Rzeszów, 04.07.2017



Instrukcja laboratoryjna

Budowa aplikacji internetowej z użyciem   
ASP.NET Core.

Przygotował: Paweł Płaza  
3 EF-DI

1. Cel ćwiczenia

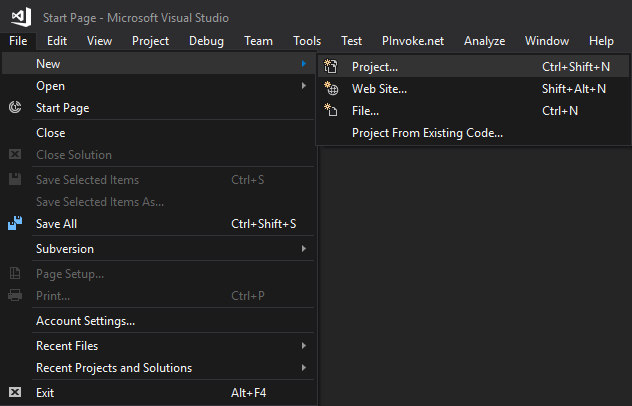
Efektem końcowym ćwiczenia będzie prosta aplikacja internetowa prezentująca listę motocykli i umożliwiająca sprawdzenie szczegółowych danych każdego z nich.

Elementy listy przechowywane będą w bazie danych, z którą będziemy komunikować się używając Entity Framework.   
  
Widok aplikacji przygotowany zostanie przy pomocy frameworka Angular 2.  
Kod widoku będziemy pisać używając języka TypeScript, który jest nadzbiorem JavaScript, umożliwia on m.in. używanie klas oraz interfejsów.  
Aby nadać aplikacji bardziej przyjazny wygląd przy niedużym nakładzie pracy użyjemy frameworka Bootstrap 4.

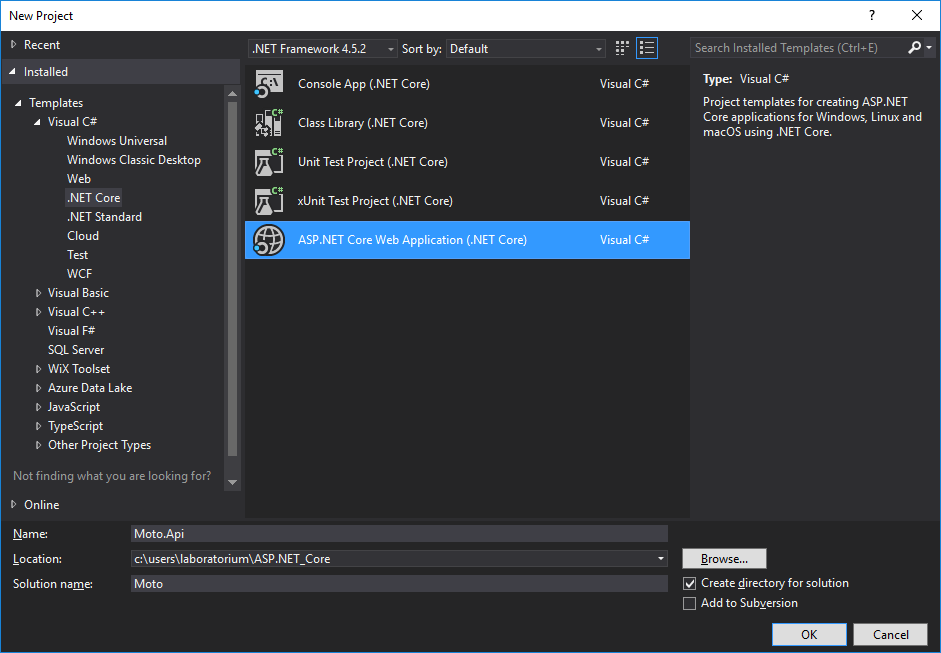
1. Przygotowanie API

Uruchamiamy środowisko Visual Studio 2017 Community.

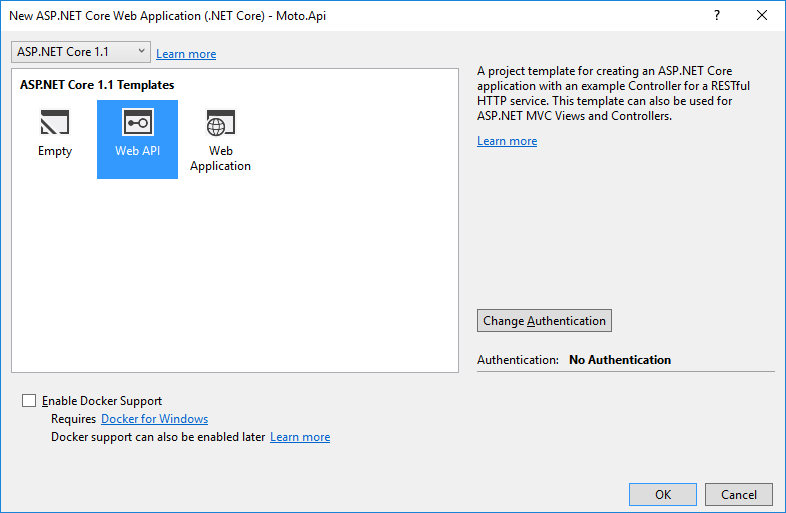
Tworzymy nowy projekt poprzez **File** -> **New** -> **Project**



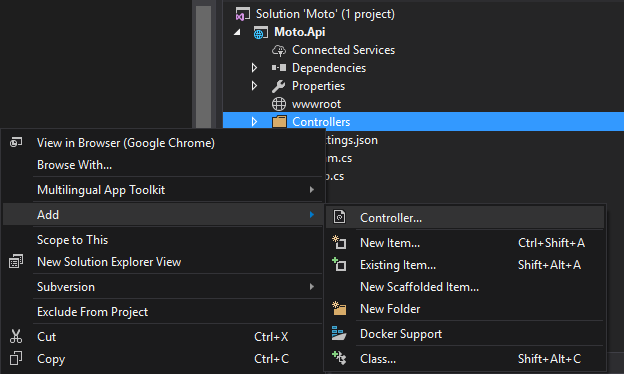
Jako typ projektu wybieramy webową aplikację ASP.NET Core  
(**Installed** -> **Templates** -> **Visual C#** -> **.NET Core** -> **ASP.NET Core Web Appliaction (.NET Core)**)  
Nazywamy nasz projekt i solucję jak na poniższym obrazku.



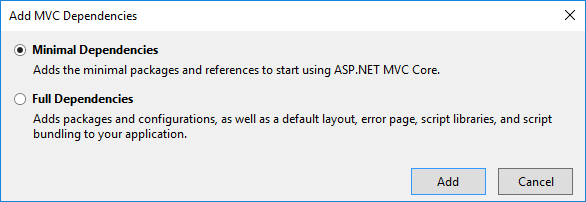
Wybieramy **Web API**



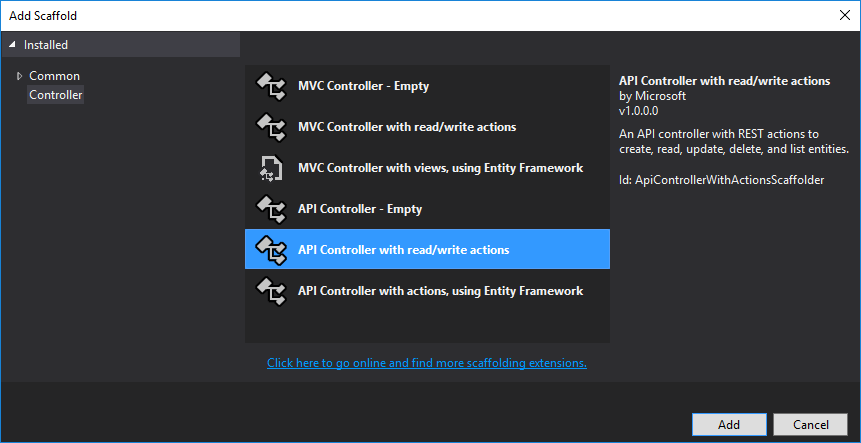
Utworzymy teraz kontroler dla motocykli. Będziemy mogli dzięki niemu pobierać dane na temat poszczególnego modelu, pobrać całą listę motocykli, edytować poszczególne wpisy itd.   
Klikamy **PPM** na folder **Controllers,** następnie **Add** -> **Controller.**



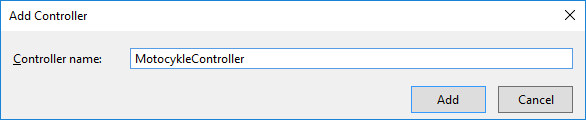
Jeśli pojawi się okno jak poniżej, wybieramy **Minimal Dependencies** i klikamy **Add.**



Następnie w oknie jak poniżej wybieramy **API Controller with read/write actions.**Jeśli okno się nie pojawiło, ponownie próbujemy dodać kontroler.

****

Nadajemy kontrolerowi nazwę i klikamy **Add.**

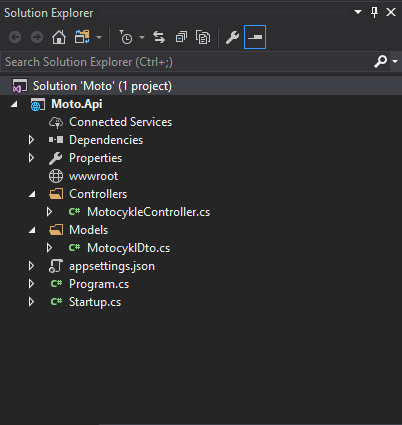


Teraz usuwamy z folderu **Controllers** plik **ValuesController.cs**, nie będzie on nam potrzebny (PPM -> Delete).

Następnie utworzymy folder, w którym przechowywać będziemy odpowiednie modele. W tym przypadku będzie to model motocykla. Klikamy **PPM** na nazwę naszego projektu (Moto.Api) -> **Add -> New Folder.** Nadajemy mu nazwę *Models.*

Teraz dodajemy do folderu **Models** nową klasę o nazwie **MotocyklDto.**(DTO – Data Transfer Object). Robimy to poprzez kliknięcie **PPM** na folder **Models -> Add -> Class.**

Struktura naszej aplikacji powinna wyglądać w tej chwili następująco:



Zajmiemy się teraz odpowiednim zaprojektowaniem klasy **MotocyklDto**.  
Dodamy kilka pól opisujących motocykl, klasę można rozbudować we własnym zakresie.   
  
**public** class MotocyklDto

**{**

**public** int Id **{** get**;** set**;** **}**

**public** string Marka **{** get**;** set**;** **}**

**public** string Model **{** get**;** set**;** **}**

**public** string Typ **{** get**;** set**;** **}**

**public** double PojemnoscSkokowa **{** get**;** set**;** **}**

**public** string Chlodzenie **{** get**;** set**;** **}**

**public** double Masa **{** get**;** set**;** **}**

**public** double PojemnoscZbiornikaPaliwa **{** get**;** set**;** **}**

**public** int PredkoscMaksymalna **{** get**;** set**;** **}**

**}**

Teraz do folderu **Models** dodamy klasę **MotocyklForCreationDto.** Będziemy jej używać do dodawania nowych modeli motocykli do bazy danych.  
Jej struktura będzie wyglądała nieco inaczej. Klasa ta nie będzie zawierać pola Id, ponieważ podczas tworzenia użytkownik nie podaje jego wartości. Dodatkowo reszta pól będzie posiadała atrybuty, których nazwy przedstawiają ich przeznaczenie.

Aby nasze środowisko rozpoznało atrybuty musimy dodać odpowiedni using.  
Możemy to zrobić poprzez kliknięcie **LPM** na nazwę atrybutu, a następnie dzięki kombinacji klawiszy **Ctrl** + **.** pokażemy podręczne menu, wybieramy z niego   
**Using System.ComponentModels.DataAnnotations.**

Kod klasy wygląda następująco:

**public** class MotocyklForCreationDto

**{**

**[**Required**]**

**[**MaxLength**(**50**,** ErrorMessage **=** "Za długa nazwa marki."**)]**

**public** string Marka **{** get**;** set**;** **}**

**[**Required**]**

**[**MaxLength**(**50**,** ErrorMessage **=** "Za długa nazwa modelu."**)]**

**public** string Model **{** get**;** set**;** **}**

**[**MaxLength**(**50**,** ErrorMessage **=** "Za długa nazwa typu."**)]**

**public** string Typ **{** get**;** set**;** **}**

**public** double PojemnoscSkokowa **{** get**;** set**;** **}**

**public** string Chlodzenie **{** get**;** set**;** **}**

**public** double Masa **{** get**;** set**;** **}**

**public** double PojemnoscZbiornikaPaliwa **{** get**;** set**;** **}**

**public** int PredkoscMaksymalna **{** get**;** set**;** **}**

**}**

Kolejnym krokiem jest utworzenie folderu dla encji, nazwiemy go **Entities.**

Umieścimy w nim reprezentacje tabeli bazy danych jako klasy **Motocykl.**

W tym celu tworzymy nową klasę w tym folderze o właśnie takiej nazwie – **Motocykl.**Po wklejeniu kodu klasy musimy pamiętać o dodaniu odpowiednich usingów tak jak w przykładzie powyżej.

**public** class Motocykl

**{**

**[**Key**]**

**[**DatabaseGenerated**(**DatabaseGeneratedOption**.**Identity**)]**

**public** int Id **{** get**;** set**;** **}**

**[**Required**]**

**[**MaxLength**(**50**)]**

**public** string Marka **{** get**;** set**;** **}**

**[**Required**]**

**[**MaxLength**(**50**)]**

**public** string Model **{** get**;** set**;** **}**

**[**MaxLength**(**50**)]**

**public** string Typ **{** get**;** set**;** **}**

**public** double PojemnoscSkokowa **{** get**;** set**;** **}**

**public** string Chlodzenie **{** get**;** set**;** **}**

**public** double Masa **{** get**;** set**;** **}**

**public** double PojemnoscZbiornikaPaliwa **{** get**;** set**;** **}**

**public** int PredkoscMaksymalna **{** get**;** set**;** **}**

**}**

Teraz musimy utworzyć klasę, która będzie odpowiedzialna za połącznie z bazą danych. W tym celu w folderze **Entities** tworzymy klasę o nazwie **MotocyklContext.**

Oto kod klasy:

**public** class MotocyklContext **:** DbContext

**{**

**public** MotocyklContext

**(**DbContextOptions**<**MotocyklContext**>** options**)** **:** **base(**options**)**

**{**

Database**.**EnsureCreated**();**

**}**

**public** DbSet**<**Motocykl**>** Motocykle **{** get**;** set**;** **}**

**}**

Dodajemy odpowiednie usingi i przechodzimy do następnego kroku, czyli dodania naszej klasy do serwisów. Przechodzimy zatem do klasy **Startup**, i uzupełniamy metodę **ConfigureServices** tak, by wyglądała następująco:

**public** void ConfigureServices**(**IServiceCollection services**)**

**{**

// Add framework services.

services**.**AddMvc**();**

var connectionString **=** @"Server=(localdb)\MSSQLLocalDB;Database=MotoDB;Trusted\_Connection=True;"**;**

services**.**AddDbContext**<**MotocyklContext**>(**options **=>**

**{**

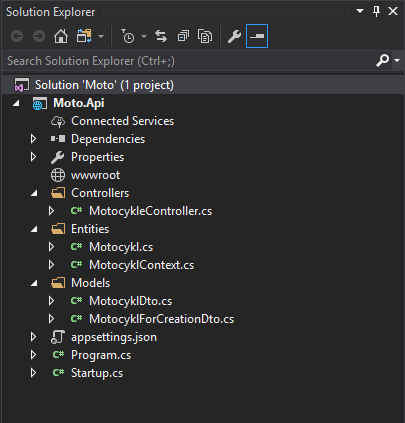
options**.**UseSqlServer**(**connectionString**);**

**});**

**}**

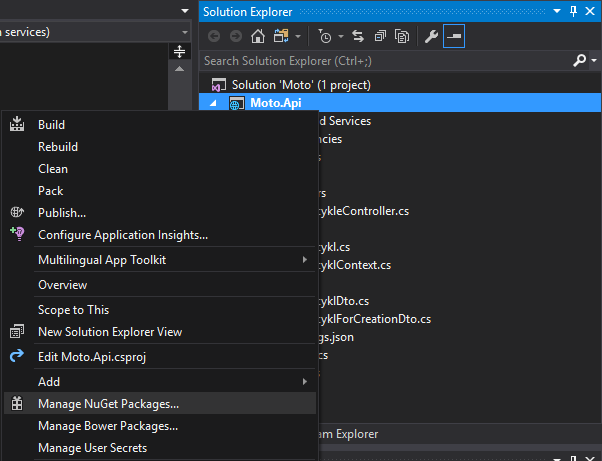
ConnectionString może różnić się w zależności od naszej bazy danych, jeżeli korzystamy z bazy danych w środowisku Visual Studio – nie musimy nic zmieniać.

W chwili obecnej struktura naszego projektu powinna wyglądać następująco:

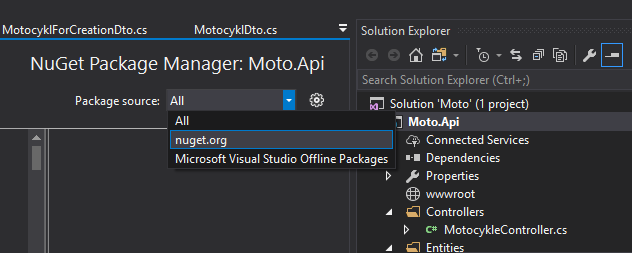


Musimy uzupełnić nasz projekt o migracje bazy danych. Migracje tworzymy po to, by podczas rozbudowy projektu móc zapisywać sposób w jaki powinna zostać utworzona baza danych lub edytowana do najnowszej wersji.

Aby to zrobić, najpierw musimy dodać odpowiednią paczkę NuGet.  
W tym celu klikamy **PPM** na naszym projekcie i wybieramy pozycję **Manage NuGet Packages…**

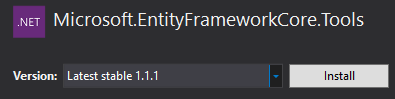


Następnie zmieniamy źródło na **nuget.org:**

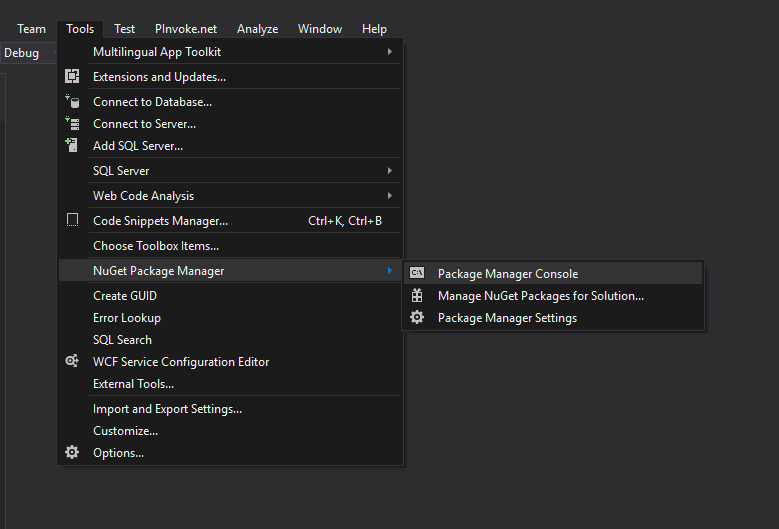


Przechodzimy na zakładke **Browse** i wpisujemy: **Microsoft.EntityFrameworkCore.Tools**

Instalujemy znalezioną paczkę:



Następnie uruchamiamy **Package Manager Console**:



W oknie konsoli wpisujemy komendę:  
Add-Migration MotocykleDBInitialMigration

Po jej użyciu utworzony zostanie folder o nazwie **Migrations**, który zawiera migrację naszej obecnej wersji bazy danych.

Mając przygotowaną migrację możemy zmienić kod konstruktora klasy **MotocyklContext**:

Zamieniamy Database.EnsureCreated(); na Database.Migrate();

Dzięki temu przy każdorazowym uruchomieniu serwera sprawdzona zostanie nasza baza danych pod względem aktualności.

Teraz będziemy potrzebowali serwisu, który udostępni nam dane z bazy danych.

W tym celu najpierw tworzymy folder o nazwie **Services,** a następnie dodajemy do niego interfejs **IMotocykleRepository.** Kod interfejsu jest następujący:

**public** **interface** IMotocykleRepository

**{**

IEnumerable**<**MotocyklDto**>** GetMotorcycles**();**

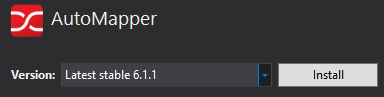
MotocyklDto GetMotorcycle**(**int id**);**

void AddMotorcycle**(**MotocyklForCreationDto motocykl**);**

void DeleteMotorcycle**(**Motocykl motorcycle**);**

**}**

Musimy dodać jeszcze jedną paczkę NuGet po to, aby móc mapować obiekty encji na obiekty DTO.   
Instalujemy paczkę o nazwie **AutoMapper.**



Kolejnym krokiem jest stworzenie mapy. Przechodzimy do metody **Configure** w klasie **Startup**, ma ona wyglądać następująco:

**public** void Configure**(**IApplicationBuilder app**,** IHostingEnvironment env**,** ILoggerFactory loggerFactory**)**

**{**

loggerFactory**.**AddConsole**(**Configuration**.**GetSection**(**"Logging"**));**

loggerFactory**.**AddDebug**();**

AutoMapper**.**Mapper**.**Initialize**(**config **=>**

**{**

config**.**CreateMap**<**Motocykl**,** MotocyklDto**>();**

config**.**CreateMap**<**MotocyklDto**,** Motocykl**>();**

config**.**CreateMap**<**MotocyklForCreationDto**,** Motocykl**>();**

**});**

app**.**UseMvc**();**

**}**

Następnie w folderze **Services** tworzymy klasę o nazwie **MotocykleRepository**, która będzie implementować interfejs **IMotocykleRepository**. Oto kod klasy:

**public** class MotocykleRepository **:** IMotocykleRepository

**{**

**private** MotocyklContext \_context**;**

**public** MotocykleRepository**(**MotocyklContext context**)**

**{**

\_context **=** context**;**

**}**

**public** void AddMotorcycle**(**MotocyklForCreationDto motocykl**)**

**{**

var entity **=** AutoMapper**.**Mapper**.**Map**<**Motocykl**>(**motocykl**);**

\_context**.**Motocykle**.**Add**(**entity**);**

\_context**.**SaveChanges**();**

**}**

**public** void DeleteMotorcycle**(**Motocykl motorcycle**)**

**{**

\_context**.**Motocykle**.**Remove**(**motorcycle**);**

\_context**.**SaveChanges**();**

**}**

**public** MotocyklDto GetMotorcycle**(**int id**)**

**{**

var entity **=** \_context**.**Motocykle**.**Where**(**value **=>** value**.**Id **==** id**).**FirstOrDefault**();**

**return** AutoMapper**.**Mapper**.**Map**<**MotocyklDto**>(**entity**);**

**}**

**public** IEnumerable**<**MotocyklDto**>** GetMotorcycles**()**

**{**

var entities **=** \_context**.**Motocykle**.**OrderBy**(**value **=>** value**.**Marka**).**ToList**();**

var dtos **=** **new** List**<**MotocyklDto**>();**

**foreach** **(**var element **in** entities**)**

**{**

dtos**.**Add**(**AutoMapper**.**Mapper**.**Map**<**MotocyklDto**>(**element**));**

**}**

**return** dtos**;**

**}**

**}**

Aby móc korzystać z naszego repozytorium, musimy je zarejestrować w klasie **Startup.** W metodzie **ConfigureServices** dodajemy linijkę:  
services.AddScoped<IMotocykleRepository, MotocykleRepository>();

AddScoped oznacza tworzenie instancji repozytorium przy każdorazowym jego wywołaniu.

Teraz, kiedy mamy już zaimplementowane repozytorium, czas zająć się kontrolerem.

Przechodzimy więc do klasy **MotocykleController.** Kontroler zaimplementujemy w bardzo prosty sposób, bez obsługi błędów. Oto jego kod:

**[**Produces**(**"application/json"**)]**

**[**Route**(**"api/Motocykle"**)]**

**public** class MotocykleController **:** Controller

**{**

**private** IMotocykleRepository \_motocykleRepository**;**

**public** MotocykleController**(**IMotocykleRepository motocykleRepository**)**

**{**

\_motocykleRepository **=** motocykleRepository**;**

**}**

// GET: api/Motocykle

**[**HttpGet**]**

**public** IEnumerable**<**MotocyklDto**>** Get**()**

**{**

**return** \_motocykleRepository**.**GetMotorcycles**();**

**}**

// GET: api/Motocykle/5

**[**HttpGet**(**"{id}"**,** Name **=** "Get"**)]**

**public** MotocyklDto Get**(**int id**)**

**{**

**return** \_motocykleRepository**.**GetMotorcycle**(**id**);**

**}**

// POST: api/Motocykle

**[**HttpPost**]**

**public** void Post**([**FromBody**]**MotocyklForCreationDto value**)**

**{**

\_motocykleRepository**.**AddMotorcycle**(**value**);**

**}**

// DELETE: api/ApiWithActions/5

**[**HttpDelete**(**"{id}"**)]**

**public** IActionResult Delete**(**int id**)**

**{**

var motorcycle **=** \_motocykleRepository**.**GetMotorcycle**(**id**);**

**if** **(**motorcycle **==** **null)**

**{**

**return** NotFound**();**

**}**

\_motocykleRepository**.**DeleteMotorcycle**(**AutoMapper**.**Mapper**.**Map**<**Motocykl**>(**motorcycle**));**

**return** NoContent**();**

**}**

**}**

Dla wygody użytkowania API przejdźmy jeszcze do pliku **launchSettings.json**, który znajduje się w folderze **Properties.**

Zmieniamy obie wartości **„launchUrl”** na **„api/Motocykle”.**