

APBD - Ćwiczenia 3

pgago@pja.edu.pl

15 marca 2020

1 Zadanie 1 - instalacja Postman

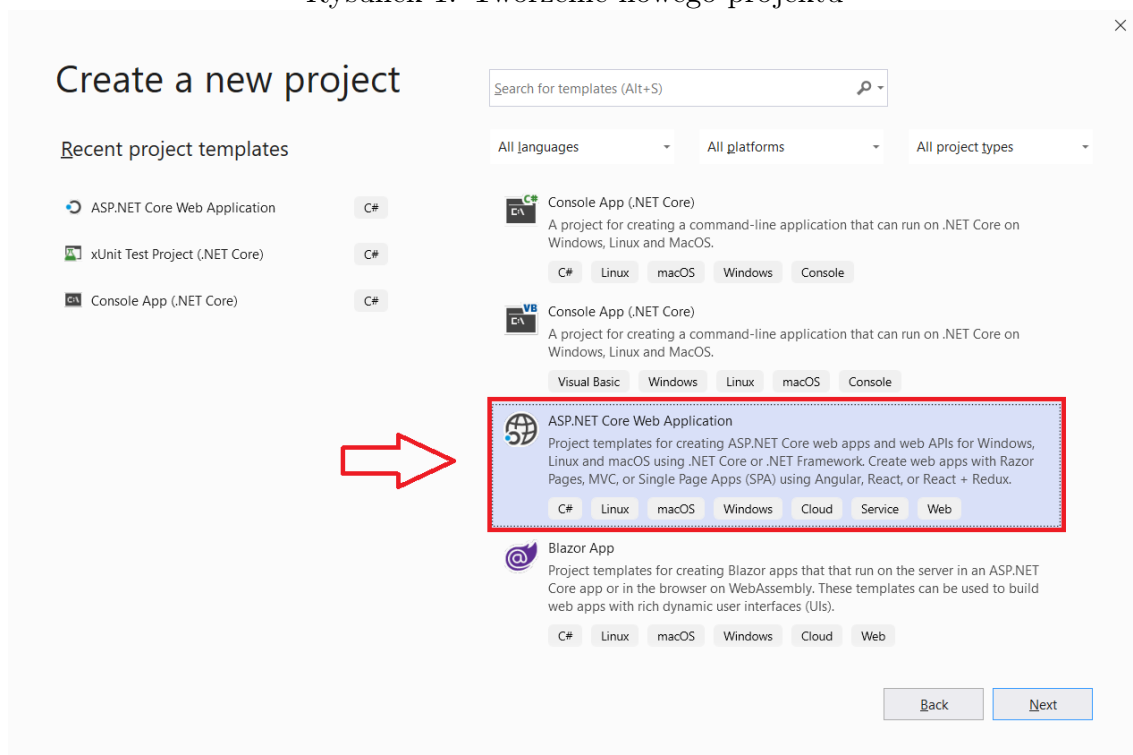
W niniejszych ćwiczeniach stworzymy nową aplikację typu Web API. Podczas testowania aplikacji webowej przyda nam się dodatkowa aplikacja o nazwie Postman. Aplikacja jest darmowa i istnieje zarówno w formie plugin'u do przeglądarki Chrome bądź osobnej aplikacji desktopowej. Proszę o jej zainstalowanie. Najwygodniej korzystać z wersji desktopowej. Adres URL:

<https://www.postman.com/downloads/>

2 Zadanie 2 - tworzenie nowego projektu

1. Proszę o stworzenie nowego repozytorium o nazwie Cw3.
2. Następnie proszę o stworzenie nowego projektu "ASP.NET Core Web Application". Z dostępnych szablonów proszę wybrać API".

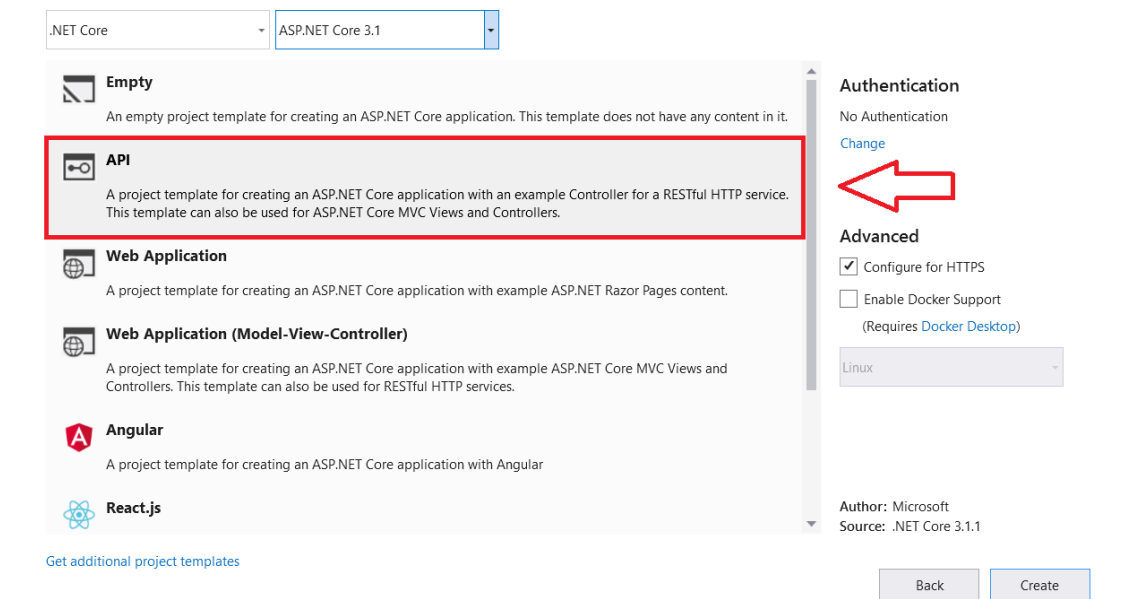
Rysunek 1: Tworzenie nowego projektu



Rysunek 2: Wybór odpowiedniego szablonu

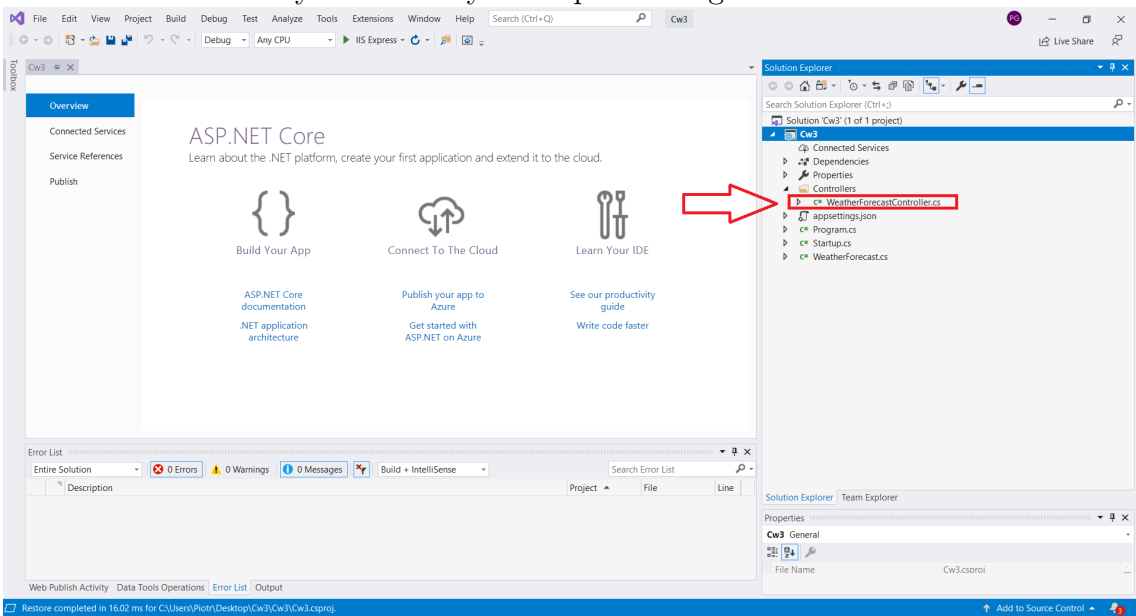
x

Create a new ASP.NET Core web application



3. Po stworzeniu nowego projektu proszę o usunięcie pliku Controllers/WeatherForecastController.cs

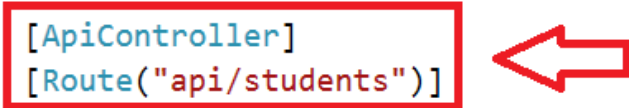
Rysunek 3: Wybór odpowiedniego szablonu



3 Zadanie 3 - Dodanie kontrolera

1. W tym zadaniu dodamy swój pierwszy kontroler. Tak jak pamiętamy, kontroler przetwarza żądania użytkowników i zwraca oczekiwany rezultat.
2. Do folderu Controllers dodajemy nowy kontroler o nazwie `StudentsController.cs`.
3. Pamiętajmy, że myśląc w kontekście API zgodnego z podejściem REST - myślimy o zasobach, które nasze API udostępnia za pośrednictwem protokołu HTTP. Nazwy kontrolerów odzwierciedlają zazwyczaj udostępniane zasoby.
4. Następnie proszę zmodyfikować kod, tak aby nasza klasa była przygotowana do realizowania żądań typu HTTP REST.
5. Nad nazwą klasy dodajemy dwa atrybuty. Pierwszy z nich `[ApiController]` oznacza nasz kontroler jako API. Dzięki temu będziemy mogli skorzystać z kilku wbudowanych mechanizmów związanych z walidacją, które przydadzą się później. Atrybut `[Route]` pozwala określić adres jaki będzie identyfikował nasz kontroler. Wszystkie żądania wysyłane na ten adres będą automatycznie przekierowywane do niniejszego kontrolera.

```
7 namespace Cw3.Controllers
8 {
9     [ApiController]
10    [Route("api/students")]
11    public class StudentsController : Controller
12    {
13        public IActionResult Index()
14        {
15            return View();
16        }
17    }
18 }
```



6. Następnie zmieniamy klasę po której dziedziczymy z `Controller` na `ControllerBase`. Klasa `ControllerBase` zawiera wiele przydatnych metod pomocniczych z których będziemy korzystać.

```

namespace Cw3.Controllers
{
    [ApiController]
    [Route("api/students")]
    0 references
    public class StudentsController : ControllerBase
    {
        0 references
        public IActionResult Index()
        {
            return View();
        }
    }
}

```

7. Metoda Index() przestała się kompilować. W tym momencie dodajemy pierwszą metodę, która będzie zwracała dane. Modyfikujemy metodę Index() w następujący sposób.

```

[ApiController]
[Route("api/students")]
0 references
public class StudentsController : ControllerBase
{
    [HttpGet]
    0 references
    public string GetStudent()
    {
        return "Kowalski, Malewski, Andrzejewski";
    }
}

```

8. Następnie spróbujemy uruchomić aplikację. Następnie proszę spróbować wpisać adres "https://localhost:44316/api/students". Proszę wykonać takie same żądanie HTTP z pomocą aplikacji Postman. Pamiętajcie, że w Waszym wypadku adres końcówki może być inny.

←

→

↺

🔒

localhost:44316/api/students

Kowalski, Malewski, Andrzejewski

Untitled Request

Comments 0

GET

https://localhost:44316/api/students

Send

Save

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

Cookies

Code

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body

Cookies

Headers (5)

Test Results

Status: 200 OK

Time: 26ms

Size: 208 B

Save Response

Pretty

Raw

Preview

Visualize

Text

1

Kowalski, Malewski, Andrzejewski

Bootcamp

Build

Browse

4 Zadanie 4 - przekazywanie parametru jako URL segment

1. W poprzednim zadaniu udało nam się uruchomić aplikację webową i wykonać żądania HTTP GET. Żądanie przesłane na adres `/api/students` zostało przekierowane do kontrolera `StudentsController` i metody `GetStudents`.
2. Wybierając odpowiedni adres URL i metodę HTTP (np. GET, PUT, POST, DELETE, PATCH) możemy wybrać jaką metoda zostanie uruchomiona. Zazwyczaj jednak musimy przesłać do serwera pewne dodatkowe dane. Mamy na to kilka sposobów. Pierwszy z nich to użycie segmentu URL.
3. Segment URL wybieramy wtedy, kiedy przesyłane informacje związane są z identyfikacją jakiegoś zasobu (podobnie jak id w bazie danych). Np. może to być id studenta z bazy danych. Kiepskim byłoby natomiast wykorzystanie segmentu URL do przesyłania np. informacji o tym jak chcemy posortować dane.
4. Proszę dodać nową metodę, która pozwoli nam na przekazanie parametru. Jak widać w atrybucie `HttpGet` dodaliśmy parametr `id`. Ponadto dodana metoda ma również parametr o tej samej nazwie. Następnie podczas wykonywania żądania parametr z odpowiedniej części URL będzie automatycznie przekazany do odpowiedniej metody.

Rysunek 4: Uruchomienie aplikacji

```
[HttpGet("{id}")]
0 references
public IActionResult GetStudent(int id)
{
    if (id == 1)
    {
        return Ok("Kowalski");
    } else if (id == 2)
    {
        return Ok("Malewski");
    }

    return NotFound("Nie znaleziono studenta");
}
```

5. Ponadto wykorzystujemy tutaj interfejs `IActionResult`. Jest to interfejs wykorzystywany do zwrotu różnego rodzaju danych - zarówno tekstu, plików JSON, XML i innych. W naszym wypadku wykorzystujemy dwie dodatkowe metody `Ok()` i `NotFound()`. Obie metody pochodzą z nadklasy `ControllerBase`. Pozwalają nam w łatwy sposób zwrócić

dane i opakować"je w odpowiednia odpowiedź i kod HTTP (np. 200 OK lub 404 Not Found).

6. Następnie proszę przetestować końcówkę z pomocą aplikacji Postman. Proszę spróbować przekazać dane w różny sposób i sprawdzić jaki będzie efekt.

Przetestowanie końcówki

Untitled Request Comments 0

GET

https://localhost:44316/api/students/2

Send

Save

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies Code

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (5) Test Results Status: 200 OK Time: 4.61s Size: 184 B Save Response

Pretty Raw Preview Visualize Text

1 Malewski

5 Zadanie 5 - przekazywane danych z pomocą QueryString

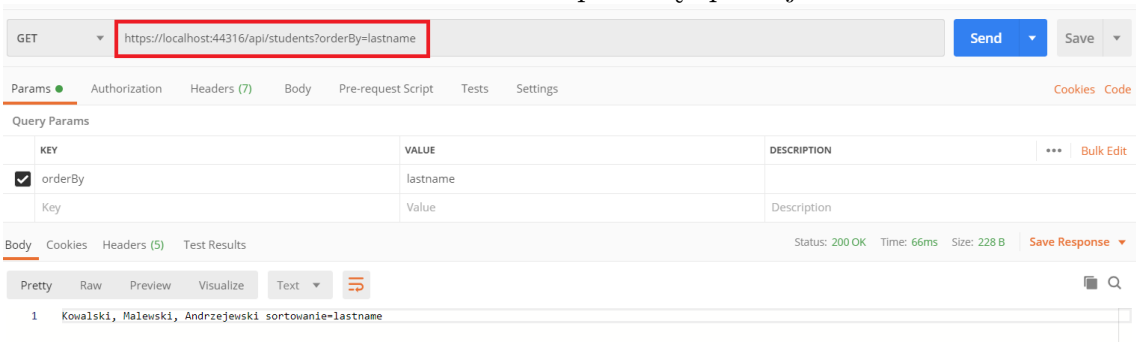
1. W tym ćwiczeniu zmodyfikujemy nieco metodę, którą napisaliśmy w poprzednim zadaniu.
2. Chcemy przekazać parametr, który pozwoliłby nam określić w jaki sposób dane powinny zostać posortowane. W tym wypadku nie pobieramy jeszcze danych z bazy danych, ale na razie przyglądamy się jedynie temu w jaki sposób przekazać niezbędne dane do serwera.
3. Zmodyfikuj pierwszą metodę dodaną do StudentsController tak, aby wyglądała jak ta poniżej.

Modyfikacja metody pozwalającej na przekazanie danych w QueryString

```
[HttpGet]
0 references
public string GetStudents(string orderBy)
{
    return $"Kowalski, Malewski, Andrzejewski sortowanie={orderBy}";
}
```

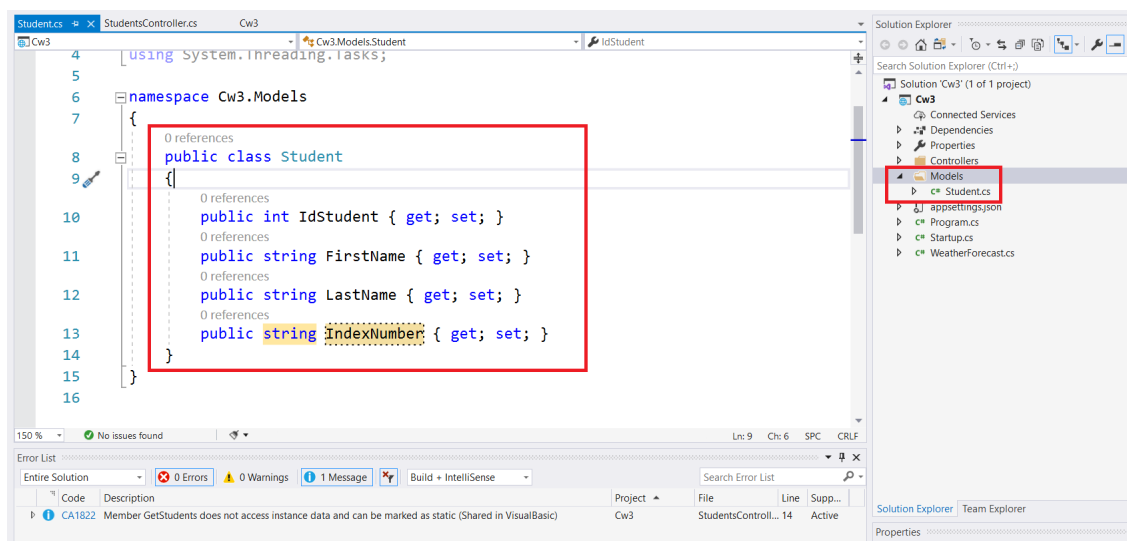
4. Następnie proszę uruchomić aplikację i sprawdzić czy jesteśmy w stanie przekazać dane z pomocą QueryString'a.

Przetestowanie końcówki z pomocą aplikacji Postman



6 Zadanie 6 - przekazywanie danych w ciele żądania

1. W kolejny zadaniu przygotuje, u metodę, która ma odpowiadać na żądania HTTP typu POST. Tego typu metody służą zazwyczaj do dodawania nowych elementów do bazy danych. Przygotujemy metodę, która będzie nam służyć do dodawania do bazy danych nowych studentów. Oczywiście na razie nie korzystamy z realnej bazy danych.
2. Najpierw dodajmy do projektu folder o nazwie "Models". Wewnątrz niego dodajmy plik o nazwie "Student.cs". Będzie to klasa, która modeluje nasze dane. Za jej pomocą prześlemy dane do kontrolera.



3. Następnie proszę dodać kolejną metodę do kontrolera.

```
[HttpPost]
0 references
public IActionResult CreateStudent(Student student)
{
    //... add to database
    //... generating index number
    student.IndexNumber = $"s{new Random().Next(1, 20000)}";
    return Ok(student);
}
```

4. Jak widać metoda nie wykonuje zbyt dużo. Zwróćmy uwagę na atrybut [HttpPost] i parametr "Student student". Za każdym razem kiedy jako parametr wykorzystywany jest typ złożony (np. klasa Student) dane są deserializowane i pobierane z ciała żądania HTTP.
5. Następnie proszę uruchomić aplikację i przetestować z pomocą aplikacji Postman.

Untitled Request

Comments 0

POST

https://localhost:44316/api/students

Send

Save

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

Cookies

Code

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

Beautify

1 {

2 "idStudent": 1,

3 "firstname": "Jan",

4 "lastname": "Kowalski"

5 }

Body

Cookies

Headers (5)

Test Results

Status: 200 OK

Time: 69ms

Size: 259 B

Save Response

Pretty

Raw

Preview

Visualize

JSON

1 {

2 "idStudent": 1,

3 "firstname": "Jan",

4 "lastname": "Kowalski",

5 "indexNumber": "s9785"

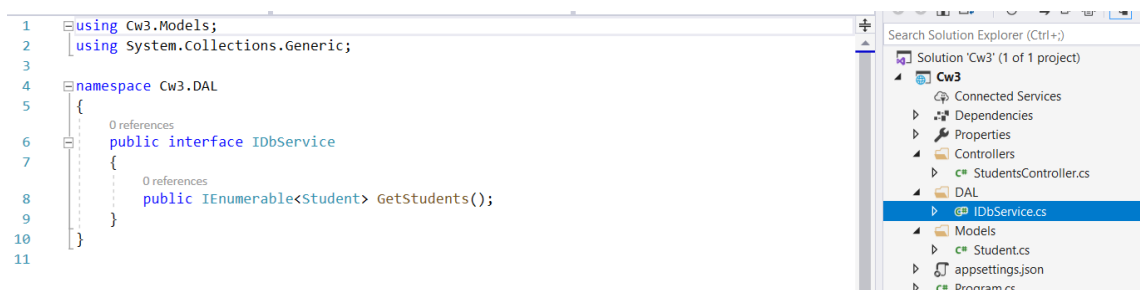
6 }

7 Zadanie 7 - dodanie dodatkowych metod

1. Proszę samodzielnie dodać metody odpowiadające na żądania typu PUT i DELETE.
2. W każdej z tych metod proszę zwrócić kod 200 i wiadomość Aktualizacja dokończona-
lub "Usuwanie ukończone".
3. Obie metody powinny przyjmować jako parametr id przekazywane jako segment adresu URL.
4. Przetestuj działanie dodanych metod w aplikacji Postman.

8 Zadanie 8 - dodanie sztucznej"bazy danych

1. W ostatnim ćwiczeniu spróbujemy dodać sztuczną bazę danych. Często podczas pracy nad aplikacją chcielibyśmy się zająć implementacją kolejnych elementów nie czekając na administratora bazy danych.
2. W takim wypadku warto zamockować"bazę danych, którą później podmienimy na "prawdziwą"bez większych problemów wykorzystując kontener zależności.
3. Na początek dodajemy folder DAL (ang. Data Access Layer) do projektu.
4. Wewnątrz umieszczamy dwa pliki. Pierwszy z nich będzie reprezentować interfejs z jedną metodą zwracającą kolekcję studentów. Zwróćcie uwagę, że jako typ zwracany wybrałem interfejs IEnumerable<T>. Staram się wykorzystywać abstrakcję.



5. Następnie w tym samym folderze dodajemy klasę MockDbService.cs. Wewnątrz niej dodajemy kolekcję zawierającą kilku studentów.



6. Jak widać klasa MockDbService zwraca przygotowaną wcześniej kolekcję. Interfejs posłuży nam jako "klej" lub warstwa abstrakcji, która pozwala nam wykorzystać klasę MockDbService w dowolnym kontrolerze. Wykorzystanie interfejsu pozwoli nam również łatwo wymienić implementację interfejsu na inną w przyszłości.
7. Moglibyśmy skorzystać teraz z klasy MockDbService bezpośrednio w kontrolerze StudentsController. Nie chcemy jednak tworzyć silnej zależności między klasą StudentsController i MockDbService.
8. Najpierw zmodyfikujemy klasę StudentsController w następujący sposób.

```

[ApiController]
[Route("api/students")]
1 reference
public class StudentsController : ControllerBase
{
    private readonly IDbService _dbService;

    0 references
    public StudentsController(IDbService dbService)
    {
        _dbService = dbService;
    }
}

[HttpGet]
0 references
public IActionResult GetStudents(string orderBy)
{
    return Ok(_dbService.GetStudents());
}

```

9. Jak widać przekazujemy parametr w konstruktorze. Typ parametru to IDbService.
10. Następnie musimy zarejestrować nasz serwis. Mechanizm ten opiszemy jeszcze w czasie najbliższego wykładu. Modyfikujemy metodę ConfigureServices w klasie Startup.cs.

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton<IDbService, MockDbService>();
    services.AddControllers();
}

```

11. Następnie proszę uruchomić aplikację i z pomocą Postman'a wykonać żądania HTTP GET na adres `api/students`.

