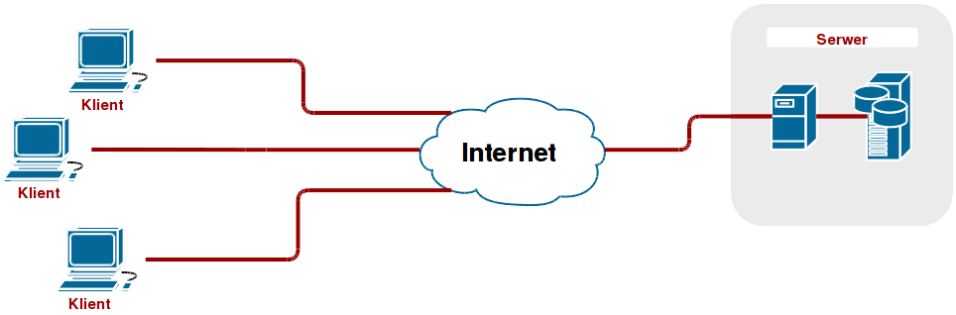
Aplikacje WWW
(studia niestacjonarne)
Laboratorium 1

Environment setup, Project setup, Build data model

1. Wprowadzenie

Aplikacje webowe są aplikacjami typu klient-serwer, czyli w uproszczeniu w odległym miejscu ustawiony został serwer podłączony do sieci Internet. Serwer ten ma nadany unikatowy adres sieciowy, po którym możemy do niego wysyłać paczki danych i utworzyć połączenie. Jego zadanie to nasłuchiwanie i przyjmowanie przychodzących połączeń. Sam zazwyczaj nie nawiązuje samodzielnie połączeń.



Na podstawie adresu i danych, które przychodzą od klienta, serwer generuje odpowiedź i odsyła klientowi, po czym połączenie zostaje zamknięte.

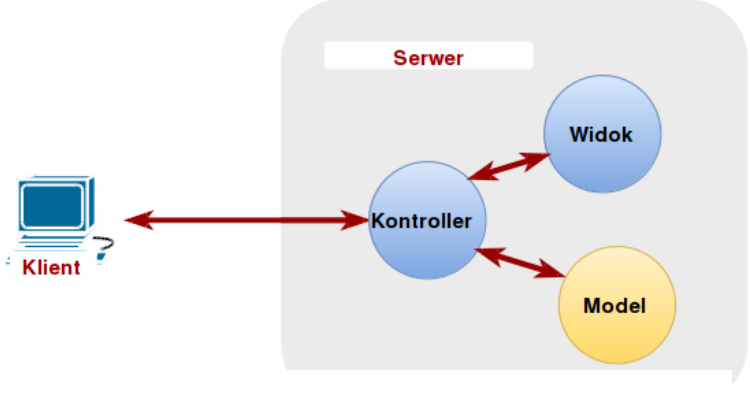
Klientami w przypadku aplikacji www są przeglądarki. Ich zadaniem jest pobieranie tekstowych dokumentów HTML-a z serwerów i renderowanie w ich wizualnej reprezentacji użytkownikowi.

Przeglądarka w odpowiedzi na akcję wykonywaną przez użytkownika zbiera dane oraz adres. Następnie nawiązuje połączenie z serwerem i przesyła dane. W odpowiedzi otrzymuje kontekst do wyrenderowania kolejnej strony.

Celem zajęć laboratoryjnych jest stopniowe tworzenie aplikacji webowej opartej o technologię **ASP.NET MVC Core**.

MVC (ang. Model, View, Controller - Model Widok Kontroler) - jest wzorcem projektowym oraz modelem programowania. Model Widok Kontroler.

- **Model** – Odpowiada za dostęp i odwzorowanie danych w tym:
  - Określa tzw. Data model – jest to model domeny aplikacji, tzn. opisujący jej cechy.
  - **Modele** klasy mapujące-odwzorowujące dane w aplikacji www np. encje bazy sql użytkowników serwisu. Przechowują dane pobrane z bazy danych.
- **Controller** – W wielu przypadkach tworzy logikę rozwiązania w tym:
  - Obsługuje żądania użytkowników.
  - Wykonuje akcje na modelu danych.
  - Dokonuje walidacji otrzymanych danych od widoku.
  - Dostarcza odpowiedni interfejs HTML (Widok) użytkownikom.
- **View** – Określa wygląd i funkcjonowanie interfejsu użytkownika **(HTML wyświetlany w przeglądarce)** oraz określa sposób prezentacji danych.



1.1. Controllers (Kontrolery)

W uproszczeniu jest to klasa, której metody są wystawione na zewnątrz serwera. Wprowadzając w przeglądarce odpowiedni adres URL można zdalnie wywołać wskazany kontroler oraz jego metodę. Domyślnie adres URL interpretowany jest w następujący sposób:

http://host:port/*{kontroler}*/*{metoda}*/*{id}*

- ***{kontroler}*** **–** nazwa klasy kontrolera z ścieżki: **/Controllers/{kontroler}Controller.cs.**

Domyślna wartość → Home

- ***{metoda}*** **–** nazwa metody z Waszego kontrolera, do której serwer przekaże wątek.

Domyślna wartość → Index

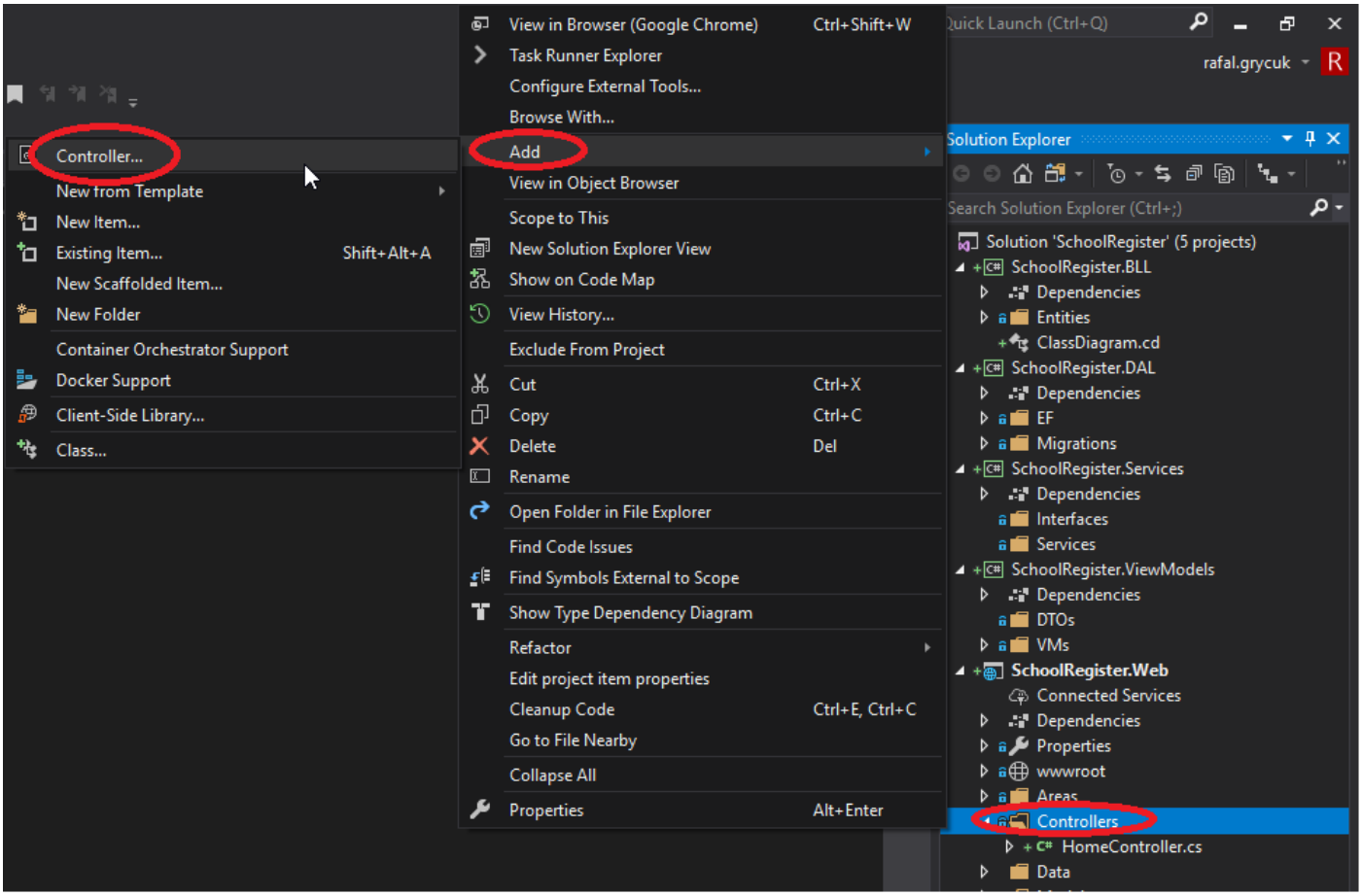
- ***{id}*** **–** dodatkowy atrybut **id** przekazywany do metody, czyli np. **Index(int id){}**

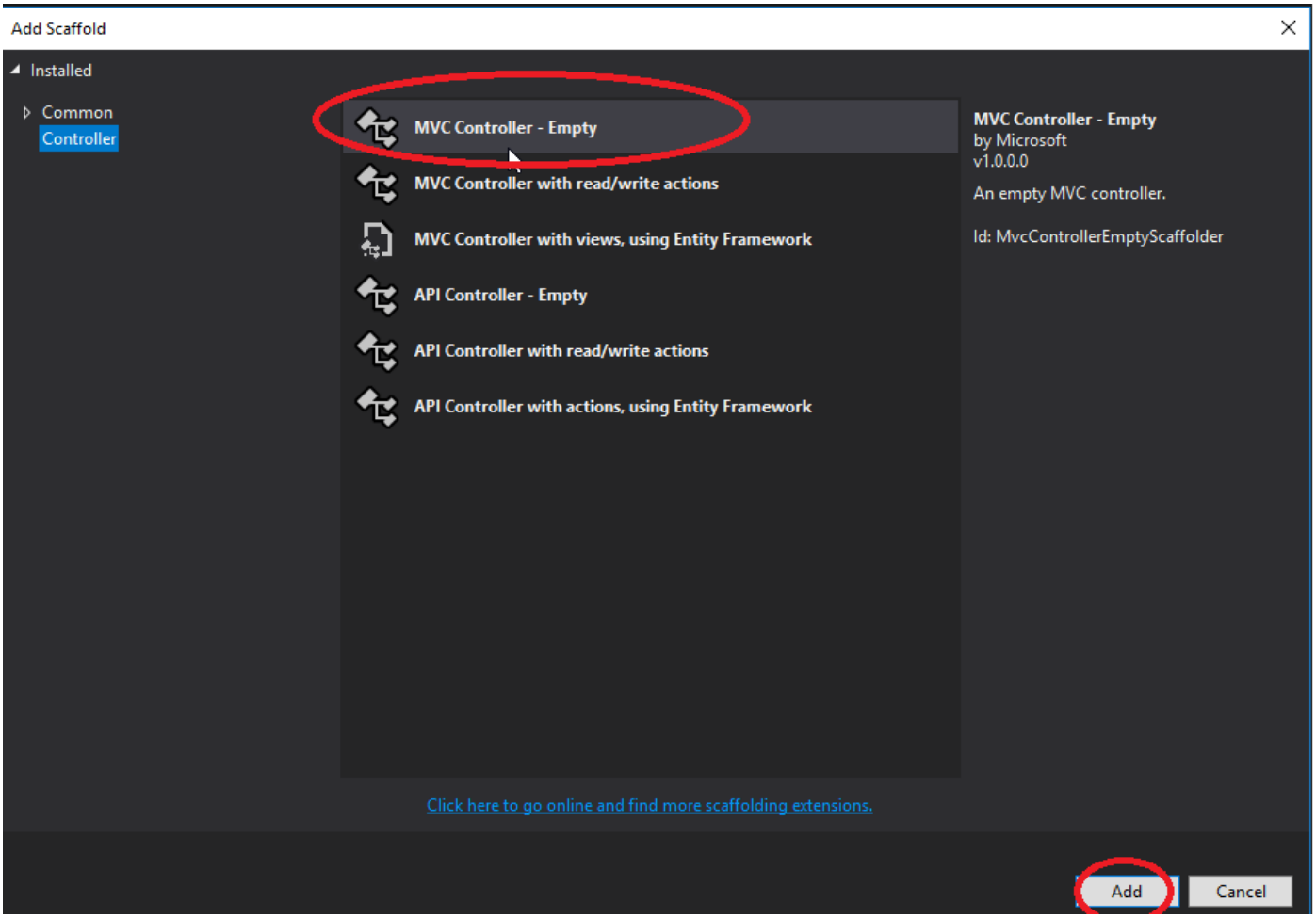
Domyślna wartość → brak

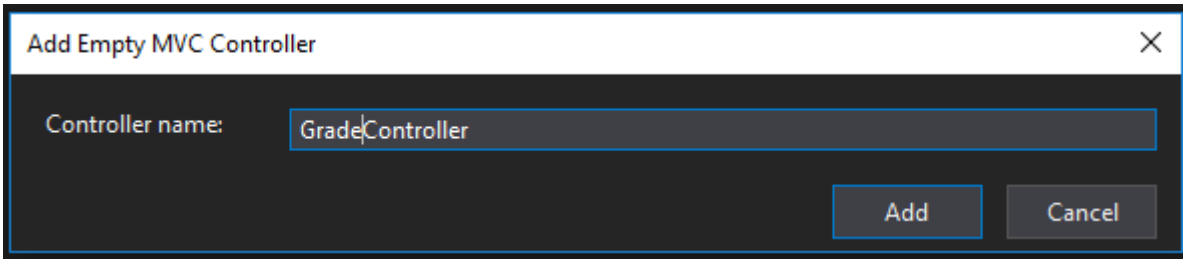
Przykład:

- **localhost:1213/Test/Test** → wywoła metodę **Test()** z klasy **Controllers/TestController** .
- **localhost:1213/Test** → wywoła metodę **Index()** z klasy **Controllers/TestController** .
- **localhost:1213/Home/About** → wywoła metodę **About()** z klasy **Controllers/HomeController** .
- **localhost:1213/Home/Index/2** → wywołań metodę **Index(int id)** z klasy Controllers/HomeController z parametrem **id** wynoszącym **2.**

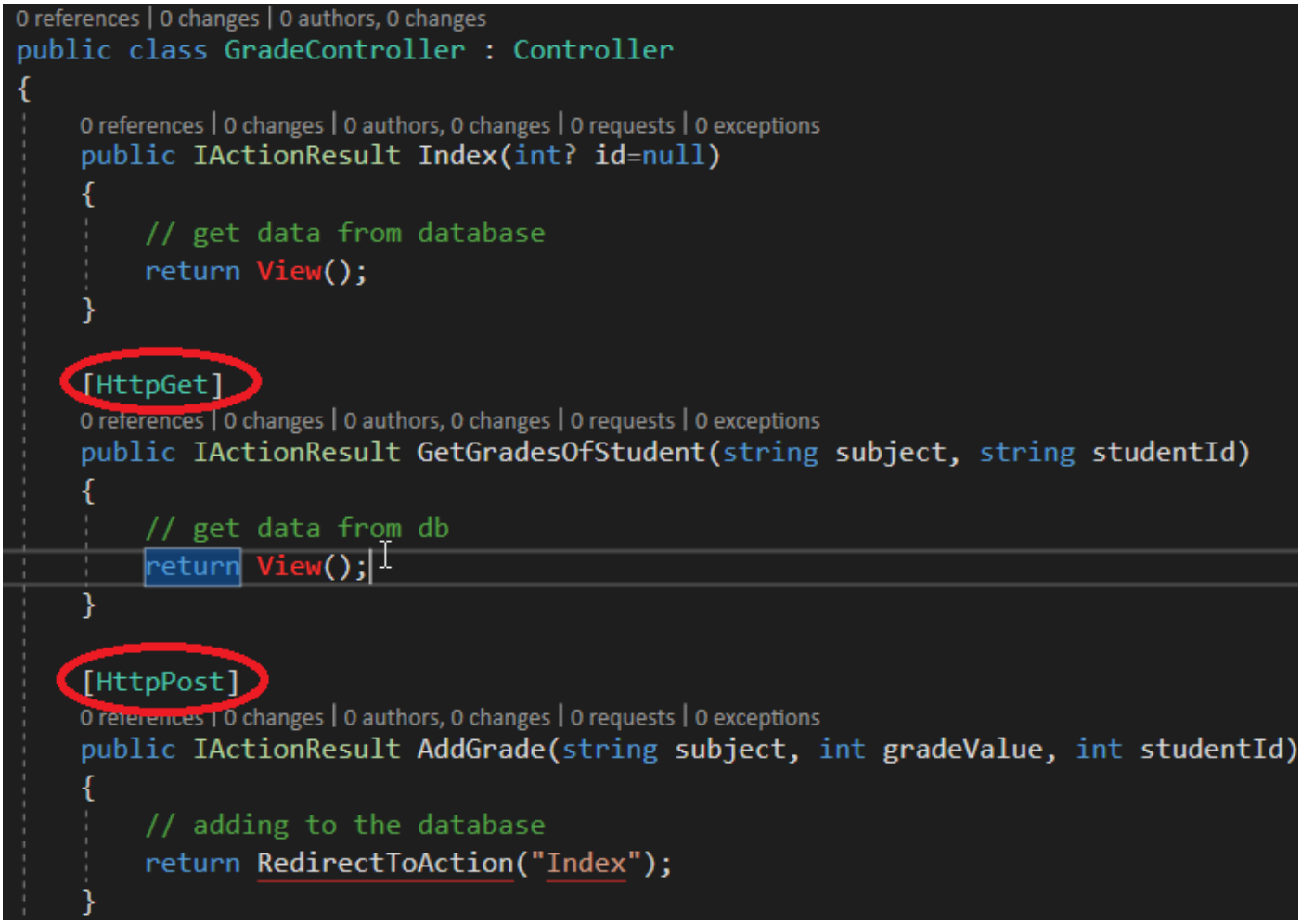
Dodawanie kontrolera (nazwa pliku ↔ nazwa kontrolera):








```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;

namespace SchoolRegister.Web.Controllers
{
    0 references | 0 changes | 0 authors, 0 changes
    public class GradeController : Controller
    {
        0 references | 0 changes | 0 authors, 0 changes | 0 requests | 0 e
        public IActionResult Index()
        {
            return View();
        }
    }
}
```



```
0 references | 0 changes | 0 authors, 0 changes
public class GradeController : Controller
{
    0 references | 0 changes | 0 authors, 0 changes | 0 requests | 0 exceptions
    public IActionResult Index(int? id=null)
    {
        // get data from database
        return View();
    }

    [HttpGet]
    0 references | 0 changes | 0 authors, 0 changes | 0 requests | 0 exceptions
    public IActionResult GetGradesOfStudent(string subject, string studentId)
    {
        // get data from db
        return View();
    }

    [HttpPost]
    0 references | 0 changes | 0 authors, 0 changes | 0 requests | 0 exceptions
    public IActionResult AddGrade(string subject, int gradeValue, int studentId)
    {
        // adding to the database
        return RedirectToAction("Index");
    }
```

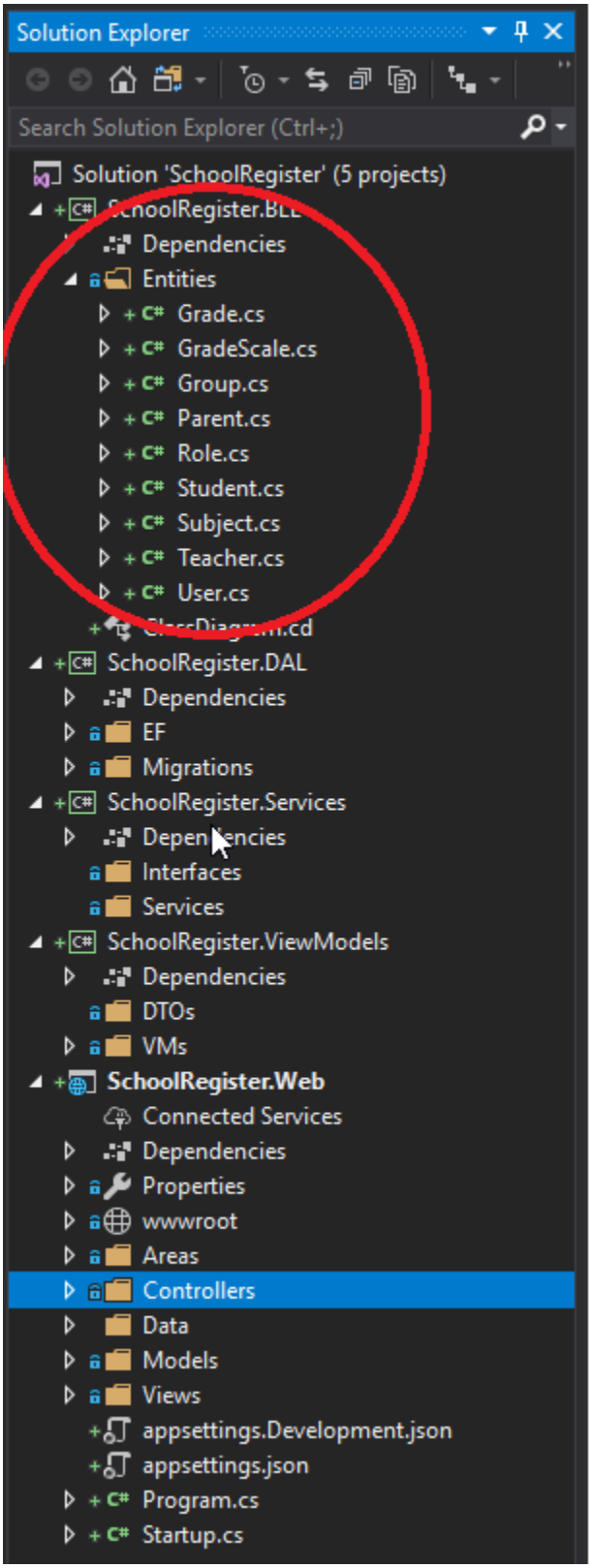
Dodając kolejne metody do kontrolera możemy uzyskać pewną funkcjonalność. W powyższym przykładzie posiadamy trzy metody, jedną **Index** drugą **AddGrade** oraz **GetGradeOfStudent.**

- **Index** przyjmuje parametr *id* typu *int?* (int lub null). URL do niego może wyglądać następująco:
  - /Grade → Index(id <= null)
  - / Grade /Index/2 → Index(id <= 2)
  - / Grade /Index → Index(id <= null)
- **GetGradeOfStudent** wymaga obu prametrów *studentId* oraz *subject***.**
  - / Grade / GetGradeOfStudent?subject=programowanie_obiektowe&studentId =10
- **AddGrade** przyjmuje trzy parametry. Metoda ta dodatkowo tworzy i zwraca przekierowanie do metody indeks po wykonaniu. Tzn. Przeglądarka trafiając w tą metodę otrzyma polecenie przekierowania na adres /Grade/Index.

Metody **Index** oraz **GetGradeOfStudent** są one typu HTTP GET. Świadczy o tym parametr zawarty tzn. parametry żądania są przesyłane w adresie URL. Metody typu GET wykorzystujemy gdy chcemy pobrać jakieś dane (lub zasób z serwera). Metoda **AddGrade** jest typu HTTP POST. Oznacza to że wartości parametrów przesyłane są do serwera w sposób niejawny, tzn. w nagłówku żądania.

1.2.Model (Data models)

Modele danych zwykle są przechowywane w folderze lub projekcie **Models**, reprezentują one tzw. domenę aplikacji czyli wszystkie obiekty które będą kooperowały w ramach aplikacji. Np. gdy domeną aplikacji jest biblioteka. Jej modelami mogą być: książka, czytelnik, pracownik biblioteki itp. W naszym przypadku będziemy przechowywać modele danych w postaci encji. W przypadku zastosowania architektury wielowarstwowej (N-Layer Architecture), są one przechowywane w osobnej bibliotece/projekcie. Zwykle taki projekt jest nazwany Model lub BLL (Business Logic Layer) lub BAL.



1.3. View (widoki)

Każda metoda kontrolera, ostatecznie ma za zadanie zwrócić wygenerowany kod HTML, który przeglądarka wyrenderuje użytkownikowi. Kod ten mógłby być generowany w kontrolerach, jednakże kod aplikacji straci dużo na przejrzystości. Z tego powodu fragmenty kodu odpowiadające za wygenerowanie widoków zostały przeniesione do katalogu **Views.** Każdy podkatalog odpowiada nazwie kontrolera, dla którego zawiera zestaw widoków. Katalog **Views/Grade/** zawiera widoki przeznaczone dla kontrolera Dom. Dodatkowo istnieje katalog **Shared** gdzie umieszcza się widoki wspólne dla wielu kontrolerów.

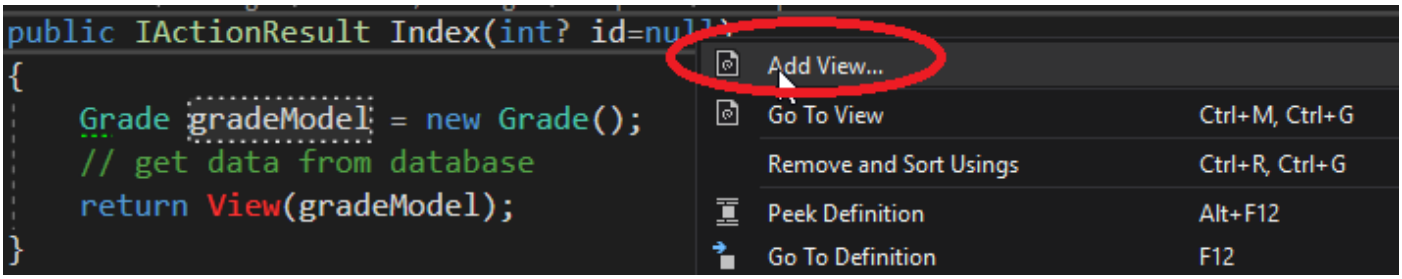
W kontrolerze wywołując metodę **View** otrzymujemy wygenerowany kod HTML na podstawie odpowiedniego widoku powiązanego z danym kontrolerem. Następnie pozostaje jedynie zwrócić go z metody kontrolera.

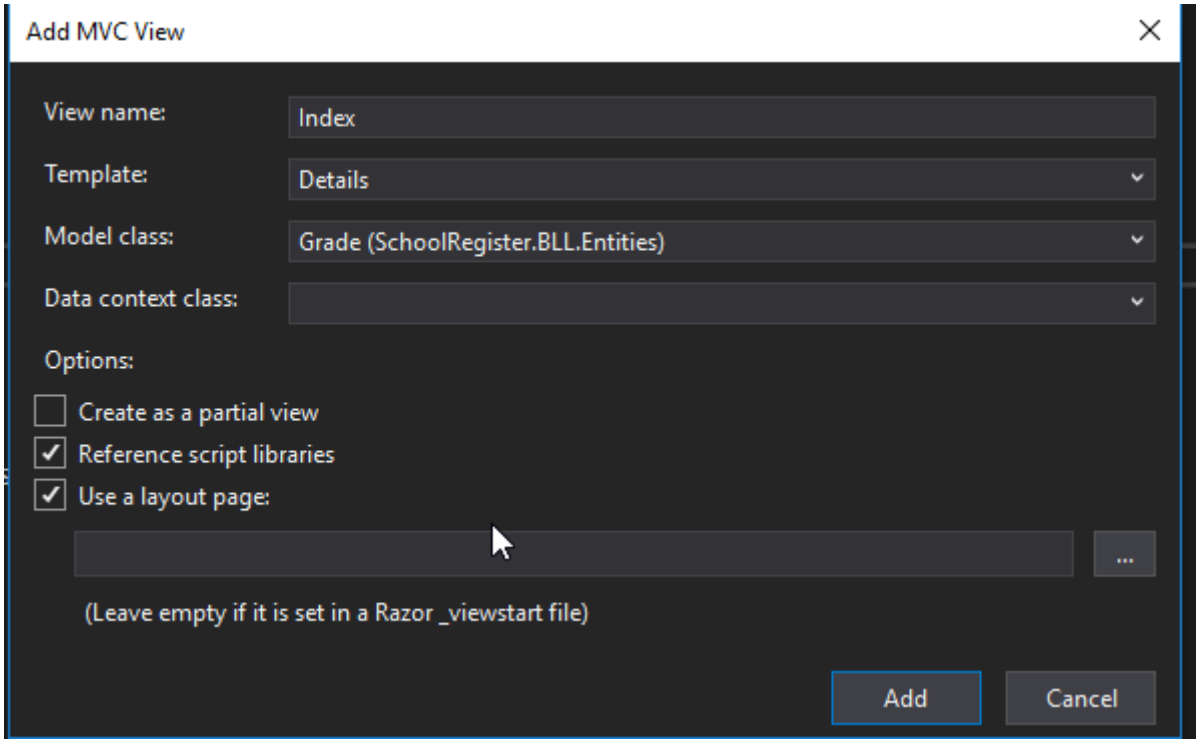
- Return View() → zwróci widok z katalogu **/Views/{Kontroler}/{Metoda}.cshtml**
- Return View(„_nazwa_") → zwróci widok z katalogu **/Views/{Kontroler}/{_nazwa_}.cshtml**

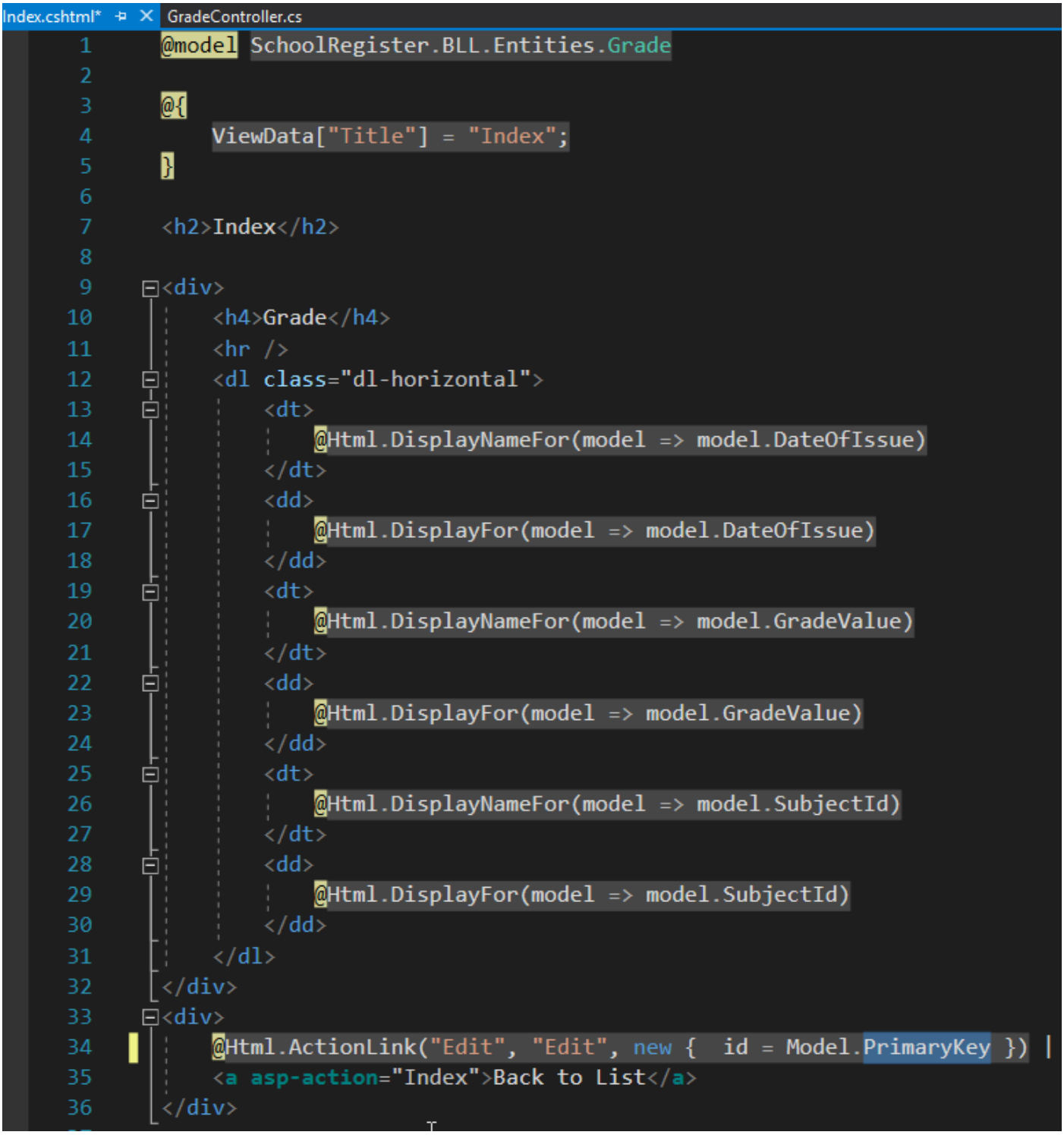
lub jeśli go nie znajdzie przeszuka następnie

- **/Views/Shared/{_nazwa_}.cshtml**

Aby dodać widok można go utworzyć w odpowiednim katalogu lub kliknąć prawym przyciskiem w bloku metody kontrolera i wybrać „**Add View".**







Jak widzimy utworzony widok zawiera jedynie wycinek HTML-a. Całość znajduje się w pliku **/Views/Shared/_Layout.cshtml,** gdzie nasz fragment kodu będzie wyrenderowany i wklejony w linijce **@RenderBody().**

Widok został odseparowany od klasy kontrolera, z tego powodu można przekazywać dane do widoku za pomocą:

- **ViewData[„nazwa elementu”]** - tablica asocjacyjna widoczna w widoku i kontrolerze.
- **ViewBag.nazwa_element** – obiekt typu ***dynamic***, może przechowywać dowolny typ danych.
- **TempData[„nazwa elementu”] –** tablica asocjacyjna, przechowuje wartości, dopóki nie zostanie odczytana. Do sprawdzenia danych bez usuwania można użyć metod *Keep* and *Peek*. *TempData* jest szczególnie przydatna do przekierowania, gdy dane są wymagane dla więcej niż jednego żądania.

- **Model Danych** → Widok typu „strong typed” (silnie typowany) - widok dedykowany pod specyficzny obiekt modelu danych.

W widokach typu Razor odwołania do wartości zmiennej lub innych np. funkcji poprzedza się znakiem '**@**'

- **using** → dołączenie namespace np. modeli danych
  @using SchoolRegister.Entities;
- **pętle**

```
@foreach (var item in Model){
 <p>@item</p>
}
```

- **warunki**

```
@if (Model.Length == 0){
 <p>Brak towaru</p>
}
else{
 <p>Towar dostepny</p>
}
```

- **boki kodu**

```
@{
 string text="Hello";
 text+=" World";
}
@text
```

2. Przygotowanie środowiska (Environment setup)

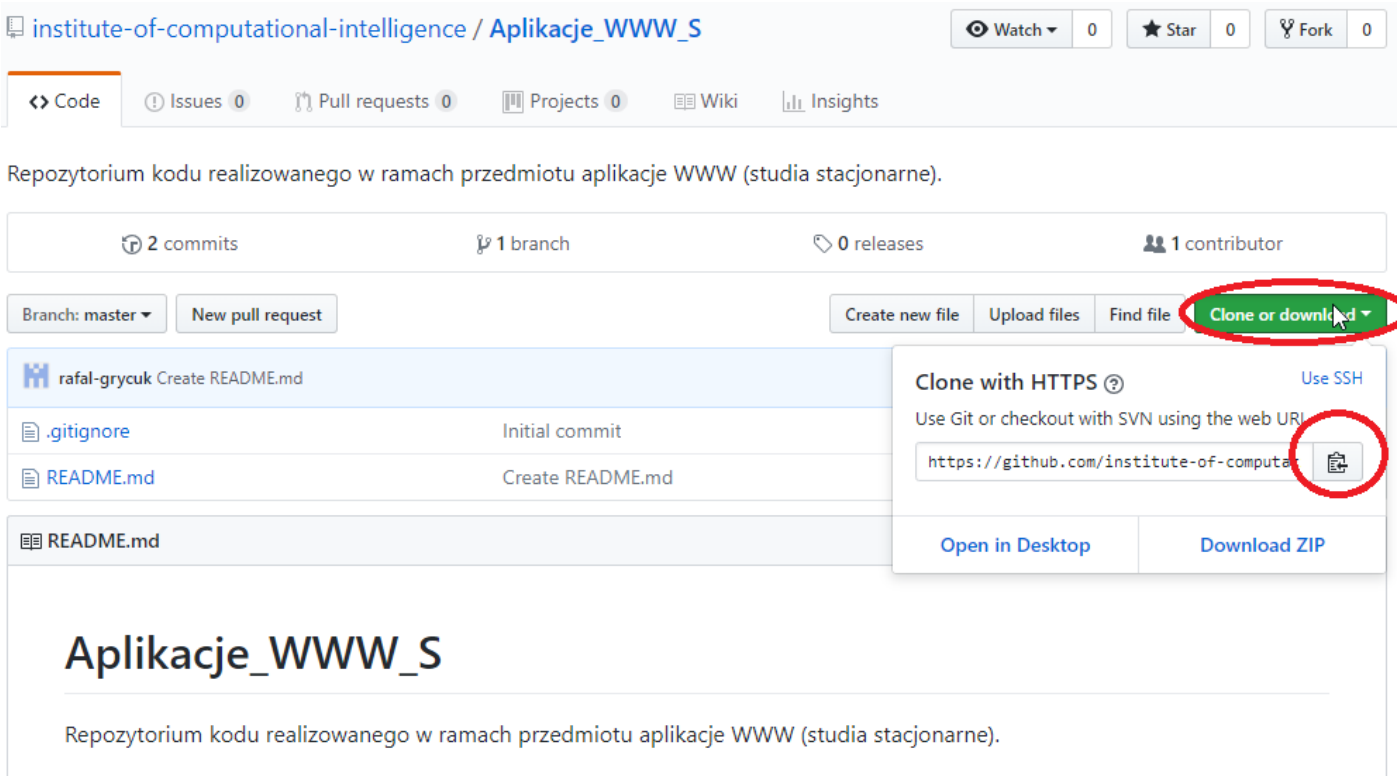
W trakcie zajęć laboratoryjnych będziecie Państwo korzystać z poniższego oprogramowania proszę się upewnić że jest ono zainstalowane na Państwa komputerze.

- Git (MacOS, Linux) lub Git Extensions (Windows)
- .NET SDK (.NET 5.0 lub wyższy)
- SQL Server LocalDB (Windows) lub pełny SQL Server Express (MacOS, Linux)
- Azure Data Studio
- Visual Studio Code

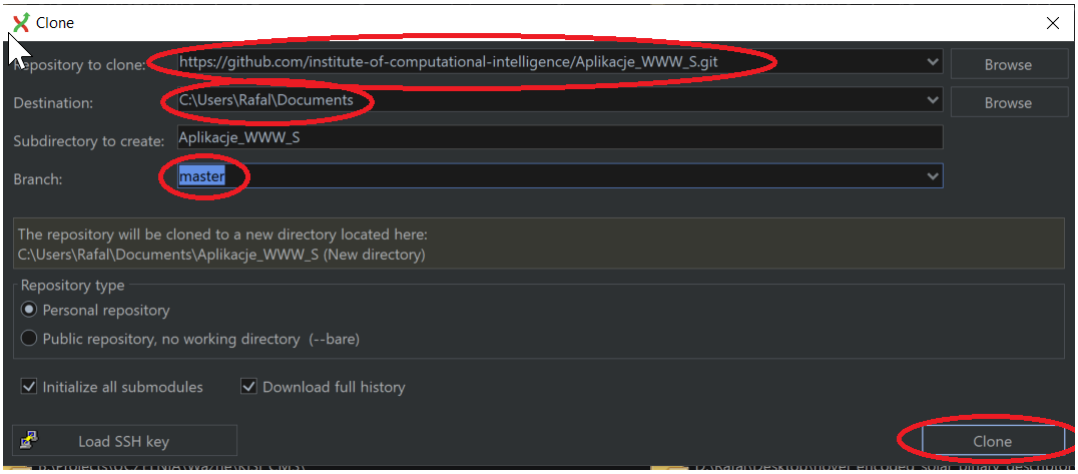
2.1. Git Extensions

Do celów wersjonowania będziemy wykorzystywać platformę github.com. W tym celu potrzebny jest klient Git. Jednym z nich jest Git Extensions. Program Git Extensions jest dostępny jedynie dla systemu Windows, jednakże użytkownicy Linuxa lub MacOS mogą wykorzystać program git zawarty w git extensions. Po pobraniu i instalacji narzędzie należy:

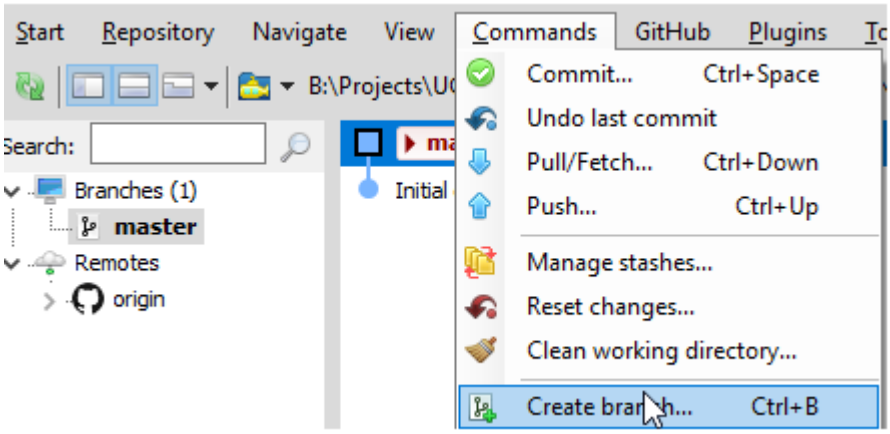
1) Proszę przejść pod poniższy adres:
   https://github.com/department-intelligent-computer-systems/Aplikacje_WWW_N
2) Następnie klikamy *Clone or download* oraz kopiujemy link.



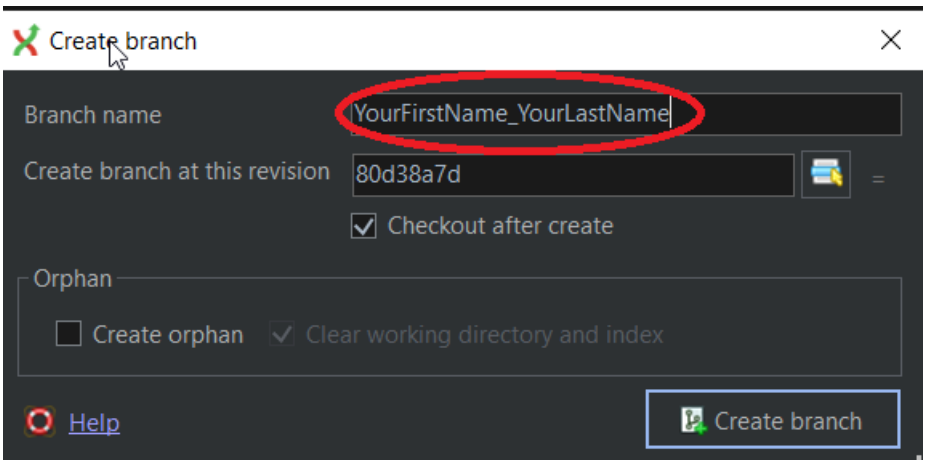
3) Następnie uruchamiany klienta gita np. Git Extensions.
4) Klikamy *Clone repository.*
5) W nowym okienku wklejmy link
6) Następnie wskazujemy ścieżkę w której będziemy przechowywać całą solucję.



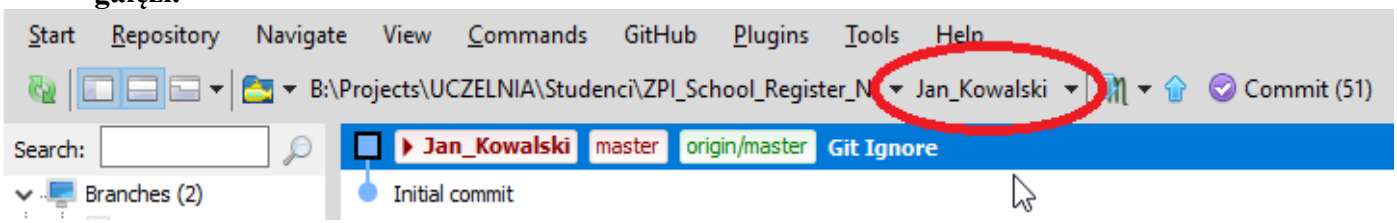
7) Na końcu klikamy *Clone.* W tym momencie klient gita pobierze całe repozytorium z github.com oraz oraz zapyta się czy chcemy je otworzyć, klikamy *yes.*
8) Następnie należy utworzyć nową gałąź (jedna gałąź = jeden student).
9) W Git Extensions klikamy *Commands-> Create branch.*



10) Następnie proszę wpisać **swoje** imię i nazwisko jako nazwę gałęzi, według poniższego wzroru: {imię}_{nazwisko}. Przykład poniżej:

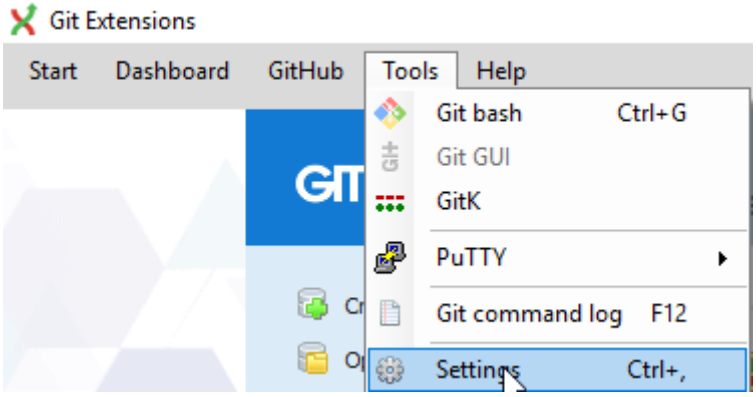


11) **Podczas pracy nad projektem proszę zawsze sprawdzać czy znajdujecie się państwo na swojej gałęzi.**

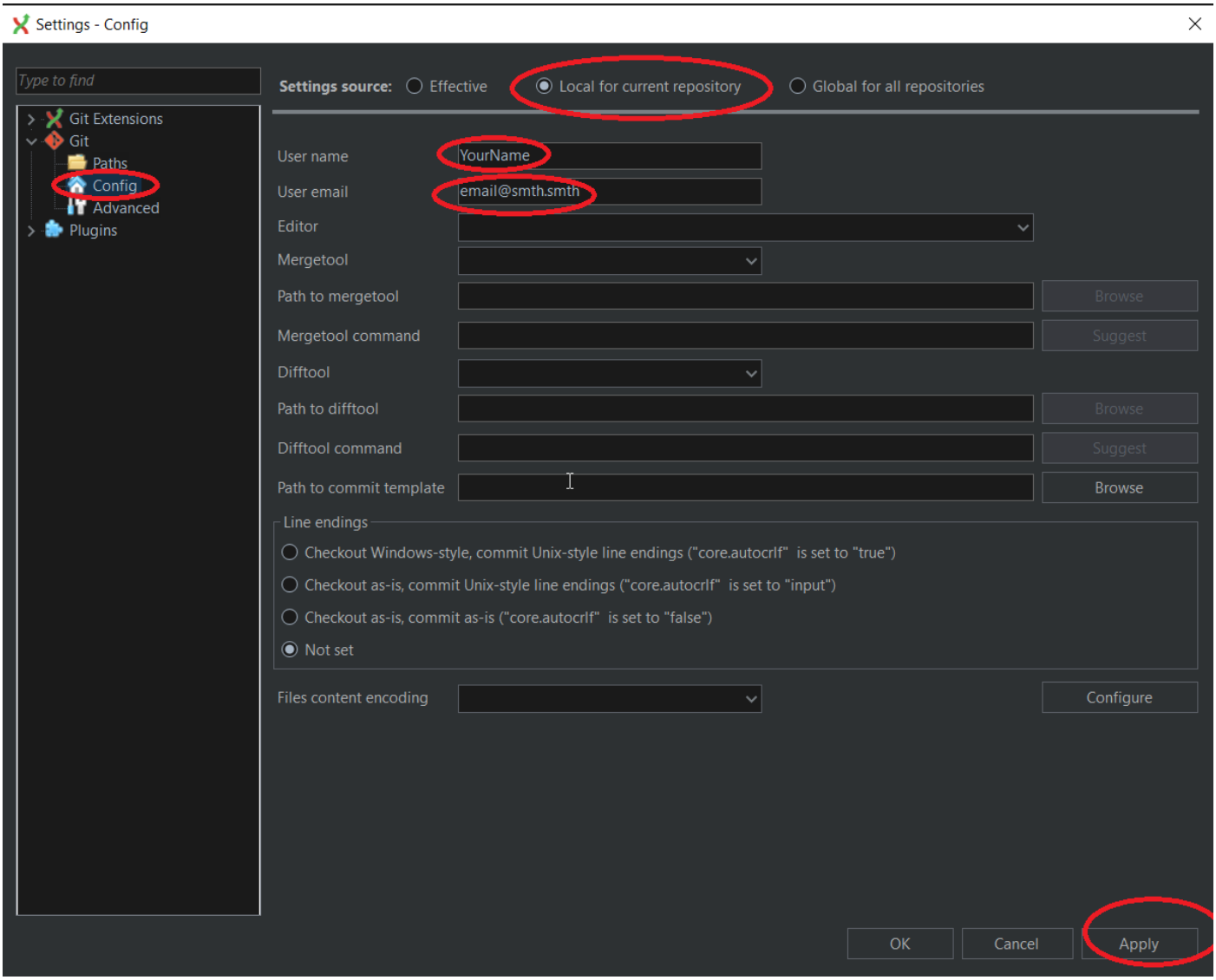


Rys. 1

12) Należy również zwrócić uwagę na ustawienia Git Extensions. Tools -> Settings



13) Następnie przechodzimy do git->Config.



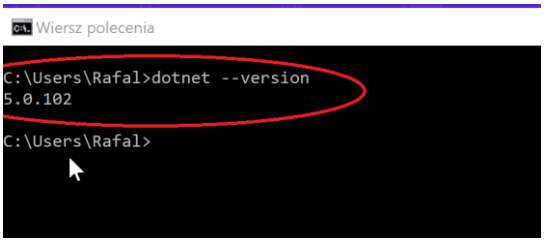
14) Podajemy dowolną nazwę użytkownika oraz email. Następnie zatwierdzamy.

2.2. .NET SDK

Aby tworzyć aplikacje we frameworku .NET 5 lub wyższym, należy zainstalować środowisko uruchomieniowe .NET SDK. Aplikacje napisane na tej platformie mogą być uruchamiane zarówno na systemie Windows jak i Linux czy MacOS. Środowisko to wymaga dedykowanego SDK, dostępnego dla różnych platform, które należy pobrać i zainstalować. Niezwykle istotnym jest aby zainstalować wersję .NET 5.0 dla wybranej platformy. Ważne jest również aby był to pakiet SDK. Po poprawnej instalacji proszę uruchomić konsolę i wpisać:

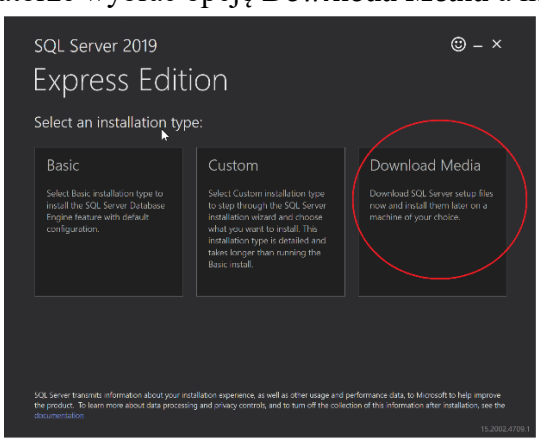
```
dotnet --version
```

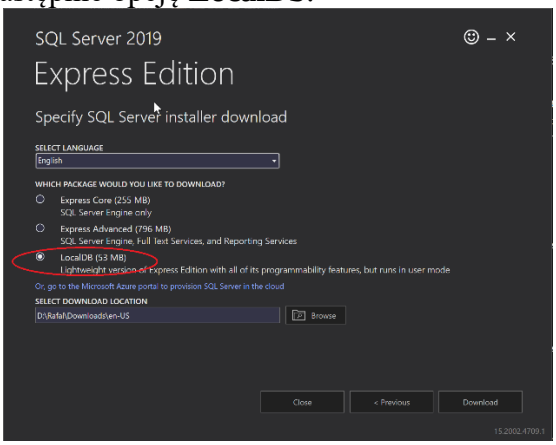
Jeśli w odpowiedzi otrzymaliście Państwo wersję .NET SDK oznacza to że pakiet został zainstalowany poprawnie.



2.3. SQL Server LocalDB

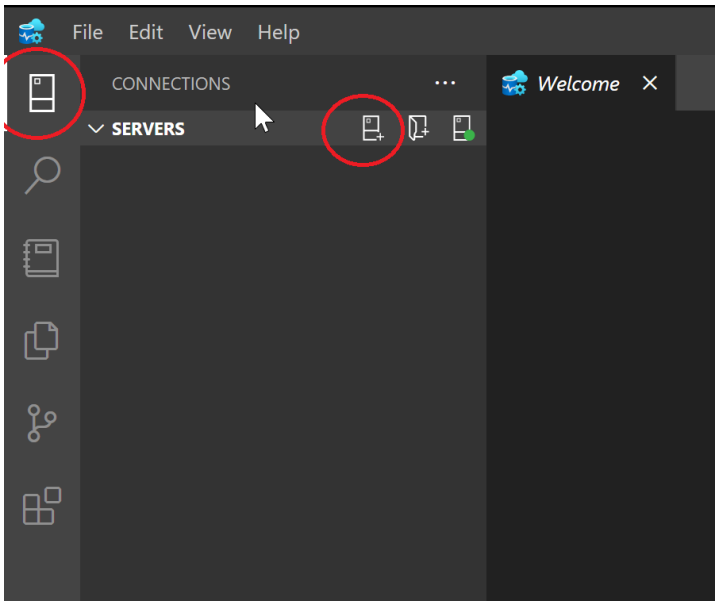
W celu przechowywania danych aplikacji oraz zachowania mechanizmu wielodostępności bazy danych zdecydowano się na SQL Server Express LocalDB. Narzędzie to pozwala na korzystania z silnika bazy danych SQL Server. Zawiera jedynie niezbędne składniki aby korzystać z bazy danych. W systemie Windows możliwe jest zainstalowanie jedynie składnika SQL Server LocalDB. Natomiast w przypadku systemu Linux lub MacOS należy zainstalować całą bazę SQL Server Express. Jeżeli jeszcze nie zostało ono zainstalowane, proszę pobrać i zainstalować to oprogramowanie. Aby zainstalować **LocalDb** proszę w instalatorze wybrać opcję ***Download Media*** a następnie opcję **LocalDb**.





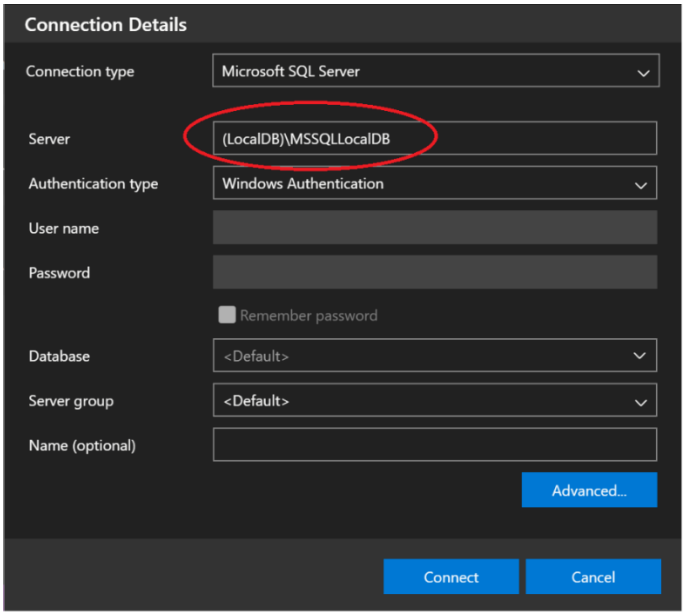
2.4. Azure Data Studio

Azure Data Studio to wieloplatformowe narzędzie służące do eksploracji baz danych oraz lokalnych i chmurowych platform danych w systemach Windows, macOS i Linux. Azure Data Studio oferuje nowoczesne środowisko edytora z technologią IntelliSense, fragmentami kodu, integracją kontroli źródła i zintegrowanym terminalem. Został zaprojektowany z myślą o użytkowniku platformy danych, z wbudowanym modułem wykresów oraz zestawów wyników zapytań i konfigurowalnymi pulpitami nawigacyjnymi. Jeżeli jeszcze nie zostało ono zainstalowane, proszę pobrać i zainstalować to oprogramowanie. Aby przetestować działanie Azure Data Studio oraz SQL Server LocalDB proszę stworzyć połączenie:

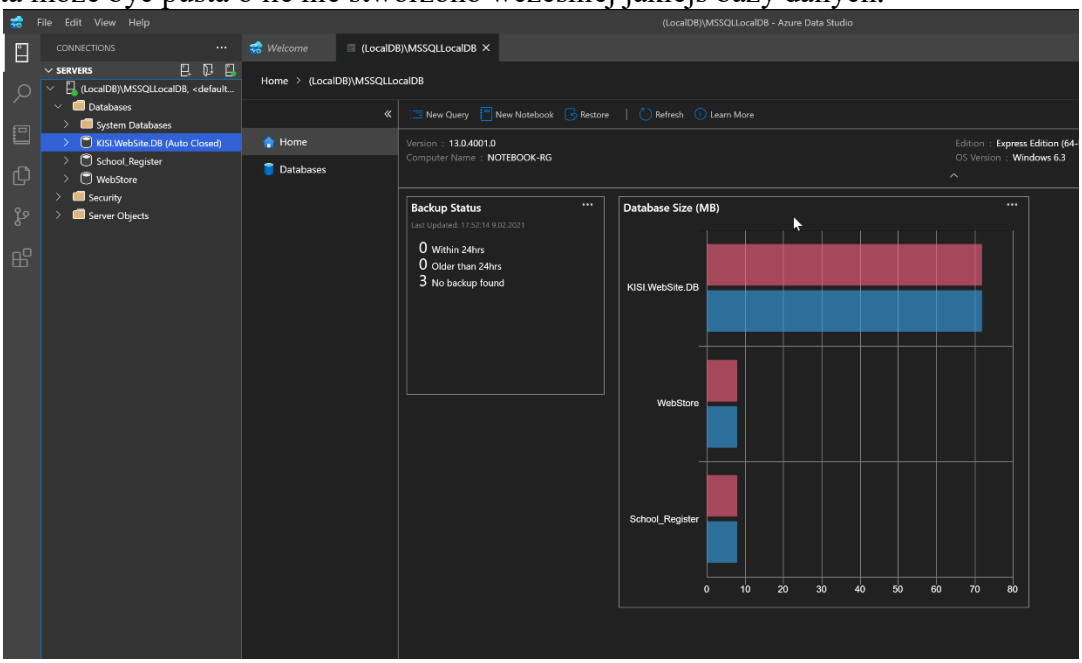


15) Następnie w oknie konfiguracji połączenia, w sekcji Server proszę wpisać:

```
(LocalDB)\MSSQLLocalDB
```

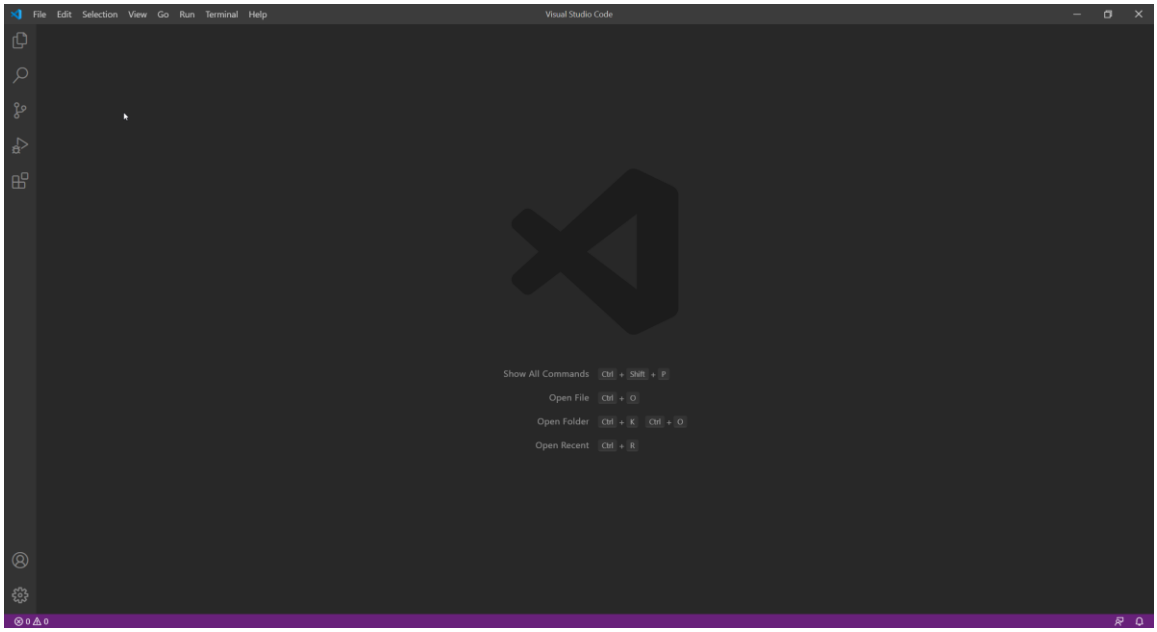


W efekcie powinniście Państwo połączyć się z serwerem oraz otrzymać listę baz danych (oczywiście lista ta może być pusta o ile nie stworzono wcześniej jakiejś bazy danych.



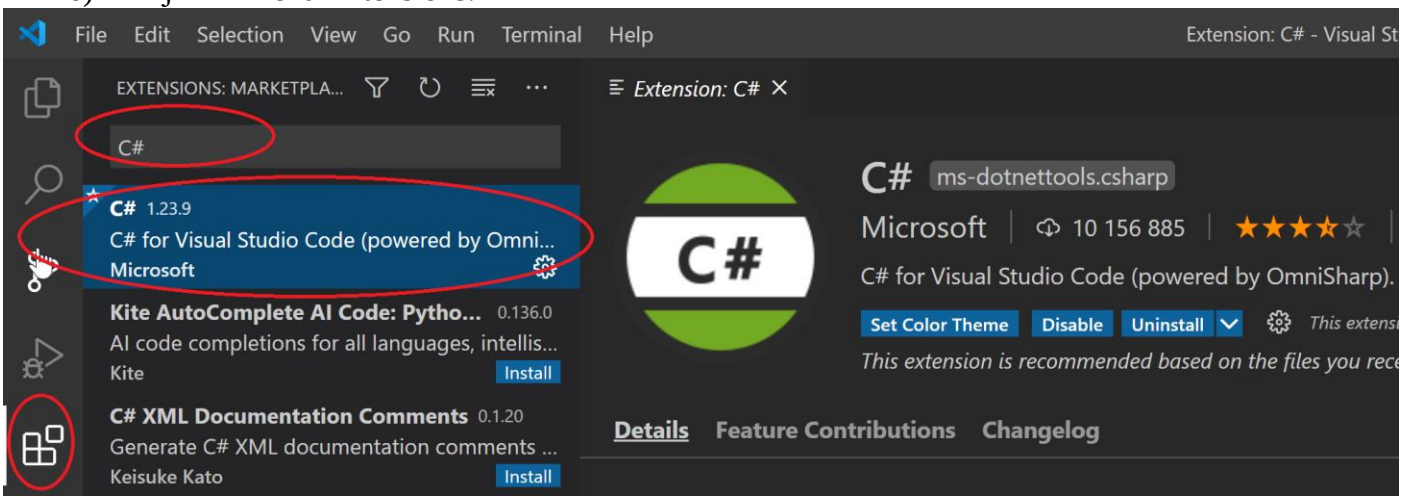
2.5. Visual Studio Code

Visual Studio Code to rozszerzony edytor kodu z obsługą operacji programistycznych, takich jak debugowanie, uruchamianie zadań oraz kontrola wersji. Ma na celu zapewnienie tylko niezbędnych narzędzi potrzebnych programistom do szybkiego cyklu kompilowania i debugowania kodu. Jest to rozwiązanie w pełni modułowe i poprzez wykorzystanie dodatków (extensions) możliwe jest rozbudowanie Visual Studio Code do pełnoprawnego IDE. Jednakże w takim przypadku może się okazać że lepszym rozwiązaniem jest zainstalowanie Visual Studio IDE. VS Code to rozwiązane dedykowane dla wielu języków oraz frameworków. W większości z nich samo narzędzie zaproponuje najbardziej odpowiedni dodatek do zainstalowania. Ogromną zaleta VS Code jest wieloplatformowość. Obsługuje zarówno system Windows, jak i Linux czy MacOS. Kolejną zaletą tego narzędzia jest stosunkowo niewielki rozmiar w szczególności w porównaniu do VS IDE. Jeżeli jeszcze nie zostało ono zainstalowane, proszę pobrać i zainstalować narzędzie VS Code. Po uruchomieniu powinniście Państwo uzyskać poniższy rezultat:



Aby tworzyć aplikacje w przy pomocy .NET oraz C# należy zainstalować dodatek umożliwiający korzystanie z IntelliSense oraz analizy języka. W tym celu proszę:

16) Przejść do menu Extensions:



17) Następnie proszę wpisać C# w polu wyszukiwania i zainstalować powyższy dodatek.
W ten sposób środowisko do tworzenia aplikacji webowych zostało przygotowane.

3. Tworzenie solucji (Solution setup)

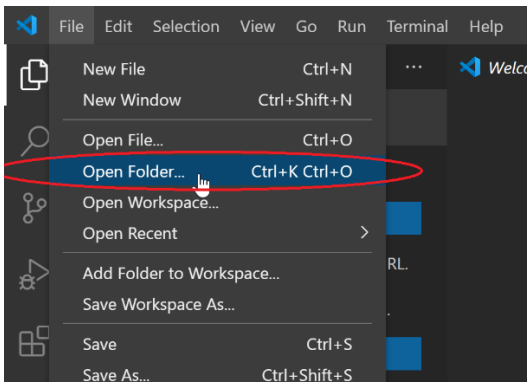
W trakcie zajęć laboratoryjnych będziecie Państwo tworzyć i rozwijać nowy system, będący dziennikiem szkolnym *School Register*. System będzie rozwijany przy pomocy frameworka ASP.NET Core MVC.
System początkowo powinien posiadać następujące funkcjonalności:

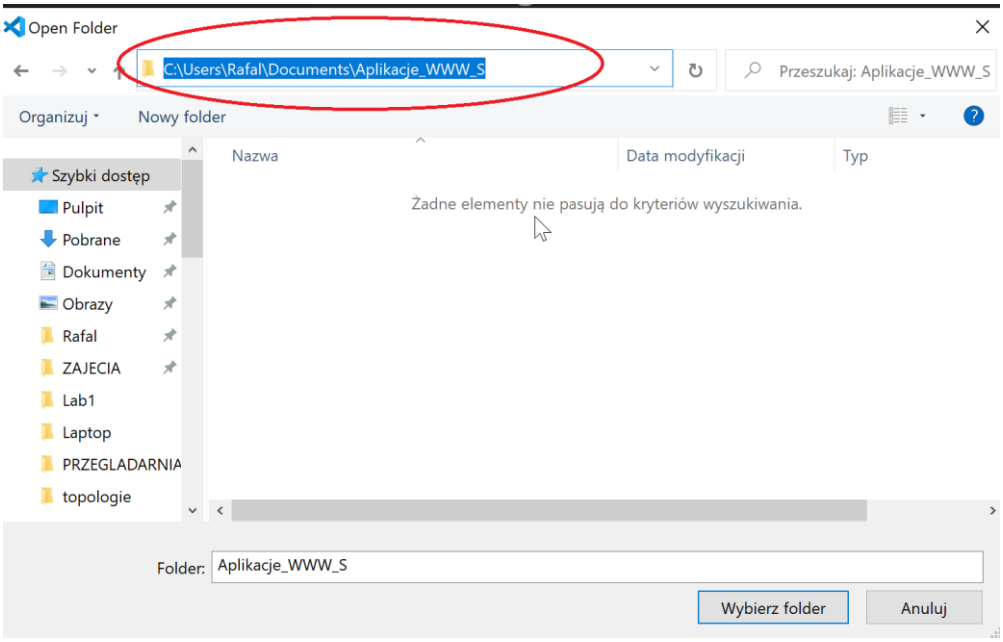
- Logowanie do systemu.
- Zarządzanie użytkownikami (tworzenie, modyfikacja) przez rolę *Admin*.
- W systemie powinny istnieć następujące role użytkowników: (*Teacher, Student, Parent, Admin).*
- Dodawanie, modyfikacja, oraz usuwanie przedmiotów.
- Wystawianie ocen przez użytkowników z rolą *Teacher*.
- Możliwość wyświetlania swoich ocen przez użytkowników z rolą *Parent* lub *Student*.
- Funkcjonalność obliczenia oraz wyświetlenia średniej arytmetycznej dla wszystkich ocen.
- Funkcjonalność obliczenia oraz wyświetlenia średniej arytmetycznej z podziałem na przedmioty.
- Użytkownicy z rolą *Teacher* posiadają możliwość wysyłania emaila do użytkowników z rolą *Parent*.
- Zarządzanie grupami (klasami) (tworzenie oraz usuwanie grup)
- Dodawanie oraz usuwanie studentów do i z grup.

W celu stworzenia solucji z podziałem na poszczególne projekty należy postępować według poniższych kroków:

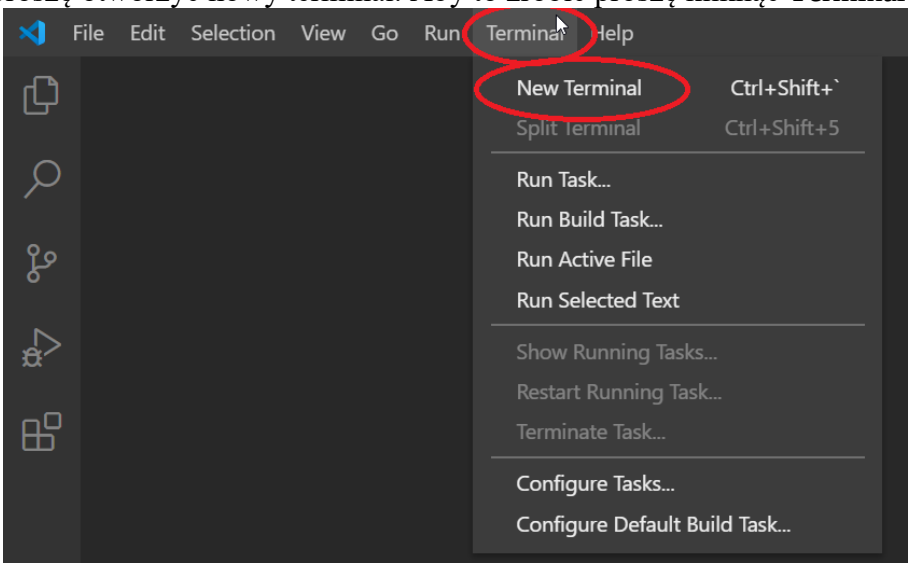
1) W kolejnym kroku uruchamiamy Visual Studio Code.
2) Klikamy *File -> Open folder*. Następnie przechodzimy do folderu w którym zostało sklonowane repozytorium. Niezwykle ważne jest aby koniec ścieżki zawierał folder:

```
…\Aplikacje_WWW_N
```





3) Następnie proszę otworzyć nowy terminal. Aby to zrobić proszę kliknąć Terminal->New Terminal



4) Aplikacja będzie składała się z następujących warstw (projektów):

Nazwa projektu	**Typ projektu**	**Parametr**
SchoolRegister.Model	Class library	classlib
SchoolRegister.DAL	Class library	classlib
SchoolRegister.Services	Class library	classlib
SchoolRegister.ViewModels	Class library	classlib
SchoolRegister.Web	ASP.NET Core Web App (Model-View-Controller)	mvc

5) W związku z tym aby stworzyć pierwszy projekt, proszę w terminalu wpisać następujące polecenie

```
dotnet new classlib -n SchoolRegister.Model -f net5.0
```

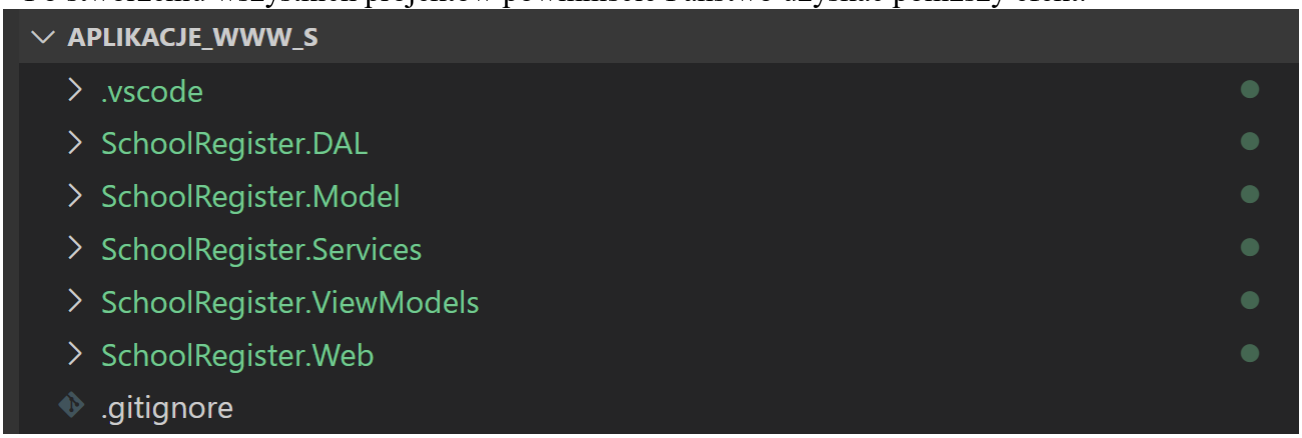
Jak można zauważyć parametr -n oznacza nazwę projektu natomiast parametr -f oznacza wersję .NET.

```
PS B:\Projects\UCZELNIA\Studenci\Aplikacje_WWW_S> dotnet new classlib -n SchoolRegister.Model -f net5.0
```

6) Proszę powtórzyć powyższy punkt dla wszystkich projektów typu *Class library*. Czyli wszystkich pozostałych **oprócz *SchoolRegister.Web***
7) Wspomniany projekt ***SchoolRegister.Web*** należy stworzyć przy pomocy poniższego polecenia:

```
dotnet new mvc -n SchoolRegister.Web -au Individual -f net5.0 -rrc -uld
```

Parametr -au określa typ autentykacji w naszym przypadku będzie to Individual, co oznacza że dane autentykacyjne będą przetrzymywane lokalnie. Parametr -rrc oznacza razor runtime compilation, co pozwala na dynamiczną kompilację widoków bez restartu serwera. Parametr -uld pozwala na wykorzystanie LocalDb zamiast SQL Lite.
Po stworzeniu wszystkich projektów powinniście Państwo uzyskać poniższy efekt:



8) Następnie należy stworzyć plik solucji: aby to zrobić proszę wykonać poniższe polecenie:

```
dotnet new sln -n SchoolRegister
```

9) Kolejno należy dodać wszystkie wcześniej stworzone projekty do stworzonej przed chwilą solucji.

```
dotnet sln SchoolRegister.sln add (ls -r **/*.csproj)
```

```
PS B:\Projects\UCZELNIA\Studenci\Aplikacje_WWW_S> dotnet sln SchoolRegister.sln add (ls -r **/*.csproj)
Dodano projekt "SchoolRegister.DAL\SchoolRegister.DAL.csproj" do rozwiązania.
Dodano projekt "SchoolRegister.Model\SchoolRegister.Model.csproj" do rozwiązania.
Dodano projekt "SchoolRegister.Services\SchoolRegister.Services.csproj" do rozwiązania.
Dodano projekt "SchoolRegister.ViewModels\SchoolRegister.ViewModels.csproj" do rozwiązania.
Dodano projekt "SchoolRegister.Web\SchoolRegister.Web.csproj" do rozwiązania.
```

10) W kolejnym kroku należy dodać odpowiednie referencje do projektów. **Można to zrobić na dwa sposoby**:
    - Poprzez wykonanie komendy `dotnet add {source project} reference {destination project}`
      Przykład dodania referencji do projektu *SchoolRegister.Model* z projektu SchoolRegister.Web, dzięki temu klasy stworzone w projekcie *SchoolRegister.Model*, będą widoczne w projekcie *SchoolRegister.Web*

```
dotnet add SchoolRegister.Web/SchoolRegister.Web.csproj reference
SchoolRegister.Model/SchoolRegister.Model.csproj
```

- Drugim sposobem na dodanie referencji jest dodanie wpisu w pliku *.csproj. Przykład dodania referencji do projektu *SchoolRegister.Model* z projektu *SchoolRegister.DAL*. Poniżej zawartość pliku SchoolRegister.DAL.csproj

```
<Project Sdk="Microsoft.NET.Sdk">

  <ItemGroup>
    <ProjectReference Include="..\SchoolRegister.Model\SchoolRegister.Model.csproj" />
  </ItemGroup>

  <PropertyGroup>
    <TargetFramework>net5.0</TargetFramework>
  </PropertyGroup>
</Project>
```

11) Proszę dodać referencje do poszczególnych projektów w następujący sposób:
- Z projektu *SchoolRegister.DAL* do:
  - *SchoolRegister.Model*
- Z projektu *SchoolRegister.ViewModels* do:
  - *SchoolRegister.Model*
- Z projektu *SchoolRegister.Services* do:
  - *SchoolRegister.Model*
  - *SchoolRegister.DAL*
  - *SchoolRegister.ViewModels*
- Z projektu *SchoolRegister.Web* do:
  - *SchoolRegister.Model*
  - *SchoolRegister.DAL*
  - *SchoolRegister.ViewModels*
  - *SchoolRegister.Services*

Poniżej znajdują się zawartości plików *.csproj dla powyższych projektów:
- *SchoolRegister.Model*

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <TargetFramework>net5.0</TargetFramework>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Microsoft.Extensions.Identity.Stores" Version="5.0.3" />
  </ItemGroup>

</Project>
```

- *SchoolRegister.DAL*

```
<Project Sdk="Microsoft.NET.Sdk">

  <ItemGroup>
    <ProjectReference Include="..\SchoolRegister.Model\SchoolRegister.Model.csproj" />
  </ItemGroup>

  <PropertyGroup>
    <TargetFramework>net5.0</TargetFramework>
  </PropertyGroup>
</Project>
```

- *SchoolRegister.ViewModels*

```
<Project Sdk="Microsoft.NET.Sdk">

  <ItemGroup>
    <ProjectReference Include="..\SchoolRegister.Model\SchoolRegister.Model.csproj" />
  </ItemGroup>

  <PropertyGroup>
    <TargetFramework>net5.0</TargetFramework>
  </PropertyGroup>

</Project>
```

- *SchoolRegister.Services*

```
<Project Sdk="Microsoft.NET.Sdk">

  <ItemGroup>
    <ProjectReference Include="..\SchoolRegister.Model\SchoolRegister.Model.csproj" />
    <ProjectReference Include="..\SchoolRegister.DAL\SchoolRegister.DAL.csproj" />
    <ProjectReference Include="..\SchoolRegister.ViewModels\SchoolRegister.ViewModels.csproj" />
  </ItemGroup>

  <PropertyGroup>
    <TargetFramework>net5.0</TargetFramework>
  </PropertyGroup>

</Project>
```

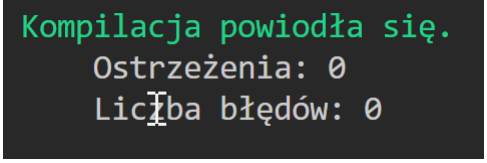
- *SchoolRegister.Web*

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>net5.0</TargetFramework>
    <UserSecretsId>aspnet-SchoolRegister.Web-9FD2D47D-E61A-4AC3-A821-C3CC24A0F1A7</UserSecretsId>
    <CopyRefAssembliesToPublishDirectory>false</CopyRefAssembliesToPublishDirectory>
  </PropertyGroup>

  <ItemGroup>
   <PackageReference Include="Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore" Version="5.0.3"/>
    <PackageReference Include="Microsoft.AspNetCore.Identity.EntityFrameworkCore" Version="5.0.3" />
    <PackageReference Include="Microsoft.AspNetCore.Identity.UI" Version="5.0.3" />
    <PackageReference Include="Microsoft.AspNetCore.Mvc.Razor.RuntimeCompilation" Version="5.0.3" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="5.0.3" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="5.0.3" />
  </ItemGroup>

  <ItemGroup>
    <ProjectReference Include="..\SchoolRegister.Model\SchoolRegister.Model.csproj" />
    <ProjectReference Include="..\SchoolRegister.DAL\SchoolRegister.DAL.csproj" />
    <ProjectReference Include="..\SchoolRegister.ViewModels\SchoolRegister.ViewModels.csproj" />
    <ProjectReference Include="..\SchoolRegister.Services\SchoolRegister.Services.csproj" />
  </ItemGroup>

</Project>
```

12) Proszę skompilować solucję, w tym celu należy w konsoli wykonać poniższe polecenie:

```
dotnet build
```

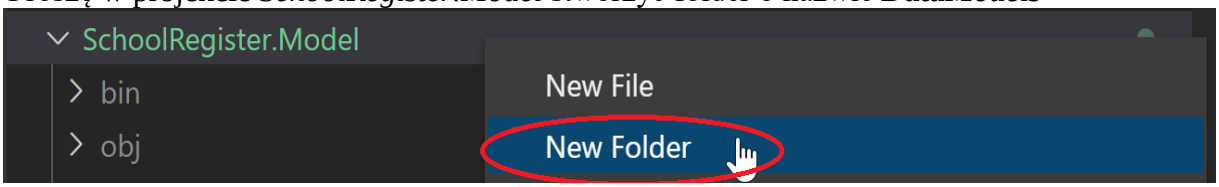
Jeżeli wszystkie kroki zostały wykonane poprawnie kompilacja powinna zakończyć się sukcesem.



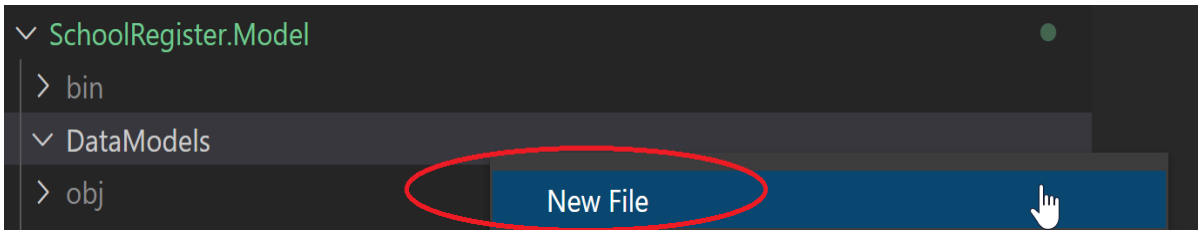
4. Tworzenie modelu danych (Creating Data Model)

Fundamentem aplikacji webowych jest tzw. data model. Jest to model na którym będzie operować aplikacja. Najczęściej model danych tworzy się przy wykorzystaniu klas, które definiują wspomniany model danych. Aby stworzyć model danych należy wykonać poniższe kroki:

1) Proszę usunąć w projekcie *SchoolRegister.Model* usunąć plik **Class1.cs**
2) Proszę w projekcie *SchoolRegister.Model* stworzyć folder o nazwie ***DataModels***



3) Następnie w folderze ***DataModels*** proszę stworzyć plik o nazwie klasy którą chcemy stworzyć. Np. ***User.cs***



4) Następnie proszę zaimplementować tę klasę w poniższy sposób:

User.cs

SchoolRegister.Model > DataModels > User.cs > ...

```
1   using Microsoft.AspNetCore.Identity;
2   using System;
3
4   namespace SchoolRegister.Model.DataModels
5   {
        3 references
6       public class User : IdentityUser<int>
7       {
            0 references
8           public string FirstName { get; set; }
            0 references
9           public string LastName { get; set; }
            0 references
10          public DateTime RegistrationDate { get; set; }
11      }
12  }
```

5) Jak można zauważyć wystąpił problem z referencją do klasy bazowej oraz do zasobu ją zawierającego: W tym momencie należy zainstalować pakiet z serwisu NuGet o nazwie:

Microsoft.Extensions.Identity.Stores w wersji ***5.0.3***

Aby to zrobić należy najpierw w konsoli przejść do folderu projektu *SchoolRegister.Model:*

```
cd SchoolRegister.Model
```

```
PS B:\Projects\UCZELNIA\Studenci\Aplikacje_WWW_S> cd SchoolRegister.Model
PS B:\Projects\UCZELNIA\Studenci\Aplikacje_WWW_S\SchoolRegister.Model>
```

Następnie proszę wykonać w konsoli następujące polecenie:

```
dotnet add package Microsoft.Extensions.Identity.Stores --version 5.0.3
```

Po wykonaniu powyższych kroków i skompilowaniu całej solucji (należy najpierw wyjść do folderu ***Aplikacje_WWW_N***) projekt powinien się skompilować bez problemów.

6) Następnie proszę wykonać poniższe zadanie.

5. Zadanie

Proszę stworzyć klasy, ich składowe oraz relacje w oparciu o poniższy diagram UML. Należy każdą klasę stworzyć w osobnym pliku. **Po zakończonej pracy proszę użyć funkcji *Commit & Push* w Git Extensions (lub Commit a następnie Push w Git Shell), aby zapisać zmiany w swojej gałęzi na serwerze. Podczas wykonywania polecenia Push może nastąpić sytuacja w której zostaniecie Państwo poproszeni o poświadczenia. Zostaną one Państwu podane przez prowadzącego.**