Wymagania do zadania:

- **Postman**. Można go pobrać ze strony: https://www.getpostman.com/
- **Visual Studio 2017** z zainstalowanym komponentem WEB
- **Pobranie przygotowanej solucji**

Cel zadania:

Celem tego zadania jest zapoznanie się ze sposobem projektowania oraz implementacji aplikacji REST'owych za pomocą framework'a Microsoft ASP.NET Core Web Api. Podczas wykonywania zadań stworzycie proste API służące do gromadzenia informacji o studentach, do której następnie jest możliwość stworzenia aplikacji klienckiej np. frontend'owej lub mobilnej.

Microsoft ASP.NET Web Api

Komunikacja w aplikacjach Web API odbywa się na zasadzie Klient-Serwer. Serwer udostępnia poszczególne metody, do których klient (np. aplikacja frontend'owa) może wykonać zapytanie o otrzymanie odpowiednich danych znajdujących się na serwerze.

Protokół HTTP udostępnia poszczególne metody m.in.: GET (odczytywanie), POST (dodawanie), PUT (edycja) oraz DELETE (usuwanie). Są to metody CRUD, które umożliwiają nam podstawowe operacje na danych.

Zwracają one odpowiednie statusy:

- 200 – Ok, wszystko się udało
- 204 – No Content, status oznaczający sukces metody
- 404 – Not Found, element nie został znaleziony
- 400 – Bad Request, zapytanie zostało sformułowane nieprawidłowo
- 500 – Server Internal Error, wystąpił nieobsłużony błąd

Koncepcja usług HTTP jako źródła danych jest definiowana po przez adres URL. Sam adres więc stanowi formę zapytania do aplikacji. Metody HTTP stanowią pewny mechanizm wysłania żądania do tej aplikacji.

Np. http://localhost:49499/api/students/3

Akcja oczywiście musi się wykonać przez metodę HTTP. Wszystkie informację o zapytaniu są zawarte w URL i raczej nie są to poufne dane. Na podstawie tego adresu możemy się domyślić, że zostanie wykonane zapytanie o dane studenta o identyfikatorze = 3.

Jeśli zapytanie do źródła danych jest bardziej złożone, możemy przesyłać dane wewnątrz danej metody(w ciele zapytania HTTP). Jest to wykorzystywane w metodzie POST oraz PUT. W ciele zapytania specyfikujemy odpowiednie parametry najczęściej w formacie JSON(Javascript Object Notation).

ZADANIA

W solucji Students został stworzony projekt aplikacji webowej Web API. Jest ona podłączona do bazy danych(umieszczonej na serwerze Politechniki) za pomocą biblioteki Entity Framework Core i podejściu Code First. Po wykonaniu pierwszego zadania w bazie danych powinna zaistnieć tabela reprezentująca dane studentów, w tym celu został stworzony model Student (folder Models) oraz klasa StudentContext(służąca do komunikacji z bazą danych) do skonfigurowania zawartości tworzonej bazy danych.

```
public virtual DbSet<Student> Students { get; set; }
```

Tworząc właściwość Students zostanie dodana tabela zawierająca wszystkich studentów.

Zadanie 1

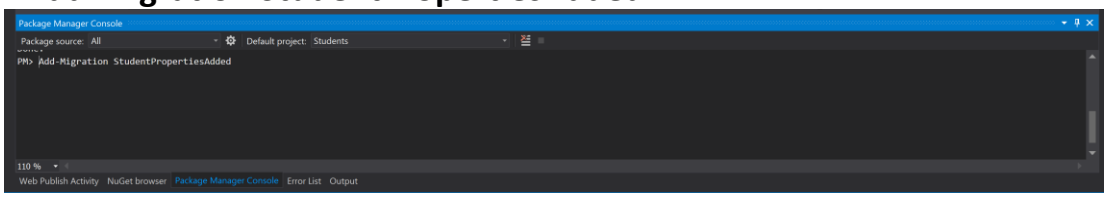
- Uzupełnij klasę Student o property: FirstName oraz LastName. Dodając atrybut [Required] nad propertę LastName, sprawiasz, że będzie ona wymagana.
- Przemapuj nazwę tabeli zawierającej informację na temat studentów, aby była ona zgodna z konwencją.
  W tym celu nadaj odpowiedni atrybut nad klasą Student:

```
[Table("nazwaGrupy_NrIndeksu_WebApi")]
```

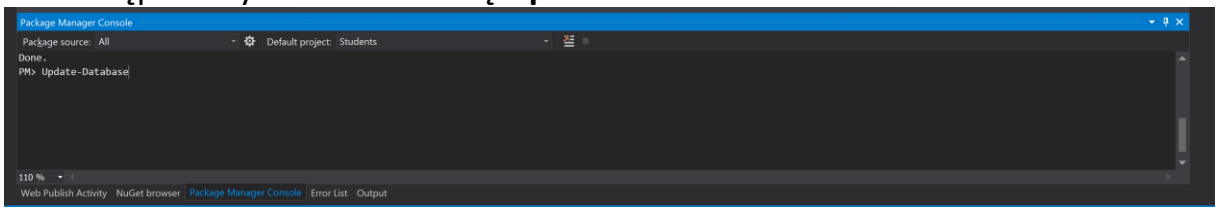
```
[Table("sr1115_228080_WebApi")]
public class Student
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    [Required]
    public string LastName { get; set; }
}
```

Aby zmiany zostały zaaplikowane do bazy danych należy dodać migrację. W oknie Package Manager Console należy wpisać komendę

Add-Migration StudentPropertiesAdded



A następnie wykonać komendę **Update-Database**



Upewnij się, że wybrane zmiany zostały zaaplikowane do bazy danych.

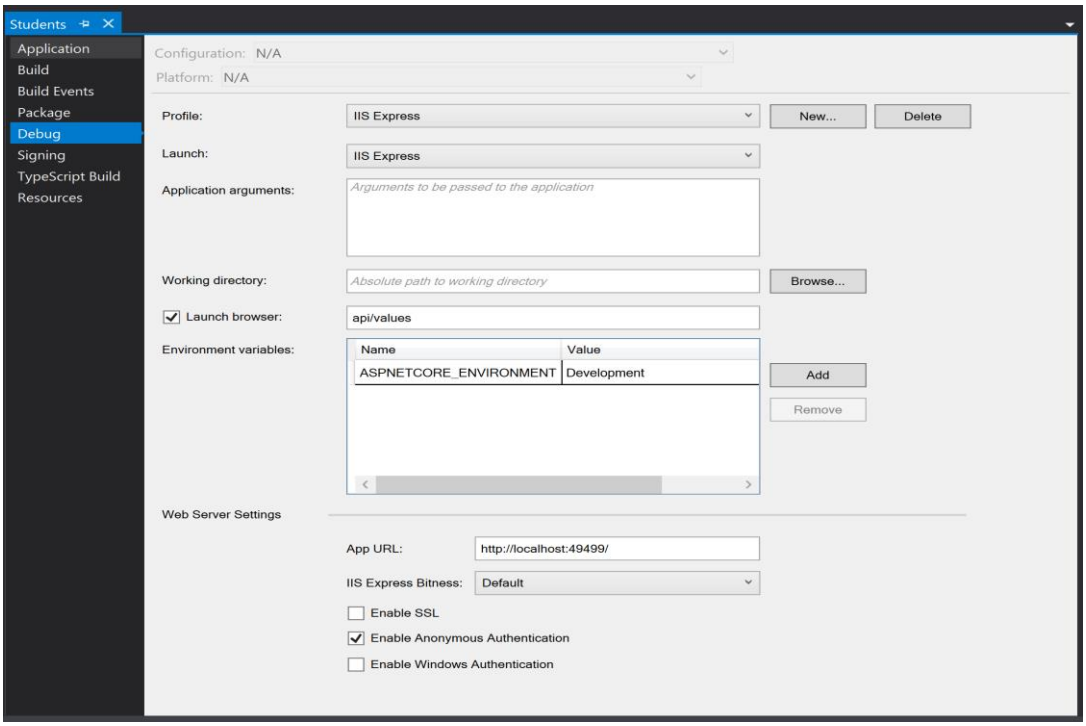
Zadania – metody http

W folderze Controllers została zdefiniowana klasa StudentsController.cs, to właśnie w niej stworzymy metody, które będzie udostępniał nasz serwer.
Nad klasą został dodany atrybut [Route()], określa on w jaki sposób możemy się dostać do naszej klasy z poziomu url'a.
Czyli w tym przypadku będzie to np.:

http://localhost:49499/api/students

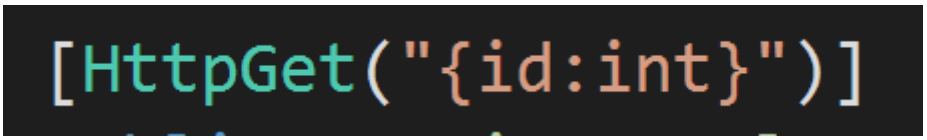
```
[Route("api/[controller]")]
public class StudentsController : Controller
{
    private readonly StudentContext _context;

    public StudentsController(StudentContext context)
    {
        _context = context;
    }
```

Dokładny url swojej aplikacji możesz sprawdzić klikając prawym klawiszem na nazwę projektu, wejść w opcje Properties (na samym dole), a następnie wybrać w wyświetlonym oknie opcje Debug.
Dokładny url twojej aplikacji znajduje się pod hasłem App URL.



Aby stworzyć metodę HTTP musi mieć ona nadany odpowiedni atrybut, jeśli jest to metoda POST należy dodać atrybut HttpPost. Dla pozostałych metod HTTP postępujemy podobnie tzn. definiujemy atrybut HttpGet, HttpPut oraz HttpDelete.
W nawiasach definiujemy Route metody, jeśli zdefiniujemy go jako pustego stringa będzie oznaczać to, że do metody dostaniemy się wpisując url kontrolera oraz zaznaczając, że (w tym wypadku) jest to metoda post

Jeśli chcemy wpisać do rout'a konkretny parametr, umieszczamy go w nawiasach klamrowych:



Oznacza to, że aby dostać się do tej metody należy wpisać url kontrolera, a następnie po ukośniku wybrany id

Np.: http://localhost:49499/api/students/3

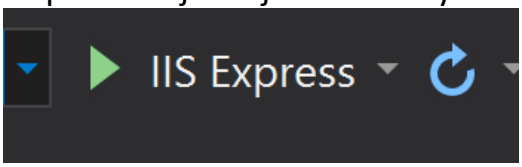
Należy pamiętać, aby metoda HTTP przyjmowała wtedy taki parametr jak jest zdefiniowany w routing'u.

Za pomocą _context.Students można dostać się do zawartości tabeli.

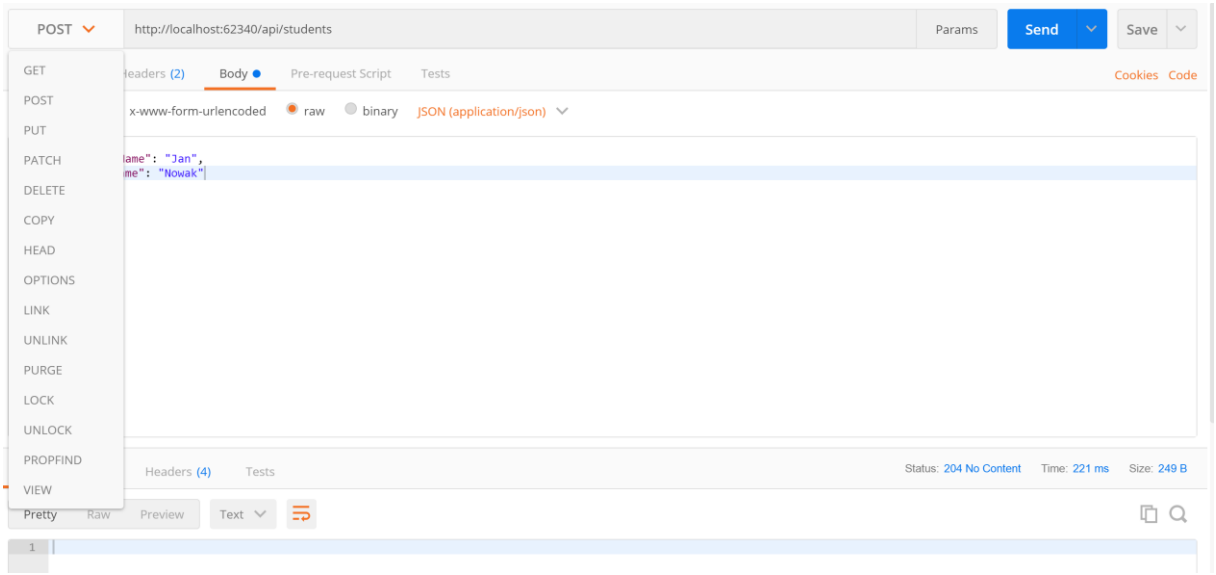
TESTOWANIE

Do testowania napisanych metod można użyć wcześniej pobranej aplikacji Postman.

W pierwszej kolejności należy uruchomić program:

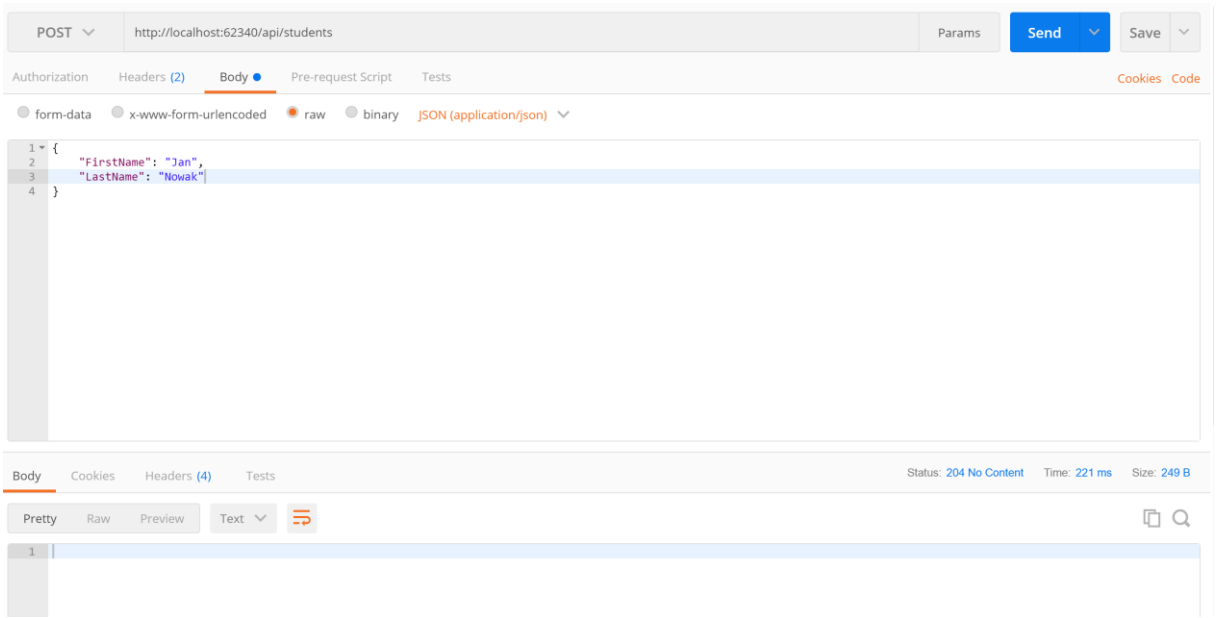


Następnie w Postmanie należy wpisać url wybranej metody



Z rozwijanego menu wybieramy rodzaj metody HTTP

W opcji Body w postaci JSON możemy wpisać obiekt, który chcemy przekazać w metodzie. Body jest dostępne dla metody PUT oraz POST



W dolnej części znajduje się odpowiedź serwera na żądanie – status oraz zwrócone dane.

Zadanie 2a

Stwórz metodę IActionResult Post([FromBody] Student studentToAdd){}

- Ustal poprawny routing metody, dostęp powinien być po wpisaniu url'a kontrolera
- Jeśli studentToAdd nie jest zgodny z modelem Student zwróć BadRequest() – status 400
  (Możemy w tym celu sprawdzić warunek ModelState.IsValid, aby uzyskać informację dlaczego nie jest prawidłowy obiekt możemy zwrócić BadRequest(ModelState))
- Jeśli student zostanie poprawnie dodany zwróć NoContent() – status 204
- [FromBody] oznacza, że obiekt student będzie przesyłany w ciele żadania http (Body w postmanie)

Zadanie 2b

Stwórz metodę IActionResult Get(int id){}

- Ustal poprawny routing metody, dostęp powinien być po wpisaniu url'a kontrolera oraz wybranego id
- Jeśli w bazie danych nie istnieje student o takim id zwróć NotFound() – status 404
- Jeśli istnieje zwróć Ok(studentFromDatabase) → otrzymamy status 200 oraz studenta

Zadanie 2c

Stwórz metodę IActionResult Get(){}

- Ustal poprawny routing metody, dostęp powinien być po wpisaniu url'a kontrolera
- Zwróć status 200 wraz z listą wszystkich studentów w bazie danych

Zadanie 2d

Stwórz metodę IActionResult Put(int id, [FromBody] Student studentToEdit){}

- Ustal poprawny routing metody, dostęp powinien być po wpisaniu url'a kontrolera oraz wybranego id
- Jeśli w bazie danych nie istnieje student o takim id zwróć NotFound – status 404
- Jeśli studentToEdit nie jest zgodny zgodny z modelem Student zwróć BadRequest() – status 400
- Jeśli został poddany edycji zwróć Ok(student)

Zadanie 2e

Stwórz metodę IActionResult Delete(int id){}

- Ustal poprawny routing metody, dostęp powinien być po wpisaniu url'a kontrolera oraz wybranego id
- Jeśli w bazie danych nie istnieje student o takim id zwróć NotFound – status 404
- Jeśli student został poprawnie usunięty zwróć NoContent()

Zadanie 3

Przetestuj dokładnie wszystkie napisane przez siebie metody za pomocą aplikacji Postman.

Przygotowali Justyna Setlak, Michał Kurpiński