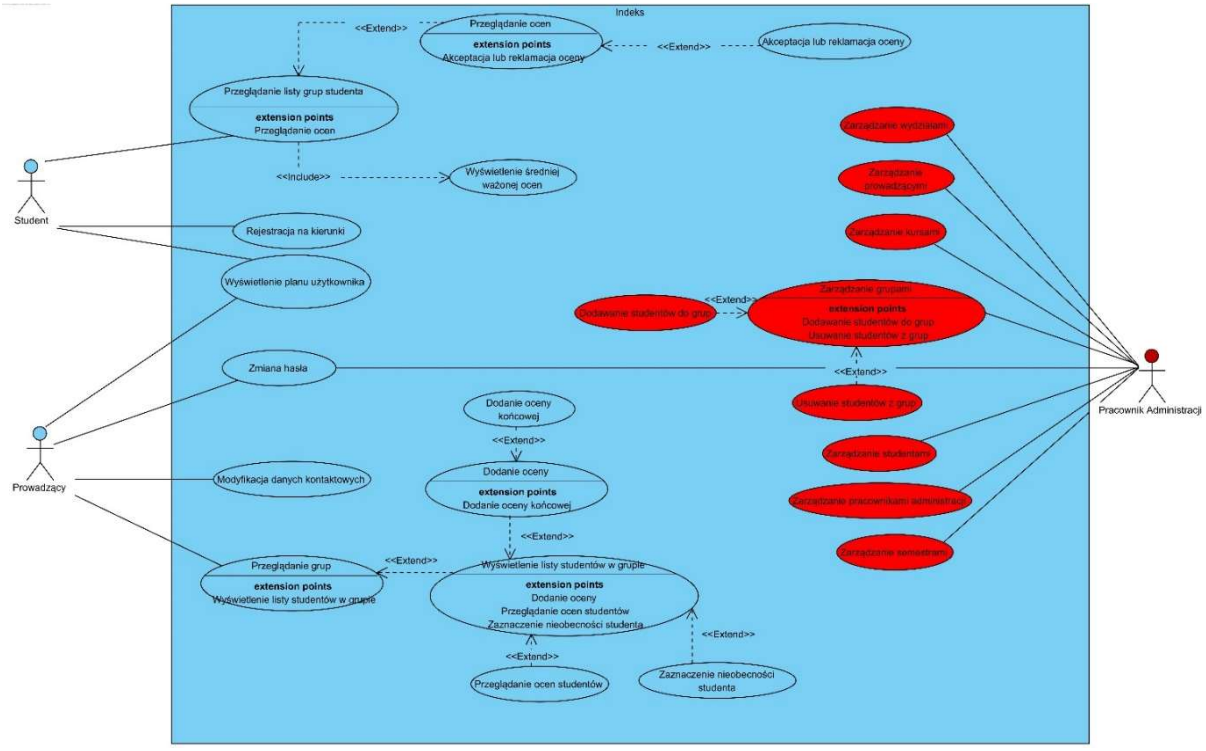
- Prowadzący
  - SELECT na wszystko
  - INSERT na Grade, Absence
  - DELETE na Absence
  - EXEC na RegisterLecturer, GetLecturerWeeklyPlan , GetStudentGradesInGroup, GetGroupGrades, GetCourseFinalGrades, GetStudentAbsencesInGroup, GetStudentsAbsencesInGroupCount
  - SELECT na GroupInfo, UserContact, CourseInfo

3.2. Projekt aplikacji użytkownika

3.2.1. Architektura aplikacji i diagramy projektowe

Struktura projektu w aplikacji będzie dziedziczyć z „modularnego monolitu”, czyli każda funkcjonalność lub zbiór mniejszych możliwości będzie osobnym projektem ze swoimi komponentami wewnętrznymi, które nie są dostępne poza nim. Pozwoli to na enkapsulację logiki i konkretnych implementacji przed resztą aplikacji. Każdy tzw. „monolit” to aplikacja ASP.NET MVC (Model-View-Controller).

Diagram przypadków użycia:



Rysunek 2 Diagram przypadków użycia stworzony za pomocą Visual Paradigm

3.2.2. Interfejs graficzny i struktura menu

Po uruchomieniu aplikacji ukazuje się strona główna. Po kliknięciu w opcję „zaloguj" pojawia formularz logowania z polami do wpisania loginu i hasła. Jest też zakładka z opcją „zarejestruj". Gdy się w nią wejdzie, pojawia się formularz rejestracji z polami „Imię", „Nazwisko", „Login", „Kierunek/Kierunki studiów", „Hasło", „Potwierdź hasło". Po poprawnym zalogowaniu się, ukazuje się strona główna. Do poszczególnych funkcjonalności można uzyskać dostęp wybierając jeden z odnośników znajdujących się w górnej belce. Możliwe opcje do wyboru: „Grupy", „Plan zajęć", „Wyszukaj prowadzącego", „Ustawienia konta", „Wyloguj się".

Zakładka „Ustawienia konta" daje dostęp do dwóch opcji: zmiana hasła oraz zarządzanie emailami.

Po wejściu na podstronę „Grupy" wyświetlają się wszystkie nazwy kursów które prowadzi prowadzący lub kursy na które aktualnie uczęszcza student. Można także zobaczyć kursy z poprzednich semestrów wybierając żądany semestr oraz dokonać filtracji kursów (wpisać w pole tekstowe nazwę poszukiwanego kursu i kliknąć przycisk). Prowadzący dla każdej grupy może wejść na podstronę „Obecność" i „Oceny". Po wejściu na podstronę „Obecność" wyświetlają się indeksy oraz imiona i nazwiska studentów z danej grupy wraz z kolumnami odpowiadającymi dacie terminu zajęć. Dla każdego studenta i terminu zajęć istnieje jedno pole, które można zaznaczyć i odznaczyć. Zaznaczenie tego pola oznacza, że student jest obecny, w przeciwnym wypadku nieobecny. Po wejściu na podstronę „Oceny" prowadzący może dodać oraz wyświetlić oceny studentów zapisanych do danej grupy. Student natomiast, po rozwinięciu kursu może zobaczyć imię i nazwisko prowadzącego kurs, ilość punktów ECTS, ocenę końcową, rodzaj zajęć (np. wykład), imię i nazwisko prowadzącego, kod grupy oraz termin zajęć (dzień tygodnia wraz z godziną rozpoczęcia i zakończenia). U studenta dodatkowo na górze wyświetla się suma ilości uzyskanych punktów ECTS w danym semestrze oraz jego średnia ważona z danego semestru.

Student może również podejrzeć swoje oceny po wejściu w wybraną grupę zajęciową.

Po wejściu na podstronę „Wyszukaj prowadzącego" wyświetla się lista imion i nazwisk wszystkich prowadzących wraz z ich e-mailami i numerami telefonów. Na górze pojawia się pole tekstowe, które umożliwia filtrację wyników po imieniu i nazwisku. Na dole można przełączać się między stronami, jeśli lista prowadzących jest wystarczająco długa.

Po wejściu na podstronę „Plan zajęć" wyświetla się plan zajęć na aktualny tydzień dla prowadzącego bądź studenta w zależności od typu zalogowanego użytkownika. Można przełączać się między tygodniami. Na planie widoczny jest dzień tygodnia oraz godzina.

Podstrona „Oceny" dostępna jest z dwóch perspektyw:
Student może przeglądać swoje oceny w danej grupie zajęciowej oraz dokonać akceptacji bądź reklamacji wystawionych ocen końcowych.
Prowadzący natomiast widzi listę ocen wszystkich studentów zapisanych do danej grupy zajęciowej, oraz może dodawać nowe oceny indywidualnym studentom.

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

Biblioteka Dapper zostanie użyta w celu pobierania i aktualizowania danych z poziomu aplikacji. Jest to lekka biblioteka typu Micro-ORM służąca do mapowania z odpowiedzi SQL na obiekty C#. Umożliwia ona na pisanie własnych zapytań co umożliwia na optymalizowanie zapytań. Przykład zapytania w Dapperze:

```
    public async Task<Dictionary<int, decimal>> GetStudentCoursesGrades(
      int studentId, string semesterId)
    {
        using var connection = await _dbConnectionFactory
                    .GetOpenStudentConnectionAsync();

        var query = @$"
SELECT
    c.Id AS [{nameof(CourseGrade.CourseId)}],
    gr.Grade AS [{nameof(CourseGrade.Value)}]
FROM [dbo].[Course] c
INNER JOIN [dbo].[Group] g ON g.CourseId = c.Id
INNER JOIN [dbo].[Student_Group] sg ON sg.GroupId = g.Id
INNER JOIN [dbo].[Grade] gr ON gr.StudentInGroupId = sg.Id
WHERE gr.IsFinal = 1
        AND g.TypeId = {(int)GroupType.Lecture}
        AND sg.StudentId = @StudentId
        AND g.SemesterId LIKE @SemesterId
";

        var queryResult = await connection
             .QueryAsync<CourseGrade>(query, new { studentId, semesterId });

        return queryResult
            .ToDictionary(x => x.CourseId, x => x.Value);
    }
```

3.2.4. Projekt zabezpieczeń na poziomie aplikacji

Podstawowym zabezpieczeniem aplikacji jest protokół https, który szyfruje ruch z i do aplikacji. Poza tym autoryzacja użytkownika odbywa się na podstawie tzw. Json Web Tokena, który trzymany jest w ciastku:

```
public string GenerateJSONWebToken(User user)
{
    var securityKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["Jwt:Key"]));
    var credentials = new SigningCredentials(securityKey,
SecurityAlgorithms.HmacSha256);

    var claims = new[]
    {
        new Claim("UserId", user.Id.ToString()),
        new Claim("UserType", ((int)user.Type).ToString()),
        new Claim(ClaimTypes.Role, user.Type.ToString()),
        new Claim("Name", $"{user.Name} {user.Surname}"),
        new Claim("Login", user.Login),
        new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())
    };

    var token = new JwtSecurityToken(_configuration["Jwt:Issuer"],
        _configuration["Jwt:Issuer"],
        claims,
        expires: DateTime.Now.AddMinutes(120),
```

```
        signingCredentials: credentials);

    return _jwtSecurityTokenHandler.WriteToken(token);
}
```

Na podstawie danych w tokenie wyznaczany jest typ użytkownika, który z kolei zabezpiecza dostęp użytkownika do poszczególnych widoków:

```
[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme, Roles=
"Student,Lecturer")]
public IActionResult Index()
{
    return _userAccessor.Type switch
    {
        UserType.Student => RedirectToAction("Index", "Student"),
        UserType.Lecturer => RedirectToAction("Index", "Lecturer"),
        _ => RedirectToAction("Error", "Home")
    };
}
```

Hasła zapisywane w bazie danych są hashowane za pomocą wpisanego hasła oraz loginu użytkownika:

```
public async Task<User?> AuthenticateUser(string login, string password)
{
    var user = await _userRepository.GetUserLoginInformation(login);
    if (user == null) return null;

    var hashUser = new User
    {
        Name = user.Login
    };

    var passwordVerificationResult =
_passwordHasher.VerifyHashedPassword(hashUser, user.Password, password);

    if (passwordVerificationResult != PasswordVerificationResult.Success) return
null;

    var userInfo = await _userRepository.GetUserModel(user.UserId);
    var usertype = await _userRepository.GetUserType(user.UserId);

    if (userInfo == null) return null;

    return new User
    {
        Id = userInfo.Id,
        Name = userInfo.Name,
        Surname = userInfo.Surname,
        Type = usertype,
        Login = user.Login
    };
}



public async Task<bool> Register(RegisterViewModel model)
{
    model.Password = _passwordHasher.HashPassword(new User() { Name = model.Login
}, model.Password);
    return await _userRepository.Register(model);
}
```

4. Implementacja systemu baz danych

Dzięki wykorzystaniu programu *Visual Paradigm* do tworzenia diagramów opisujących bazę danych udało się poprawnie stworzyć podstawową strukturę bazy.

4.1. Tworzenie tabel i definiowanie ograniczeń

Wygenerowano zapytania SQL tworzące tabele i zmodyfikowano pod własne potrzeby ze względu na niedoskonałość programu – dodano klucze główne, odniesienia do kluczy obcych. Poniżej przedstawiono przykład stworzenia tabeli „Group”:

```
CREATE TABLE [Group] (
    Id nvarchar(20) NOT NULL,
    DayOfTheWeek int NOT NULL,
    StartTime time(5) NOT NULL,
    EndTime time(5) NOT NULL,
    Classroom nvarchar(255) NOT NULL,
    Capacity int NOT NULL,
    RegularityId int NOT NULL FOREIGN KEY REFERENCES Regularity(Id) ON DELETE CASCADE,
    TypeId int NOT NULL FOREIGN KEY REFERENCES GroupType(Id) ON DELETE CASCADE,
    CourseId int NOT NULL FOREIGN KEY REFERENCES Course(Id) ON DELETE CASCADE,
    SemesterId nvarchar(20) NOT NULL FOREIGN KEY REFERENCES Semester(Id) ON DELETE CASCADE,
    PRIMARY KEY (Id)
);
CREATE INDEX Group_DayOfTheWeek ON [Group] (DayOfTheWeek);
```

4.2. Implementacja mechanizmów przetwarzania danych

Procedury

Poniżej przedstawiono przykład stworzenia procedury uzyskującej średnią ważoną ocen w danym semestrze w zależności od id studenta oraz id semestru:

```
CREATE OR ALTER PROCEDURE GetStudentWeightedAverageGradeInSemester(
    @StudentId INT,
    @SemesterId NVARCHAR(20))
AS
BEGIN
    CREATE TABLE #Grade (
        Id INT,
        Grade DECIMAL(10, 3)
    );

    INSERT INTO #Grade
    SELECT s.Id, SUM(Grade * ECTS) / NULLIF(SUM(ECTS), 1) AS Grade
    FROM [Student] s
    INNER JOIN [Student_Group] sg ON s.Id = sg.StudentId
    INNER JOIN [Group] g ON sg.GroupId = g.Id
    INNER JOIN [Course] c ON g.CourseId = c.Id
    LEFT JOIN [Grade] gr ON sg.Id = gr.StudentInGroupId
    WHERE s.Id = @StudentId
        AND g.SemesterId = @SemesterId
        AND (gr.isFinal = 1 OR gr.IsFinal IS NULL)
    GROUP BY s.Id;

    SELECT TOP 1 Grade FROM #Grade;
    DROP TABLE #Grade;
END;
```

Widoki

Poniżej ukazane jest tworzenie widoku, który pozwoli na wyświetlenie istotnych informacji na temat grupy:

```
CREATE VIEW GroupInfo AS
SELECT
    g.DayOfTheWeek,
    g.StartTime,
    g.EndTime,
    gl.GroupId,
    g.Classroom,
    r.Name AS Regularity,
    gt.Name AS GroupType,
    u.Surname AS LecturerSurname,
    u.Name AS LecturerName,
    l.Id AS LecturerId,
    c.Name AS CourseName,
    c.ECTS
FROM [Group] g
INNER JOIN [Group_Lecturer] gl ON g.Id = gl.GroupId
INNER JOIN [Regularity] r ON g.RegularityId = r.Id
INNER JOIN [GroupType] gt ON g.Id = gt.Id
INNER JOIN [Lecturer] l ON gl.LecturerId = l.Id
INNER JOIN [User] u ON l.UserId = u.Id
INNER JOIN [Course] c ON g.CourseId = c.Id;
```

4.3. Implementacja uprawnień i innych zabezpieczeń

Stworzono zapytania gwarantujące uprawnienia dla każdego typu użytkownika według wcześniej ustalonych założeń.

Przykład nadania uprawnień do SELECT, INSERT, UPDATE, DELETE na tabeli *Absence* dla administratora:

```
GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Absence TO AdministrationRole;
```

Przykład nadania uprawnień do SELECT'a na widoku *GroupInfo* dla studenta:

```
GRANT SELECT ON dbo.GroupInfo TO StudentRole;
```

Przykład nadania uprawnień do wywołania procedury *RegisterLecturer* dla prowadzącego:

```
GRANT EXECUTE ON dbo.RegisterLecturer TO LecturerRole;
```

Przykład nadania uprawnień do INSERT'a obecności i nieobecności dla prowadzącego:

```
GRANT SELECT, INSERT ON dbo.Absence TO LecturerRole;
```

Przykład nadania uprawnień do zmiany hasła dla studenta:

```
GRANT SELECT, UPDATE ON dbo.UserLoginInformation TO StudentRole;
```

4.4. Testowanie bazy danych na przykładowych danych

4.4.1. Testy struktur bazy

W celu przetestowania struktury bazy stworzona została aplikacja w C# .NET która wykorzystuje framework NUnit do wykonywania testów jednostkowych. Następnie zostały wykonane testy uprawnień użytkowników do odpowiednich obiektów w bazie, procedur oraz widoków. Przykładowy test:

```
[Test]
Odwołania: 0
public void AllUsers_Should_HaveSelectPermissions_OnAllTables(
    [ValueSource(nameof(TableNames))] string table,
    [ValueSource(nameof(UserConnections))] Func<IDbConnection> getUserConnection)
{
    //Arrange
    _dbConnection = getUserConnection();
    _dbConnection.Open();
    _transaction = _dbConnection.BeginTransaction();

    //Act
    var act = () => _dbConnection.Query($"SELECT TOP 10 * FROM {table}", transaction: _transaction);

    //Assert
    act.Should().NotThrow();

    //Teardown
    _transaction.Rollback();
    _dbConnection.Dispose();
}
```

```
[Test]
Odwołania: 0
public void RegisterStudent_ShouldAddUserAndStudentTable()
{
    // Arrange
    var login = "Test";
    var passwordHash = "abab";
    var name = "Testtestcase";
    var surname = "Test";
    var fieldOfStudiesIdsJson = "[]";

    // Act
    _unitOfWork.Execute(@"
RegisterStudent @Login, @PasswordHash, @Name, @Surname, @FieldOfStudiesIdsJson
", new { login, passwordHash, name, surname, fieldOfStudiesIdsJson }, transaction: _transaction);

    // Assert
    var user = _unitOfWork.Query<User>(@"
SELECT TOP 1 *
FROM [User]
WHERE Name = @Name AND Surname = @Surname", new { name, surname }, transaction: _transaction).SingleOrDefault();

    user.Should().NotBeNull();

    var student = _unitOfWork.Query<Student>(@"
SELECT TOP 1 s.*
FROM [Student] s
INNER JOIN [User] u ON s.UserId = u.Id
WHERE u.Name = @Name AND u.Surname = @Surname", new { name, surname }, transaction: _transaction).SingleOrDefault();
    student.Should().NotBeNull();
    student?.UserId.Should().Be(user?.Id);

    _transaction.Rollback();
    _transaction.Dispose();
    _unitOfWork.Execute("DBCC CHECKIDENT ('[User]', RESEED, @identity);", new { identity = _userIdentity });
    _unitOfWork.Dispose();
}
```

Każdy test wykonywany był w transakcji aby nie wpływać na działanie samej bazy i umożliwić ich wielokrotne uruchomienie.

Testy zostały wykonane w podejściu ArrangeActAssert tzn. Przygotowanie środowiska, uruchomienie metody oraz asercja czyli sprawdzenie poprawności wykonania. Na końcu testu wykonywane jest wycofanie transakcji oraz odłączenie od bazy danych.

Przeprowadzone testy:

- Testy CRUD(Create, Read, Update, Delete):
  - Insert_Semester_Should_InsertSemesterIntoDb
  - Insert_User_Should_InsertUserIntoDb_WithAutoIncrementId
  - Update_Should_UpdateElementInDb
  - Delete_Should_DeleteElementFromDb
- Testy kontroli dostępu:
  - AllUsers_Should_HaveSelectPermissions_OnAllTables
  - AllUsers_Should_HaveSelectPermissions_OnAllViews
  - Administration_Should_HaveDeletePermissions_OnAllTables
  - Student_ShouldNot_HaveDeletePermissions_OnAnyTables
  - Lecturer_ShouldNot_HaveDeletePermissions_OnAnyTables
  - Administration_Should_HaveInsertPermissions_OnAllTables
  - Student_ShouldNot_HaveInsertPermissions_OnAnyTables
  - Lecturer_ShouldNot_HaveInsertPermissions_OnSpecifiedTables
  - Administration_Should_HaveUpdatePermissions_OnAllTables
  - Student_ShouldNot_HaveUpdatePermissions_OnSpecifiedTables
  - Lecturer_ShouldNot_HaveUpdatePermissions_OnSpecifiedTables
  - Student_Should_HaveUpdatePermissions_OnSpecifiedTables
  - Lecturer_Should_HaveInsertPermissions_OnSpecifiedTables
  - Lecturer_Should_HaveUpdatePermissions_OnLoginInformation
  - Administration_Should_HaveExecPermissions_OnAllStoredProcedures
  - Student_Should_HaveExecPermissions_OnSpecifiedStoredProcedures
  - Lecturer_Should_HaveExecPermissions_OnSpecifiedStoredProcedures

4.4.2. Testy wydajności indeksów

Poza testami jednostkowymi wykonany został również prosty test działania indeksów na wybranej tabeli:

Dodane zostało million rekordów w paczkach po 2000. Dodatkowo dla każdego przypadku wykonano 100 prób i wyliczono średni czas wykonywania poniższego zapytania:

Zapytanie: **SELECT * FROM [dbo].[Semester] WHERE StartDate = date**

Jako date podajemy dwie wybrane na początku wartości, a między nimi umieszczamy dane wypełniające. Wyniki:

Bez indeksu

```
Inserting 500/500
Search added first started
Search added first ended after 50 ms
Search added last started
Search added last ended after 45 ms
```

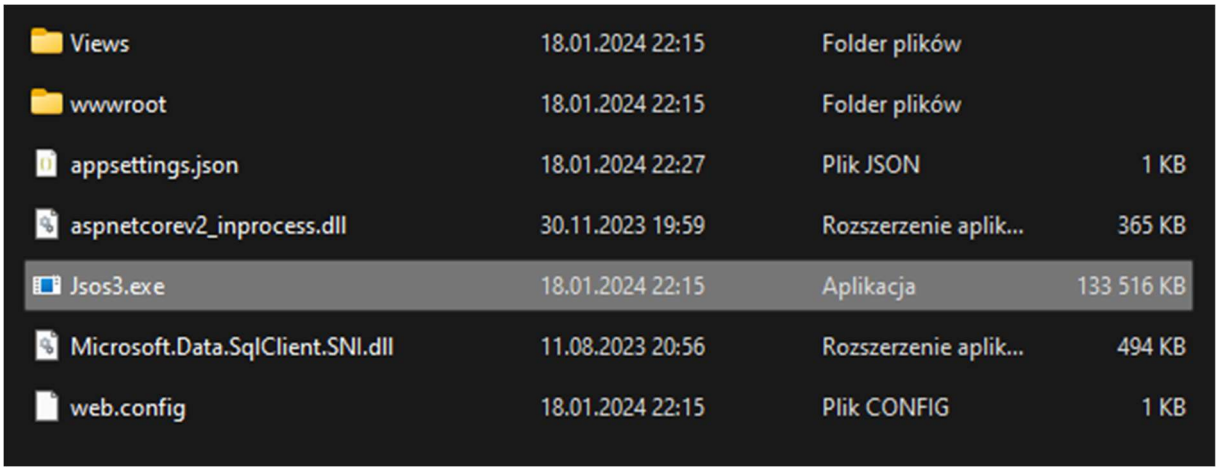
Z indeksem

```
Inserting 500/500
Search added first started
Search added first ended after 8 ms
Search added last started
Search added last ended after 0 ms
```

5. Implementacja i konfigurowanie systemu

5.1. Instalacja i konfigurowanie systemu

Instalację rozpoczynamy od pobrania pliku zip z zarchiwizowaną aplikacją. Jego zawartość powinna wyglądać następująco:



Konfiguracja

Aby uruchomić aplikację należy najpierw skonfigurować podłączenie do bazy. Oznacza to, że należy podać tzw. ConnectionStringi dla użytkowników bazy danych oraz klucz do szyfrowania haseł:

```
{
  "ConnectionStrings": {
    "StudentSql": "Server=172.28.0.1;Database=University-Main;User
Id=Student;Password=zaq1@WSX;TrustServerCertificate=True",
    "AdministrationSql": "Server=172.28.0.1;Database=University-Main;User
Id=Administration;Password=zaq1@WSX;TrustServerCertificate=True",
    "LecturerSql": "Server=172.28.0.1;Database=University-Main;User
Id=Lecturer;Password=zaq1@WSX;TrustServerCertificate=True"
  },
  "Jwt": {
    "Key": "token o dlugosci conajmniej trzydziestu dwoc znakow do szyfrowania
hasel",
    "Issuer": "adres_ip_aplikacji:port_aplikacji"
  }

}
```

Każdy connection string odpowiada połączeniu do bazy jednego użytkownika bazodanowego. Należy ustawienia te wypełnić takimi które odpowiadać będą infrastrukturze uruchomionej na uczelni. Dane do podania to: adresIp bazy danych, hasła dla użytkowników.

Zmiany te można dokonać w pliku appsettings.json albo dodać wartości te do zmiennych środowiskowych pod następującymi nazwami:

ConnectionStrings__StudentSql

ConnectionStrings__AdministrationSql

ConnectionStrings__LecturerSql

Jwt__Key

Jwt__Issuer

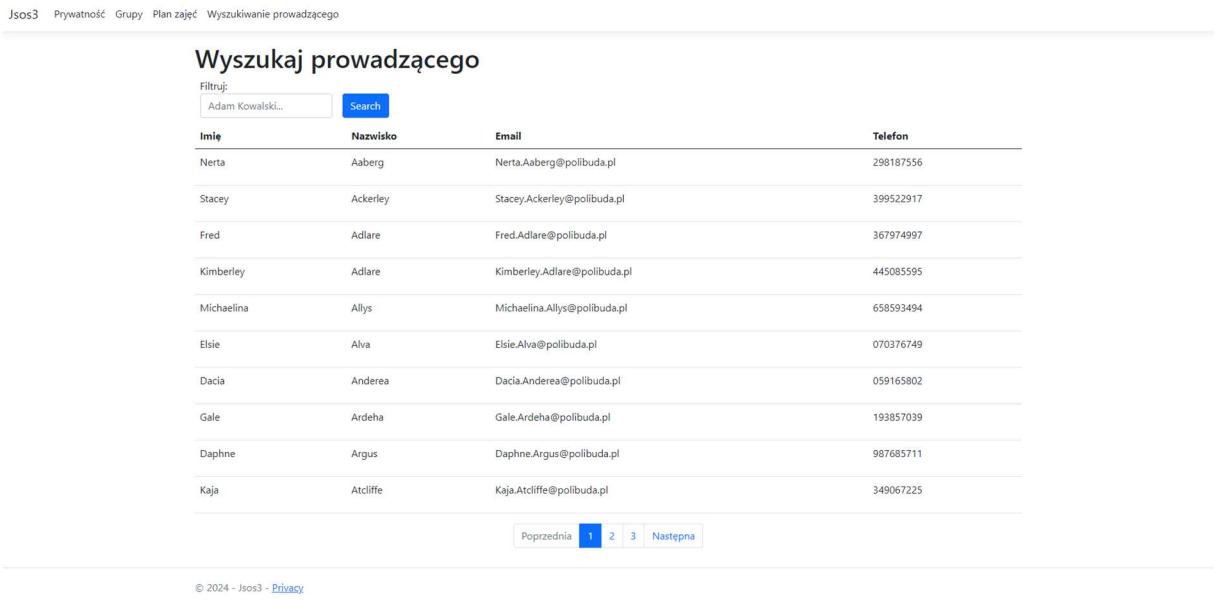
Uruchomienie

Aby uruchomić aplikację należy w wierszu poleceń będąc w katalogu aplikacji wpisać polecenie: .\Jsos3.exe --urls http://adresIpAplikacji:portAplikacji

Po włączeniu powinna pokazać się konsola:

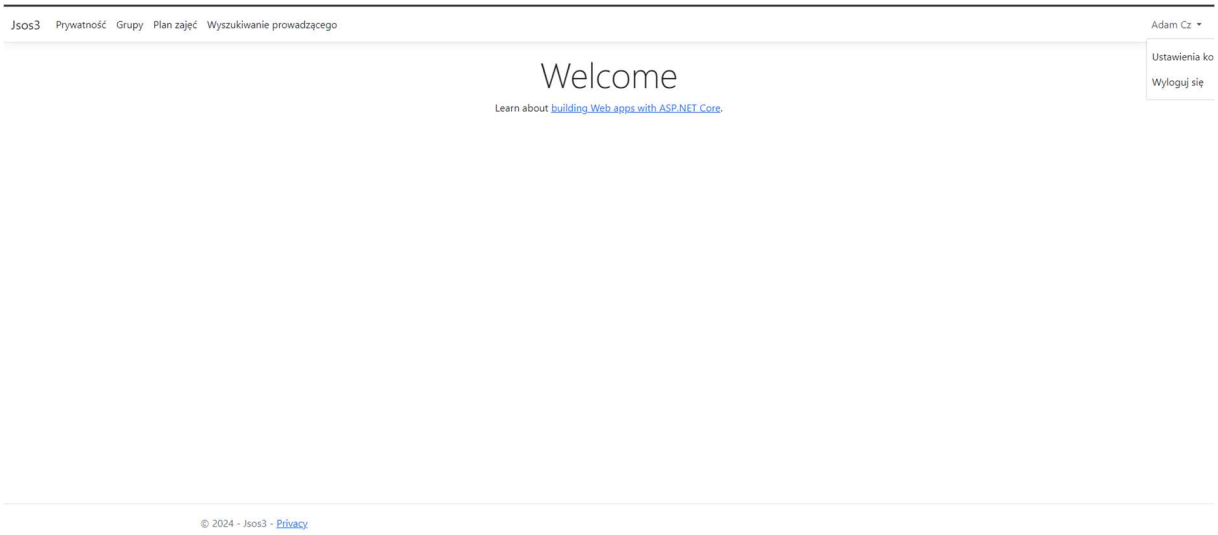
```
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://172.30.4.154:80
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\dulik\Documents\Git\DataBase2Project\Application\Jsos3\Jsos3\bin\Release\net8.0\publish\Nowy folder
```

Od tego momentu pod adresem ip hostującej maszyny strona powinna być dostępna:

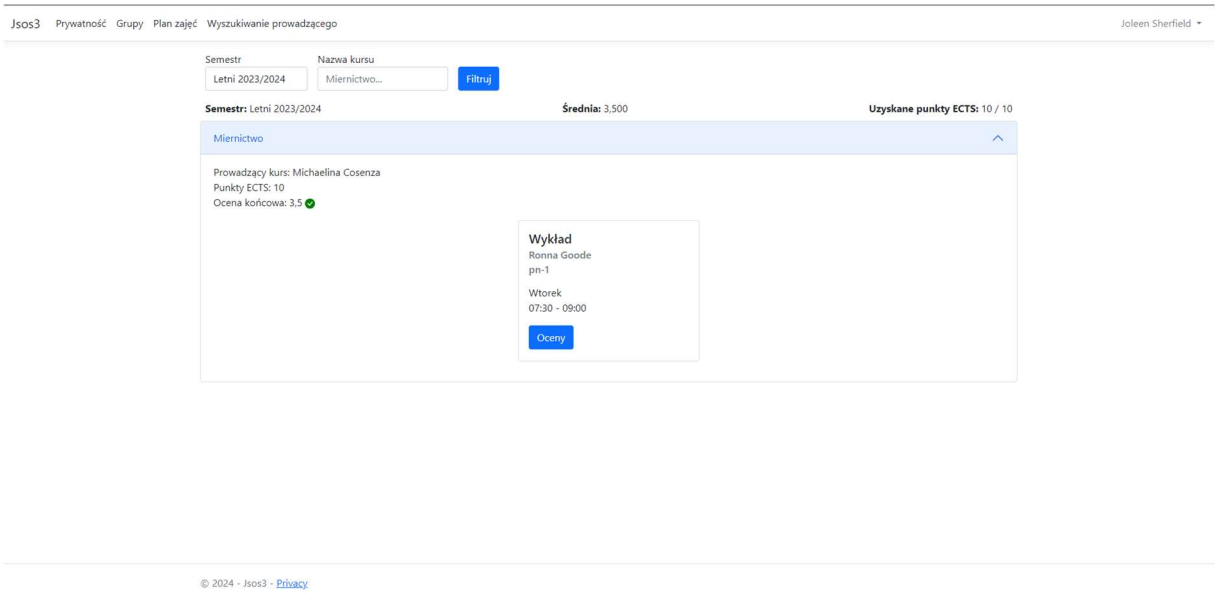


5.2. Instrukcja użytkowania aplikacji

Strona główna

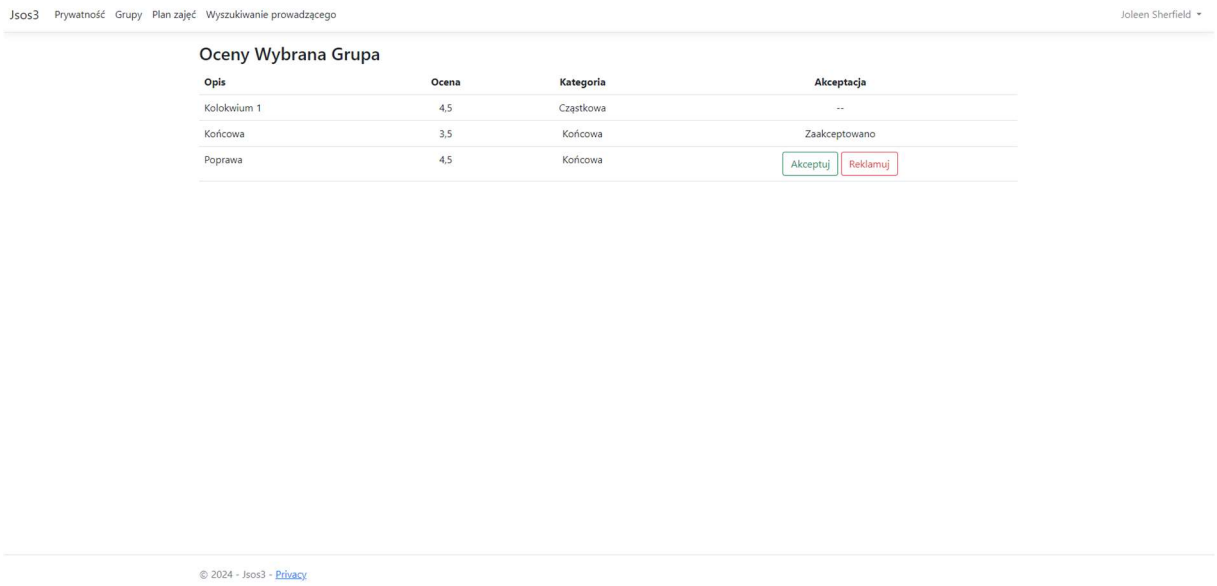


Grupy studenta



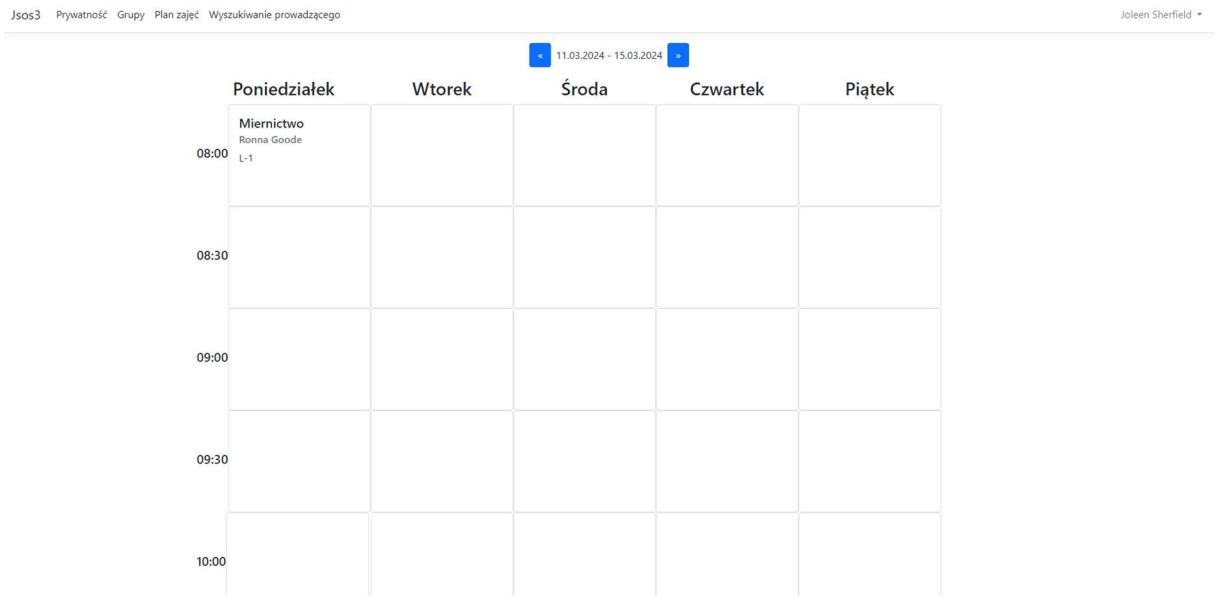
Na tym widoku można sprawdzić do jakich grup zapisany jest użytkownik. Istnieje możliwość filtrowania po nazwie kursu oraz semestrze. Widoczna jest również średnia ważona studenta oraz suma uzyskanych punktów ECTS z danego semestru. Po rozwinięciu kursu, widoczny jest prowadzący kurs, punkty ECTS, ocena końcowa za dany kurs oraz rodzaje zajęć, wraz z prowadzącymi te zajęcia, nazwą grupy zajęciowej oraz dniem tygodnia i dniem rozpoczęcia i zakończenia zajęć. Z tego poziomu można przejść też do widoku ocen z danego kursu.

Oceny perspektywa studenta



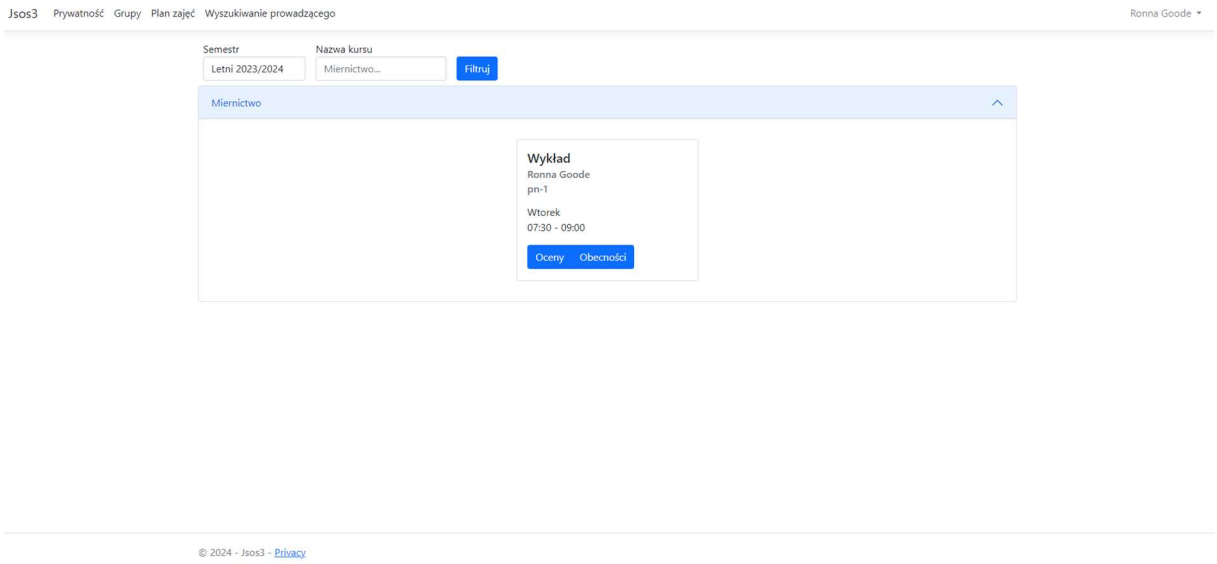
Student w widoku ocen widzi swoje oceny w wybranej wcześniej grupie zajęciowej. Ma on również możliwość reklamowania i akceptowania ocen końcowych. Widoczny jest opis, za co student dostał ocenę, jaką ocenę dostał, kategorię oceny (Cząstkowa bądź Końcowa) oraz czy ocenę „Zaakceptowano” bądź 2 przyciski „Akceptuj” i „Reklamuj”.

Tygodniowy plan zajęć



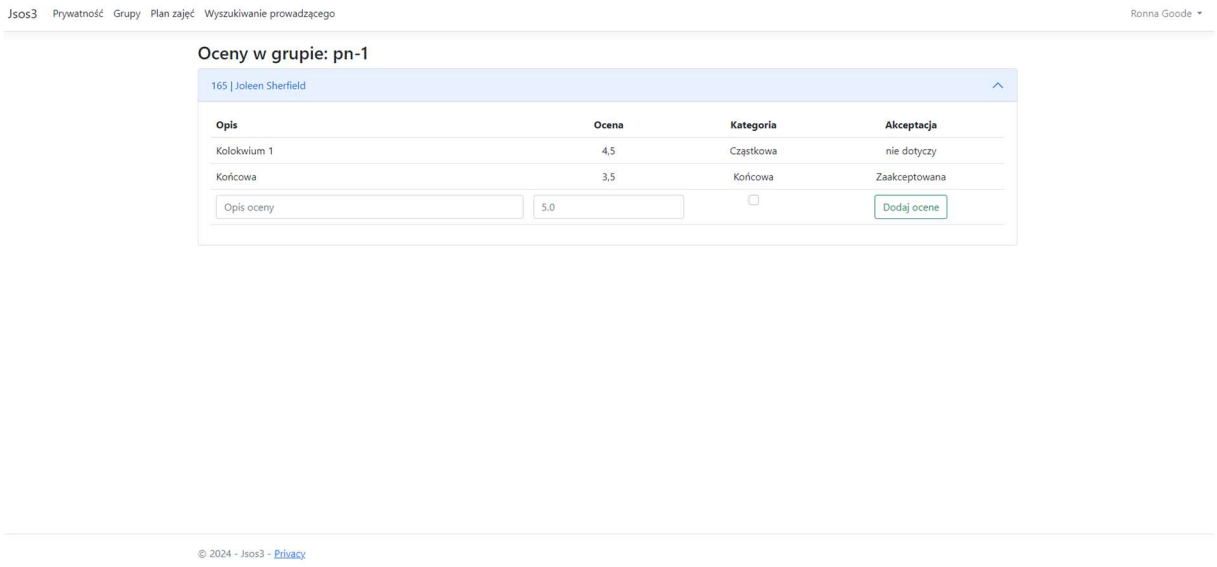
Prowadzący oraz student mają możliwość podejrzenia terminów swoich zajęć w danym tygodniu w postaci tabeli. Istnieje możliwość zmiany wybranego tygodnia.

Grupy prowadzącego



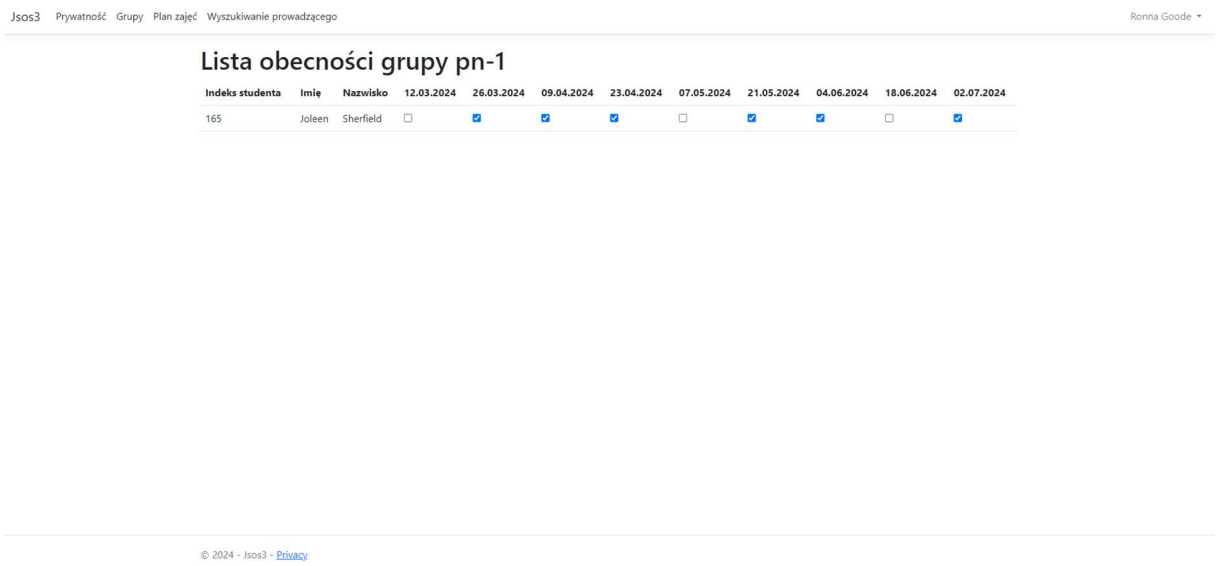
W widoku grup prowadzącego można analogicznie do tego widoku z perspektywy studenta podejrzeć grupy, dodatkowo jest możliwość przejścia do wstawiania ocen oraz obecności.

Oceny w grupie – widok prowadzącego



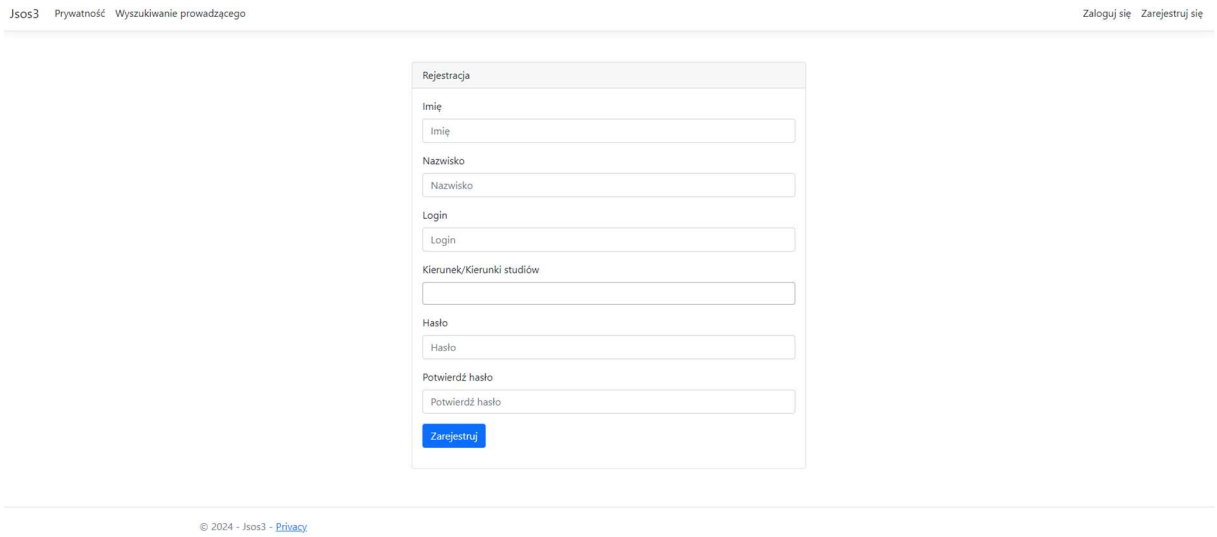
W tym widoku prowadzący ma możliwość dodania nowych ocen wraz z ich opisem, wartością oceny oraz zaznaczeniem, czy jest końcowa.

Obecności

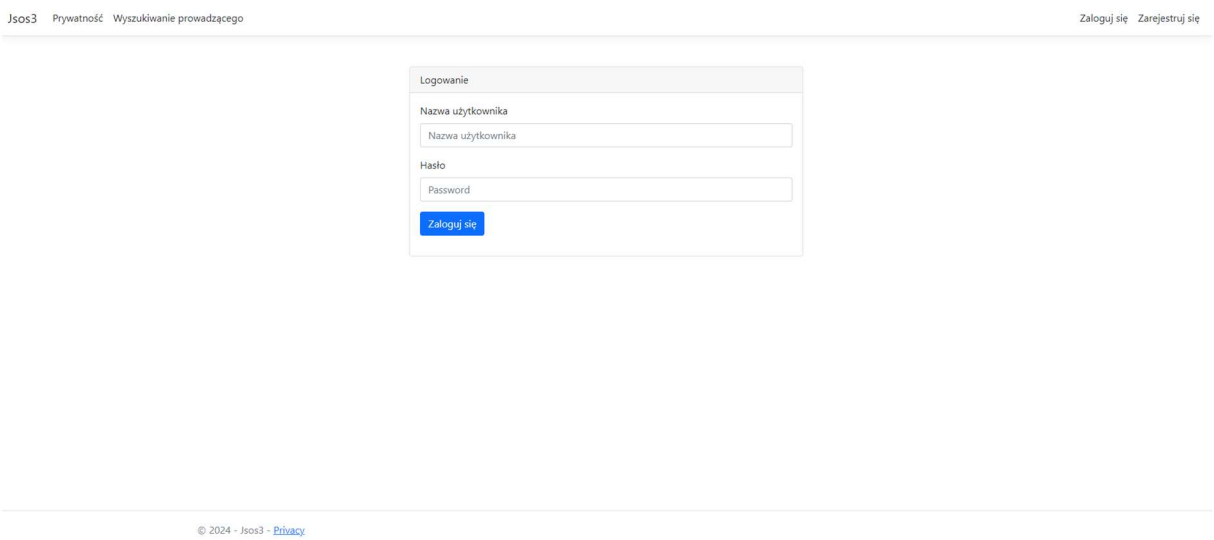


Ten widok, dostępny tylko dla prowadzącego pozwala na podgląd oraz edycję obecności studentów w danej grupie zajęciowej.

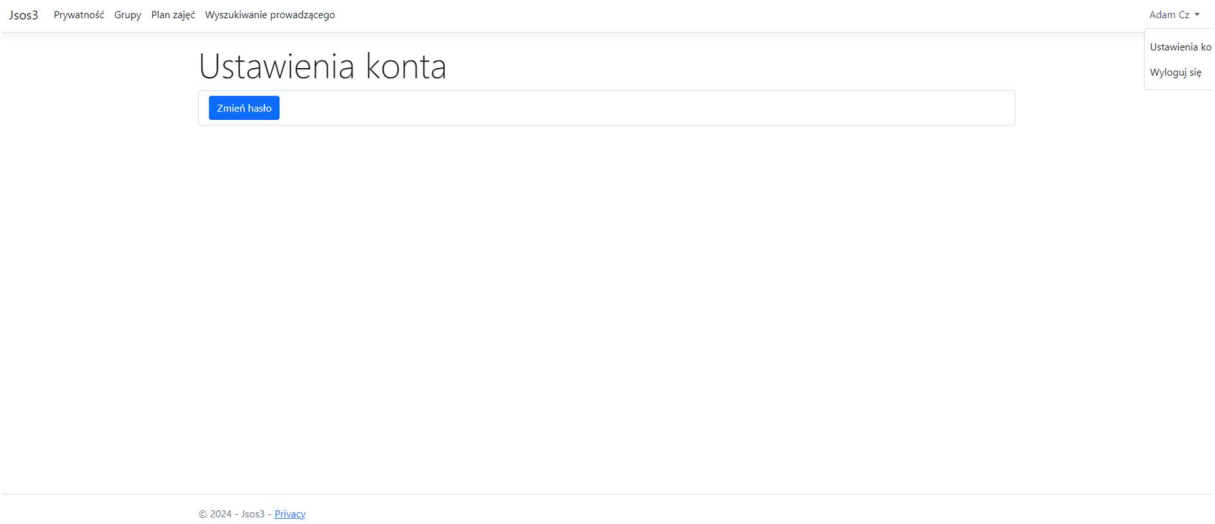
Rejestracja i logowanie



Po wpisaniu wszystkich danych w formularzu rejestracyjnym, należy kliknąć przycisk „Zarejestruj”. Wyskoczy strona logowania. Jeśli będziemy chcieli utworzyć użytkownika o zajętym już loginie lub nie zostaną wypełnione wszystkie pola, lub wpisane hasła nie będą się ze sobą zgadzać, wyskoczy komunikat „Registration unsuccessful”.



Po wpisaniu nazwy użytkownika i hasła dla użytkownika istniejącego już w bazie, logowanie odbędzie się pomyślnie i pojawi się strona główna. Natomiast gdy wprowadzimy dane nieistniejące w bazie danych, pojawi się komunikat „Invalid password”.



Zakładka ustawienia konta umożliwia na zmianę hasła.

5.3. Testowanie opracowanych funkcji systemu

W trakcie realizacji projektu przeprowadzono kompleksowe testy manualne aplikacji, które miały na celu zapewnienie wysokiej jakości i niezawodności systemu. Testy te wykonano w kilku etapach, wykorzystując różne środowiska i scenariusze.

Początkowo testy manualne skoncentrowano na lokalnych maszynach programistów. W tej fazie, aby ułatwić szybkie i skuteczne testowanie, wykorzystano nieprawdziwe implementacje użytkowników, co pozwoliło na dokładne sprawdzenie funkcjonalności aplikacji bez konieczności autoryzacji. Ta strategia pozwoliła programistom na szybkie identyfikowanie i naprawianie błędów na wczesnym etapie rozwoju, co znacząco przyspieszyło proces tworzenia aplikacji i zwiększyło jej stabilność.

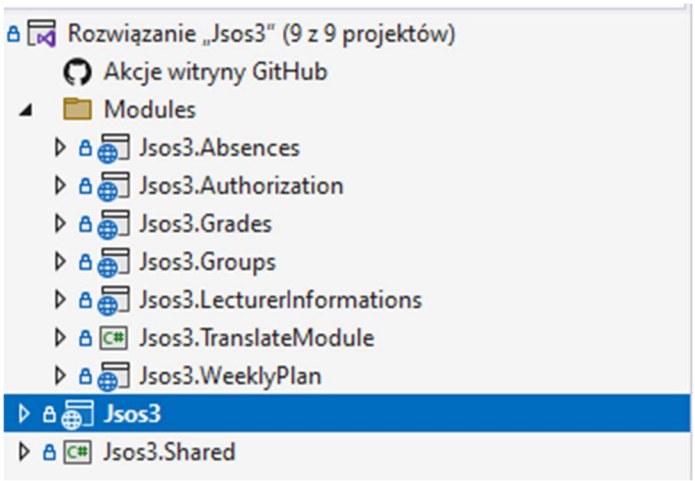
Następnie, po zakończeniu wstępnych testów i wprowadzeniu niezbędnych poprawek, przystąpiono do kolejnego etapu testowania. Tym razem aplikacja została uruchomiona jako plik wykonywalny (exe), co umożliwiło przetestowanie jej w warunkach bardziej

zbliżonych do rzeczywistego środowiska produkcyjnego. W tej fazie testów zaimplementowano pełne logowanie i autoryzację użytkowników, co pozwoliło na dokładne zweryfikowanie mechanizmów zabezpieczeń i autoryzacji oraz zapewnienie, że dane użytkowników są odpowiednio chronione.

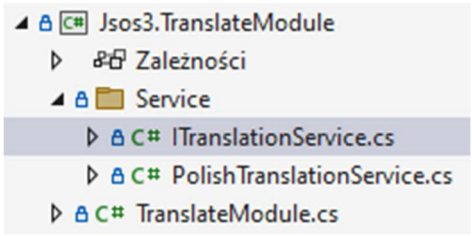
Testy te, przeprowadzone zarówno na lokalnych maszynach programistów bez pełnej autoryzacji, jak i na uruchomionym pliku exe z pełnym logowaniem, pozwoliły na dokładną weryfikację działania aplikacji w różnych scenariuszach i środowiskach. Dzięki temu udało się wykryć i rozwiązać potencjalne problemy na wczesnym etapie, zanim aplikacja została wdrożona na środowisko produkcyjne. Przeprowadzone testy manualne były kluczowym elementem procesu zapewnienia jakości i przyczyniły się do sukcesu projektu, gwarantując wysoki poziom niezawodności i bezpieczeństwa ostatecznego produktu.

5.4. Omówienie wybranych rozwiązań programistycznych

Aplikacja została napisana w języku C# z wykorzystaniem frameworka ASP.NET Core w wersji 8.0. Wykorzystany został wzór projektu typu MVC Razor Pages. Projekt został stworzony w uproszczonej strukturze Modularnego Monolitu. Jest to jedno z dwóch głownych podejść do tworzenia infrastruktury aplikacji. Drugim z nich jest podejście mikroserwisowe. W takim podejściu logika odpowiedzialna za wybraną domenę(grupę przepadków użycia) jest wydzielona do osobnej aplikacji. Takie podejście ma zalety takie jak np. niezależność języków programowania, frameworków oraz wdrażania. Jednakże takie podejście traci zalety w momencie kiedy nad całym systemem pracuje jeden mały zespół. W takim przypadku warto skorzystać z modularnego monolitu. Oznacza to, że złożone przypadki użycia zostały wydzielone nie do osobnych aplikacji ale do projektów w obrębie jednej aplikacji. Jeżeli dobrze został on zastosowany możliwe jest uzyskanie niezależność pojedynczych modułów (Separation of Concerns).



Każdy moduł może zawierać logikę dostępną dla innych modułów pod postacią interfejsu jak np. moduł Translate:

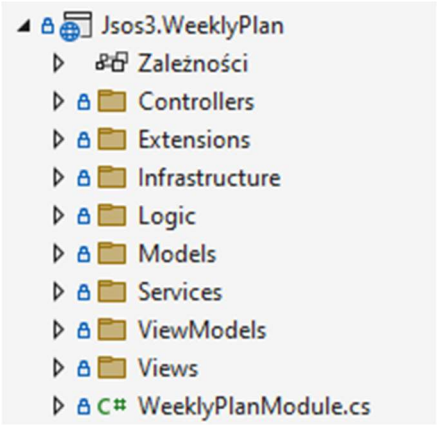



```
using Jsos3.Shared.Models;

namespace Jsos3.TranslateModule;

Odwołania: 6
public interface ITranslationService
{
    1 odwołanie
    string Translate(string key);
    Odwołania: 3
    string Translate(DayOfWeek dayOfWeek);
    Odwołania: 2
    string Translate(GroupType groupType);
    1 odwołanie
    string Translate(Regularity regularity);
}
```

Moduł może też przyjąć formę przypominającą niezależną aplikację jak np. WeeklyPlan. Projekt ten ma własne kontrolery, warstwę dostępu do danych(Data Access Layer) w formie przestrzeni nazw Infrastrucure oraz własne widoki i serwisy odpowiedzialne za realizowanie przypadków użycia. Warto dodać, że wszystkie klasy niebędące Modelami widoków bądź ich częścią i Kontrolerami są ukryte przed klientem poprzez słowo kluczowe internal:



```
public interface IPlanService
{
    Odwołania: 2
    Task<Dictionary<DateTime, List<WeeklyPlanDto>>> GetPlanForUser(int userObjectId, UserType userType, DateTime startOfWeek, DateTime endOfWeek);
}

Odwołania: 2
internal class PlanService : IPlanService
{
    private readonly IWeeklyPlanRepository _weeklyPlanRepository;

    Odwołania: 0
    public PlanService(IWeeklyPlanRepository weeklyPlanRepository)
    {
        _weeklyPlanRepository = weeklyPlanRepository;
    }

    Odwołania: 2
    public async Task<Dictionary<DateTime, List<WeeklyPlanDto>>> GetPlanForUser(int userObjectId, UserType userType, DateTime startOfWeek, DateTime endOfWeek)
    {
        var weeklyPlan = userType switch
        {
            UserType.Student => await _weeklyPlanRepository.GetStudentWeeklyPlan(userObjectId, startOfWeek, endOfWeek),
            UserType.Lecturer => await _weeklyPlanRepository.GetLecturerWeeklyPlan(userObjectId, startOfWeek, endOfWeek),
            _ => throw new ArgumentOutOfRangeException(nameof(userType), userType, null)
        };

        return weeklyPlan
            .Select(x => new WeeklyPlanDto
            {
                Date = x.Date,
                Regularity = x.RegularityId,
                Course = x.Course,
                GroupType = x.GroupTypeId,
                StartTime = x.StartTime,
                EndTime = x.EndTime,
                Classroom = x.Classroom,
                Lecturer = x.Lecturer
            })
            .GroupBy(x => x.Date)
            .ToDictionary(x => x.Key, x => x.ToList());
    }
}
```

Ostatecznie każdy moduł znajduje się na liście odniesień głównego projektu, a jego ich interfejsy, widoki i kontrolery rejestrowane w kontenerze Dependency Injection są za pomocą metod udostępnionych z każdego modułu:

```
using Jsos3.Groups;
using Jsos3.LecturerInformations;
using Jsos3.Shared;
using Jsos3.TranslateModule;
using Jsos3.WeeklyPlan;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();
builder.Services.AddMemoryCache();

builder.Services.AddAbsencesModule();
builder.Services.AddAuthorizationModule();
builder.Services.AddGradesModule();
builder.Services.AddGroupsModule();
builder.Services.AddLecturerInformationsModule();
builder.Services.AddWeeklyPlanModule();
builder.Services.AddSharedModule();
builder.Services.AddTranslateModule();

var app = builder.Build();
```

6. Podsumowanie i wnioski

Przedstawione cele i założenia projektu zostały zrealizowane.

Podsumowując, projekt wykazał, że wybór architektury modularnego monolitu i technologii ASP.NET w połączeniu z SQL Server jest solidnym rozwiązaniem dla aplikacji wymagających wysokiej niezawodności, wydajności i skalowalności. Osiągnięte cele projektowe potwierdzają, że zastosowane technologie i podejścia architektoniczne sprawdziły się w praktyce. Zastosowanie modularnego monolitu uprościło równoległą pracę w zespole, poprzez między innymi zminimalizowanie ryzyka powstania konfliktów między zmianami wprowadzonymi przez programistów w trakcie równoległej pracy. Zaleca się kontynuację monitorowania wydajności i bezpieczeństwa systemu, regularne aktualizacje komponentów oraz ewentualne rozważenie stopniowej ewolucji systemu w kierunku mikroserwisów, jeśli wzrost i wymagania biznesowe będą tego wymagać.

Repozytorium:

https://github.com/AdamusPL/DataBase2Project

Literatura:

- https://code-maze.com/using-dapper-with-asp-net-core-web-api/
- https://www.tutorialspoint.com/asp.net_core/asp.net_core_setup_mvc.htm
- https://bulldogjob.pl/readme/modular-monolith-modularnosc-droga-do-mikroserwisow
- https://www.c-sharpcorner.com/article/jwt-json-web-token-authentication-in-asp-net-core/
- https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver16