Projekt Programistyczny						
Rok akademicki	Termin	Rodzaj studiów	Kierunek	Prowadzący	Grupa	Sekcja
2019/2020	*Zajęcia zdalne*	SSI	INF	dr inż. Dariusz Myszor	BDiS	nd

Projekt Programistyczny

Autor:
Dominik Nowocień

Postęp prac

Prace obejmowały klika zagadnień. Zdecydowałem się rozszerzyć prace jakie wykonam do rzeczy które są dla mnie nowością a moją pewne zastosowanie praktyczne w programowaniu

- TDD,
- Logowanie,
- LinQ,

Refaktoryzacja / TDD

Skutkiem ubocznym pisania testów była refaktoryzacja. Za każdym razem wyglądała ona w zbliżony sposób:

1. Studiowanie kodu i jego funkcjonalności;
2. Skopiowanie nagłówka funkcji wraz z argumentami (moim zadaniem nie były duże zmiany, tak naprawdę nową interpretację istniejących interfejsów),
3. Pisanie kodu,
4. Testy,
5. Sprawdzenie końcowej funkcjonalności.

Należy zaznaczyć ze punkty 3 i 4 były ponawiane najczęściej 2-3 razy pomimo stosunkowo prostego kodu.

Efektem tej części prac jest powstanie klasy o identycznej funkcjonalności w stosunku od pierwotnego kodu. Jednak jest ona przetestowana i zrefaktoryzowana, Dodatkowo została dołożona funkcja logowania błędów.

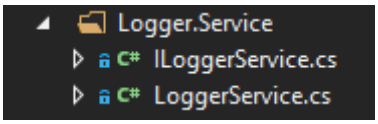
```
public void UpdateClass(ClassDTO updatedClassDTO, int classId)
{
    if (IsClassRepositoryContaining(classId))
    {
        ClassDAO updatedClassDAO = new ClassDAO
        {
            Id = classId,
            Name = updatedClassDTO.Name
        };
        classRepository.Update(updatedClassDAO);
    }
    else
    {
        throw new GradebookServerException("Nie ma klasy o podanym numerze Id");
    }
}

private bool IsClassRepositoryContaining(int classId)
{
    foreach (ClassDAO checkedClass in classRepository.GetAll().ToList())
    {
        if (checkedClass.Id == classId)
        {
            return true;
        }
    }
    return false;
}
```

```
public void UpdateClass(ClassDTO updatedClassDTO, int classId)
{
    if (classRepository.GetAll().Any(x => x.Id == classId))
    {
        classRepository.Update(new ClassDAO
        {
            Id = classId,
            Name = updatedClassDTO.Name
        });
    }
    else
    {
        logerService.Debug("Nie ma klasy o podanym numerze Id", this);
        throw new GradebookServerException("Nie ma klasy o podanym numerze Id");
    }
}
```

Rys.1 i 2 refaktoryzacja – przedstawiają wybrany fragment kodu który został przepisany zgodnie ze sztuką TDD

Loger

Można znaleźć wiele logerów jednak ja zdecydowałem się na użycie NLog.
Był stosunkowo prosty w implementacji i jest kompatybilny z .Net Core.
Jego stworzenie wiązało się ze stworzeniem nowej usługi:



Rys. 1 loger – usługa i jej interfejs.

Stworzyłem nowy interfejs, który w przyszłości pozwoli na łatwiejszą zmianę go na inny, alternatywny serwis logowania.

```
Odwołania: 6
public interface ILogerService
{
    Odwołania: 4
    void Debug(string message, object senderData);

    1 odwołanie
    void Info(string message, object senderData);

    1 odwołanie
    void Warning(string message, object senderData);

    1 odwołanie
    void Error(string message, object senderData);

    Odwołania: 4
    void Critical(string message, object senderData);
}
```

Rys.2 loger – kod interfejsu logera

Konfiguracja w programie to zaledwie kilka linii kodu, a funkcjonalność może się okazać bardzo przydatna. Zwłaszcza w poszukiwaniu błędów powstałych w czasie użytkowania aplikacji przez użytkownika końcowego.

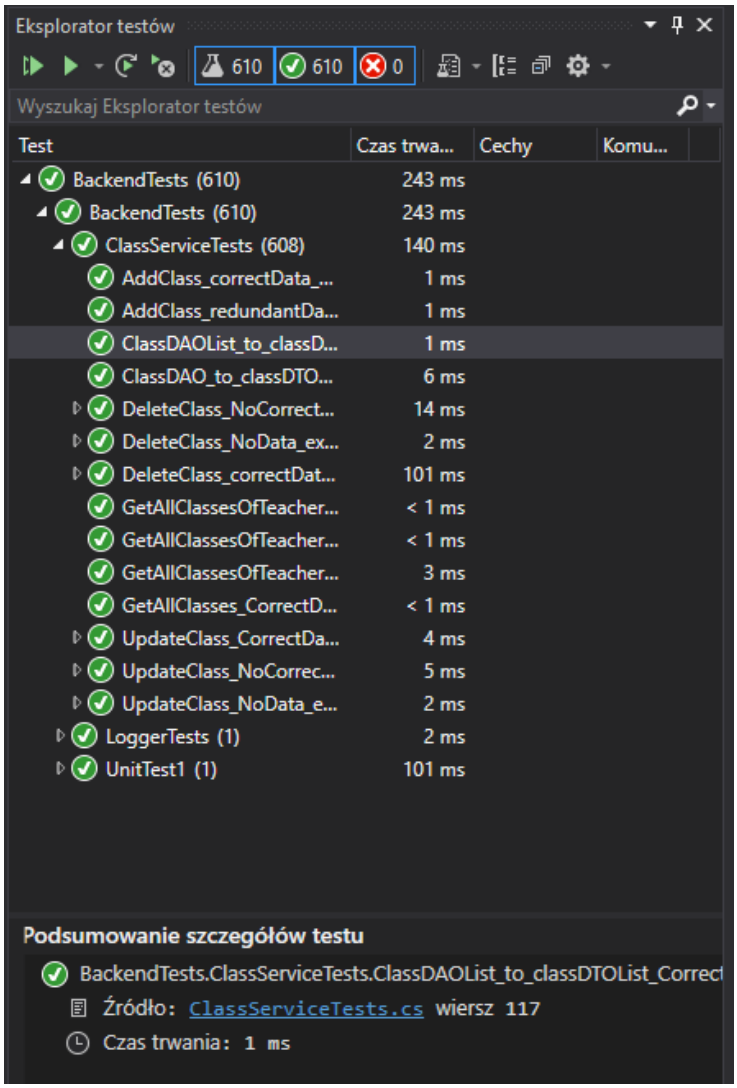
Testy

Największa część prac skupiła się na nauce pisania testów. Testy objęły sprawdzanie poprawności działania klasy ClassService. Jest to klasa która obsługuję komunikację kontrolerów z modelem bazy danych.
Od samego początku próbowałem pisać z użyciem TDD. Jedynie wsteczne testy jakie napisałem to początkowo w celu obycia się z biblioteka xUnit i Moq.

Początkowo mockowanie wydawało się bardzo nieintuicyjne jednak po zrozumieniu działania jest to bardzo przydatna funkcjonalność.

Poznawanie struktury testów było bardzo dużą częścią prac. Czytanie o testach i dobranie środowiska też okazało się kluczowe. Od razu zacząłem korzystać z xUnit z pominięciem MSTest ze względu na słabszą dokumentacje i gorsze opinie. Konfiguracja całości była dosyć trudna ze względu na chęć całkowitego rozdzielenia testów od części właściwej aplikacji.

Nakładka której używam to wbudowany w środowisko VS Studio Eksplorator Testów:



Rys.1 testy – wynik testowania programu GradeBook

Wnioski

Testy podczas nauki ich tworzenia ulegały ciągłym modyfikacjom i faktoryzacji. Podczas prac próbowałem zapoznać się z różnymi atrybutami testów.

```
[Fact]
```

```
[Theory]
[ClassData(typeof(Ids_testData))]
```

```
[Theory]
[InlineData(34)]
```

Rys.2, 3 i 4 testy – Różne atrybuty testów definiujące ich sposób wywoływania przez środowisko testwe.

Nazewnictwo

Nie znalazłem jednego zdefiniowanego typu nazywania testów, jednak kierując się wskazówkami zawartymi w różnych artykułach przyjąłem następującą zasadę:

[nazwa testowanej metody] _ [dostarczone dane] _ [spodziewany efekt]

Przykłady:

```
Odwołania: 0
public void DeleteClass_correctData_expectPass(int id)
```

```
Odwołania: 0
public void DeleteClass_NoCorrectData_expectExceprionThrown(int id)
```

```
Odwołania: 0
public void DeleteClass_NoData_expectExceprionThrown(int id)
```

```
Odwołania: 0
public void GetAllClasses_CorrectData_expectEquals()
```

Rys.5, 6, 7 i 8 testy – przyjęte nazewnictwo

Budowa

Każdy napisany test składa się z następujących części.

- arrange
- act
- assert

Arrange – przygotowanie mocków, danych wejściowych i oczekiwanych rezultatów

Act – wywołanie testowanej metody i przechwycenie jej danych.

Assert – sprawdzenie poprawności wykonania testów i zwróconych danych.

Moq

Biblioteka pozwalająca pisać mocki czyli elementy kodu podszywające się pod prawdziwe klasy. Jej budowa pozwala na zdefiniowanie działania poszczególnych zamockowanych funkcji.
Daje również dodatkową możliwość sprawdzenia zachowania się mockowanych serwisów wewnątrz testowanych metod, co również wykorzystałem w sekcji assert testów.

Przyklady

```
[Fact]
Odwołania: 0
public void GetAllClassesOfTeacher_EmptyData_expectEquals()
{
    // arrange
    int teacherID = 4;

    List<LessonDAO> lissonsDAOList = new List<LessonDAO>();
    List<ClassDTO> expected = new List<ClassDTO>();

    var classRepositoryMock = new Mock<IRepository<ClassDAO>>();
    classRepositoryMock.Setup(x => x.Get(It.IsAny<int>())).Returns(new ClassDAO() { Name = "someClass" });

    var lessonRepositoryMock = new Mock<IRepository<LessonDAO>>();
    lessonRepositoryMock.Setup(x => x.GetAll()).Returns(lissonsDAOList);

    var loggerMock = new Mock<ILogerService>();

    IClassService testService = new ClassService(
        classRepositoryMock.Object,
        lessonRepositoryMock.Object,
        loggerMock.Object);

    // act
    ClassListDTO classListDTO = testService.GetAllClassesOfTeacher(teacherID);

    // assert
    Assert.Equal(expected, classListDTO.ClassList);
}
```

Rys.9 testy – przykład testu z atrybutem [Fact]

```
[Theory]
[ClassData(typeof(Ids_testData))]
Odwołania: 0
public void UpdateClass_NoData_expectPass(int id)
{
    // arrange
    List<ClassDAO> mockClassDAOList = new List<ClassDAO>();

    ClassDTO mockData = new ClassDTO()
    {
        Id = id,
        Name = "className4",
    };

    var classRepositoryMock = new Mock<IRepository<ClassDAO>>();
    classRepositoryMock.Setup(x => x.GetAll()).Returns(mockClassDAOList);
    var lessonRepositoryMock = new Mock<IRepository<LessonDAO>>();
    var loggerMock = new Mock<ILogerService>();

    IClassService testService = new ClassService(
        classRepositoryMock.Object,
        lessonRepositoryMock.Object,
        loggerMock.Object);

    // act
    // assert
    Assert.Throws<GradebookServerException>(() => testService.UpdateClass(mockData, mockData.Id));
    loggerMock.Verify(x => x.Debug(It.IsAny<string>(), It.IsAny<ClassService>()), Times.Once);
    classRepositoryMock.Verify(x => x.GetAll(), Times.Once);
    classRepositoryMock.Verify(x => x.Update(It.IsAny<ClassDAO>()), Times.Never);
}
```

Rys.10 – testy – test z atrybutem [Theory] i przekazaniem danych testowych z klasy dziedziczącej po IEnumerable. Warto zwrócić uwagę na sekcję assert w której oczekiwanym efektem działania jest zwrócenie wyjątku.

Wnioski

// Komentowanie

Najważniejszy wnioskiem wyciągniętym w pierwszych godzinach prac jest praca nad większym szacunkiem do dokumentacji. Kod w programie nie był skomentowany a działa metod pomimo bycia oczywistym na etapie prac rok temu nagle stało się niejednoznaczne i mocno utrudnione.

TDD

Technika dla której widzę pewne zastosowanie jednak nie bardzo restrykcyjne.
Nie wyobrażam sobie w przyszłości pisania testów do każdej najmniejszej metody. Po krótkiej przygodzie wydaje mi się że w testach należy znaleźć pewniej balans który pozwoli nam na zasadne ich stosowanie. Nie widzę potrzeby testowania bardzo prostych funkcji i pokrycia kodu bliskiego 100%. Pisanie testów jest czasochłonnym zadaniem i uważam że należy się do nich stosowań w dużych projektach z naciskiem na krytyczne funkcjonalności.

Loger

Funkcja logera jest przydatna w debugowaniu. Podczas pisania pierwszej wersji projektu została całkowicie pominięta jednak jej zastosowanie mogłoby pomoc w debugowaniu kodu lub zachowywaniu aktywności użytkowników pracujących z aplikacją.

Linq

Moje odkrycie na studiach. Wspaniałe narzędzie skracające kod przyspieszające pisanie. Moje zastosowanie kwerend Linq do tej pory było bardzo ograniczone jednak teraz widzę dla niego szersze zastosowanie. Kod stał się krótszy i czytelniejszy.

Uzupełnienie sprawozdania o kratę wstępną.

Opis projektu:

Dziennik elektroniczny jest zrealizowany w formie aplikacji webowej umożliwiającej dostęp do bazy ocen i obecności, planu zajęć. Jego głównym celem jest usprawnienie pracy nauczycieli, a także uproszczenie wymiany informacji na linii nauczyciel-uczeń.

System umożliwia użytkownikom sprawdzanie swoich planów zajęć na których będą znajdowały się informacje o dniu tygodnia, w którym odbywają się dane zajęcia oraz rodzaju tych zajęć. Plan uczniów będzie informował z jakim nauczycielem będą odbywać się zajęcia natomiast plan nauczyciela przedstawi mu z jaką klasą będzie prowadził lekcje. Na każdym planie będzie widoczny dzień tygodnia oraz godzina, o której odbywają się dane zajęcia. Będą na nim także widoczne zmiany wprowadzone przez administratora planu.

Dziennik elektroniczny przechowuje także informacje na temat ocen i obecności uczniów. W bazie danych systemu każda ocena jest powiązana z nauczycielem który ją wystawił, uczniem który ją otrzymał oraz zajęciami w ramach których została ona wystawiona. Nauczyciel może także dodać do niej swój komentarz. Oceny mają wagi, pozwalające pod koniec semestru obliczyć średnią ważoną z całego przedmiotu dla konkretnego ucznia. Oceny mogą zostać modyfikowane w przypadku pomyłki albo poprawy oceny. W przypadku pomyłki ocena będzie edytowana, natomiast w przypadku poprawy ocena widziana w systemie będzie równa średniej ze starej i nowej oceny.

System umożliwia również kontrolowanie frekwencji uczniów na zajęciach. Występują w nim następujące oznaczenia:
- obecność
- nieobecność usprawiedliwiona
- nieobecność nieusprawiedliwiona

Wpisy do bazy wprowadzane będą przez nauczycieli w trakcie zajęć, a przeglądane są przez uczniów i nauczycieli, których obecności bezpośrednio dotyczą. Każdy wpis może być zmieniona w razie pomyłki albo usprawiedliwienia nieobecności. Na podstawie wprowadzonych danych system oblicza frekwencje ucznia.

Kolejną funkcjonalnością systemu jest możliwość zalogowania się do konta użytkownika. Po połączeniu z aplikacją użytkownik jest witany przez panel logowania, gdzie jest proszony o podanie nazwy użytkownika oraz hasła. System sprawdza poprawność wprowadzonych danych logowania i jeśli są prawidłowe, udziela dostępu do reszty funkcjonalności zgodnie z typem konta na który został zalogowany użytkownik. Istnieje możliwość zalogowania się jako:
- administrator
- nauczyciel
- uczeń

Konta są tworzone przez administratora, dlatego projekt nie przewiduje panelu rejestracji do systemu. Różne typy kont dają dostęp do innych uprawnień z zakresu przeglądania i

modyfikowania danych w systemie. Pierwsze hasło do konta jest generowane przez administratora tworzącego konto, jednak po zalogowaniu użytkownik będzie miał możliwość zmiany tego hasła. W przypadku utraty hasła administrator może wygenerować nowe dla każdego konta.

Administrator systemu ma dostęp do dodawania, usuwania oraz modyfikowania kont uczniów, nauczycieli oraz administratorów. Jego zadaniem będzie tworzenie klas, przedzielanie do nich uczniów oraz delegacja odpowiednich nauczycieli do nauczania konkretnych przedmiotów, a także tworzenie planów lekcji dla poszczególnych klas i nauczycieli. System umożliwia administratorowi wprowadzenie ewentualnych zmian do planu w razie potrzeby.

Nauczyciel może zostać przypisany do jednego bądź paru przedmiotów. W ramach zajęć może on wystawić uczniom oceny i uwagi oraz sprawdzać obecność uczniów na zajęciach. Aplikacja pozwala na przypisanie zajęciom tematu w celu sprawdzenia jakiej tematyki dotyczyły. Może przeglądać i edytować obecności i oceny, które zostały przez niego wystawione, a także przeglądać swój plan zajęć.

Uczniowie są przydzielani do konkretnych klas. Każda klasa posiada swój plan zajęć, w którym można znaleźć informacje o zajęciach realizowanych w tygodniu. Uczeń oprócz sprawdzania frekwencji i obecności w poszczególne dni otrzymuje także możliwość sprawdzania swoich ocen.

Wymagania niefunkcjonalne

- szybki czas reakcji porównywalny do innych aplikacji webowych
- niezawodny
- nie zajmujący zasobów na urządzeniu użytkownika
- możliwy do uruchomienia na każdym systemie operacyjnym
- wymagający jedynie przeglądarki internetowej
- działający na każdej przeglądarce
- odporny na błędy użytkownika
- bezpieczny
- łatwy w obsłudze
- z systemu może korzystać wiele użytkowników na raz
- posiadający niskie koszty utrzymania
- posiadający kompletną dokumentacje techniczną
- skalowalny do różnych rozmiarów ekranów

Wymagania funkcjonalne:

Oceny:

- Wgląd do ocen
- Wstawianie ocen
- Edycja i usuwanie ocen

    - poprawa ocen

- Obliczanie średniej
- Dodanie tematu oceny

Obecności:

- Wgląd do obecności
- Wystawianie obecności
    - obecność
    - nieobecność nieusprawiedliwiona
    - nieobecność usprawiedliwiona

- Edycja i usuwanie obecności
    - usprawiedliwienia

- Obliczanie frekwencji
- Dodanie tematu obecności

Plan lekcji:

- Wgląd do planu lekcji
- Wprowadzanie i edycja planu lekcji
- Przypisanie planu do
    - nauczyciela
    - klasy

Uwagi:

- Wstawianie uwag
- Przeglądanie uwag

Konta:

- Logowanie
- Dodawanie kont o określonej roli
- Usuwanie i edycja użytkowników
- Edycja własnego konta przez każdego użytkownika

Przedmioty:

- Dodawanie i usuwanie przedmiotów
- Przypisywanie przedmiotów do nauczycieli i klas

Klasy:

- Dodawanie i usuwanie klas

- Przypisywanie klas do nauczycieli i uczniów

Administrator:

- Może logować się do systemu.
- Może zmieniać dane potrzebne do logowania.
- Może zarządzać kontami innych użytkowników:
    - Może tworzyć konta nowych użytkowników.
        - Może tworzyć konta uczniów.
        - Może tworzyć konta nauczycieli.
        - Może tworzyć konta administratorów.
    - Może edytować konta istniejących użytkowników:
        - Może generować nowe hasło dla użytkownika.
    - Może usuwać konta użytkowników.

- Może zarządzać strukturą powiązań w systemie:
    - Może zarządzać klasami:
    - Może tworzyć klasy.
    - Może przypisywać uczniów do danej klasy.
    - Może przypisywać klasie przedmioty które realizuje w ramach zajęć.
    - Może usuwać klasy
    - Może zarządzać przedmiotami:
    - Może tworzyć nowe przedmioty.
    - Może przypisywać przedmiotom nauczycieli, którzy je realizują.
    - Może usuwać przedmioty.
    - Może zarządzać planem lekcji:
    - Może tworzyć nowe plany zajęć.
    - Może modyfikować istniejące plany zajęć.
    - Może usuwać istniejące plany zajęć.

Nauczyciel:

- Może logować się do systemu.
- Może zmieniać dane potrzebne do logowania.
- Może przeglądać informacje go dotyczące:
    - Może przeglądać swój plan lekcji.
    - Może przeglądać przedmioty, do których jest przydzielony.
    - Może przeglądać klasy, w których prowadzi zajęcia.

- Może przeglądać informacje dotyczące klas, w których naucza:
    - Może przeglądać obecności, które dotyczą zajęć, które prowadzi.
    - Może przeglądać frekwencję uczniów w bieżącej chwili.
    - Może przeglądać oceny, które dotyczą zajęć, które prowadzi.
    - Może przeglądać średnią ocen uczniów w bieżącej chwili.
    - Może przeglądać uwagi uczniów, których uczy.

- Może modyfikować oceny uczniów w ramach konkretnych zajęć:
    - Może dodawać oceny powiązane z konkretną datą i tematem zajęć.
    - Może modyfikować oceny wystawione przez niego wcześniej:

    - Może zmodyfikować ocenę.
    - Może wpisać ocenę jako poprawę danej oceny.

- Może modyfikować obecności uczniów w ramach konkretnych zajęć:
    - Może sprawdzać obecność w ramach zajęć realizowanych w konkretnym dniu.
    - Może modyfikować obecności wystawione przez siebie wcześniej:
    - Może modyfikować obecności uczniów.
    - Może usprawiedliwiać nieobecności uczniów.

- Może dodawać uwagi dotyczące konkretnych uczniów.

Uczeń:

- Może logować się do systemu.
- Może zmieniać dane potrzebne do logowania.
- Może przeglądać informacje go dotyczące:
    - Może przeglądać informacje dotyczące ocen:
    - Może przeglądać poszczególne oceny.
    - Może sprawdzać swoją średnią w bieżącej chwili.
    - Może przeglądać informacje dotyczące obecności
    - Może przeglądać poszczególne obecności.
    - Może sprawdzać swoją frekwencję w bieżącej chwili
    - Może przeglądać swój plan lekcji.
    - Może przeglądać swoje uwagi.

System:

- Oblicza statystki w czasie rzeczywistym:
    - Oblicza frekwencję ucznia na podstawie jego obecności.
    - Oblicza średnią ocen ucznia na podstawie jego ocen.

Technologia

Języki programowania

Backend - .NET Core
Baza danych - SQL Server Express
Frontend – JS, html, css

Użyte narzędzia

- git,
- VS code,
- VS studio,
- SQL Server,

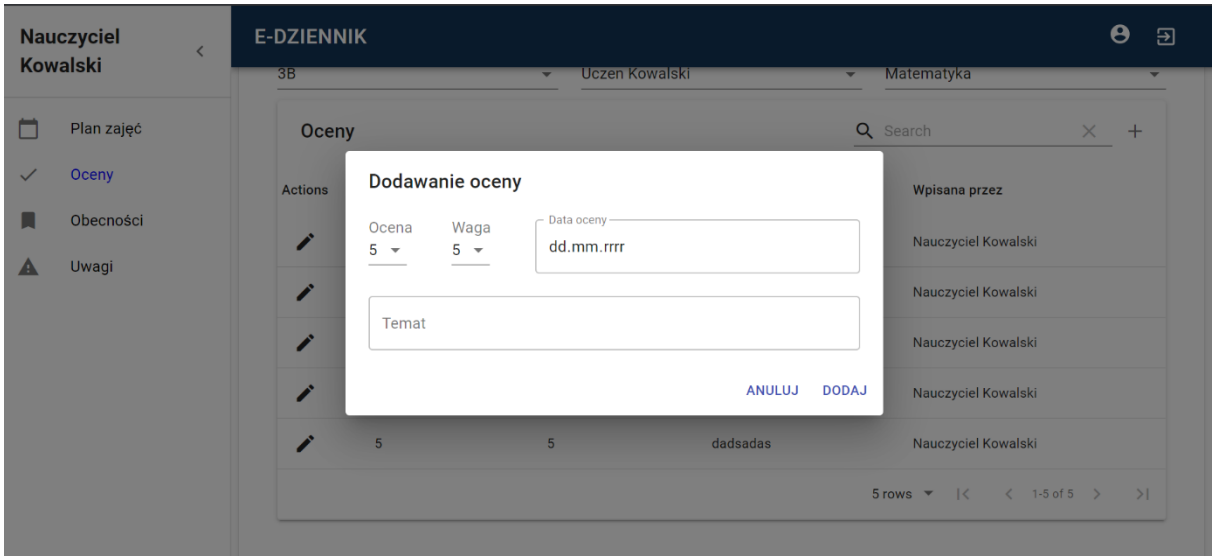
- Postman,

Specyfikacja

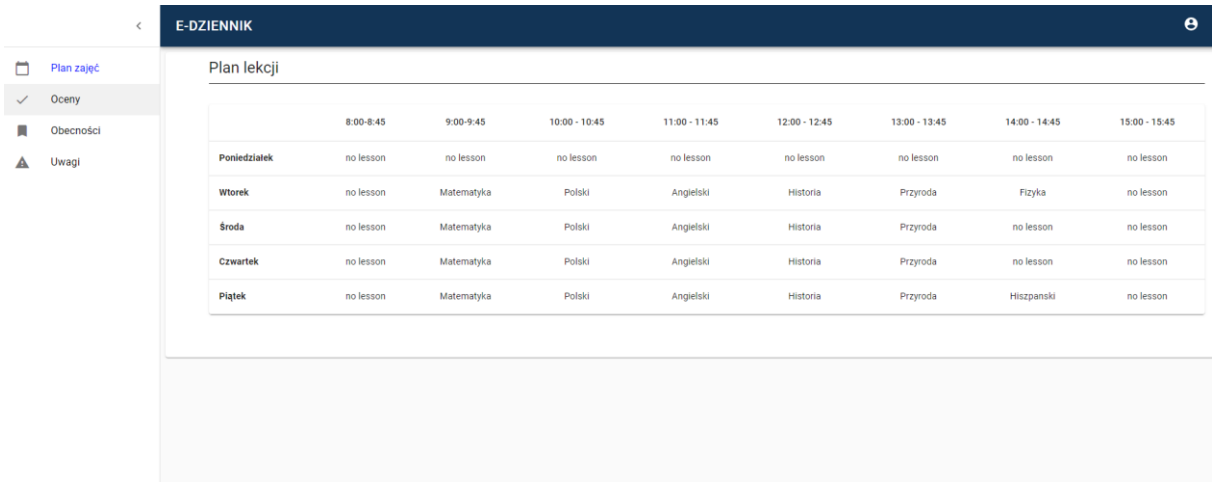
Omawiana częścią aplikacji jest stroną backendowa. Projekt jest napisany w .Net Core co umożliwia jego większą elastyczność i uniwersalność. Projekt ma możliwość we względnie prosty sposób być odpalony na serwerach linuxowych jednak jego działanie było testowane wyłącznie w Windows 10.

Aplikacja składa się z ustrukturowanej formy:
- Controllers
- DTO
- Migrations
- Model
- Repositorie
- Services
- ServicesCore
- Settings

Zdjęcia i przykład działania



Rys.1 Panel nauczyciela z oknem dodawania oceny



Rys. 2 Plan lekcji

Rys. 3 Panel ucznia z widokiem na wybór przedmiotu w oknie oceny