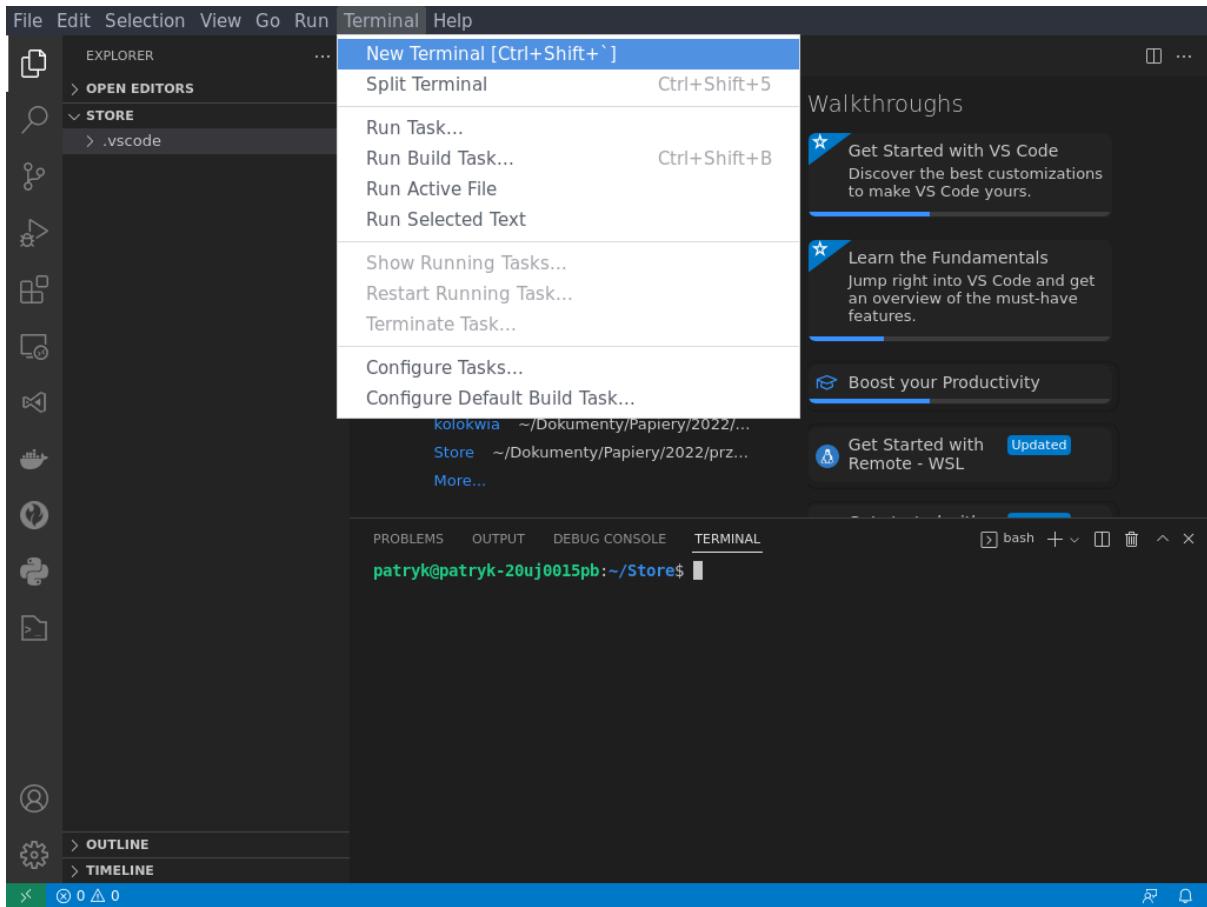


1. Tworzymy projekt

W terminalu `Start -> "cmd"` stwórz katalog dla projektu i uruchom w nim vs coda

```
mkdir Store  
code Store/
```

W vs code otwieramy terminal `Shift+Ctrl+`` z którego utworzymy szablon projektu.

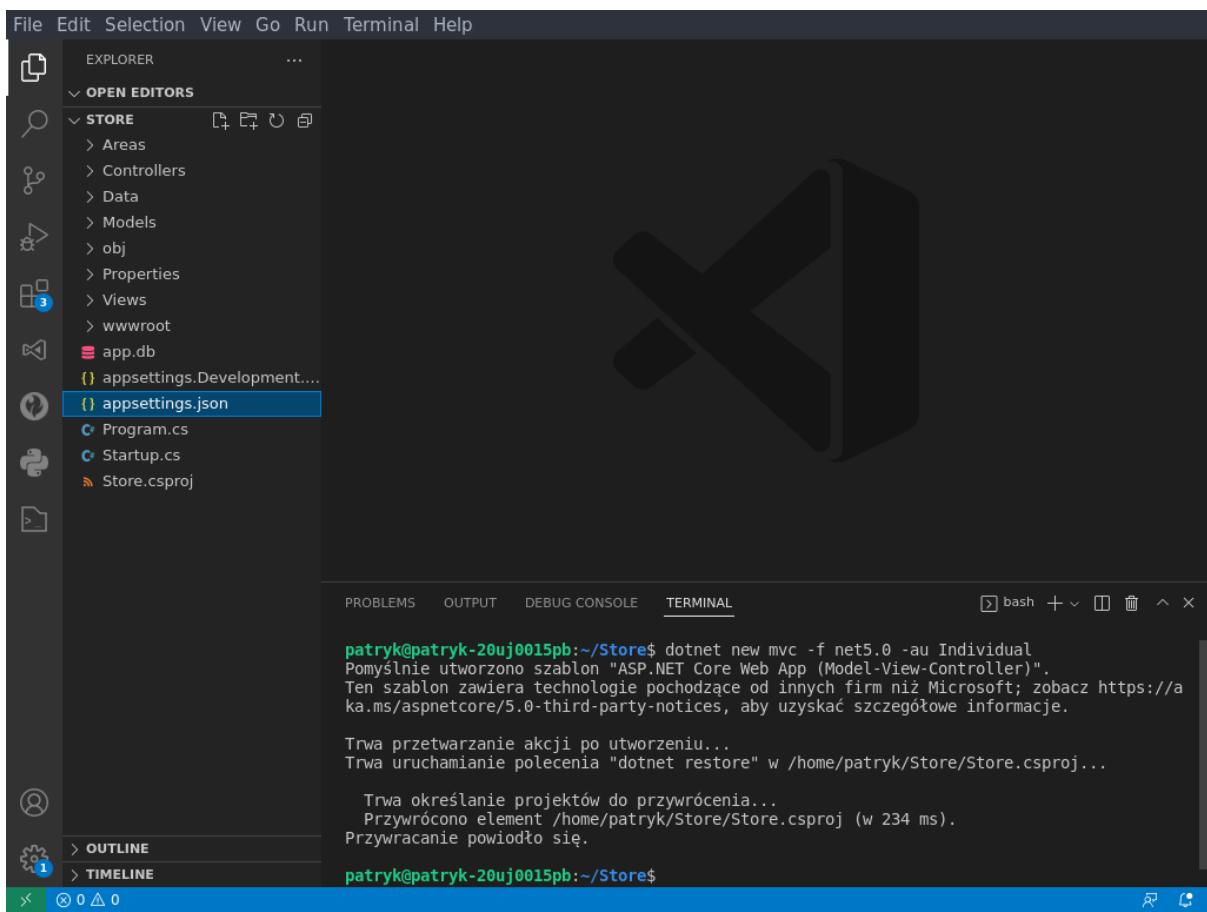


Teraz możemy utworzyć projekt naszej aplikacji web za pomocą komendy wpisanej w terminalu vs code.

```
dotnet new mvc -f net5.0 -au Individual
```

- parametrów tego polecania jest więcej i dostępne są na stronie:
<https://docs.microsoft.com/pl-pl/dotnet/core/tools/dotnet-new-sdk-templates#web-options> nam wystarczy jedynie szkielet.

Otrzymamy zestaw plików i katalogów.



Opis struktury

- **Areas** - katalog z wydzielonymi funkcjonalnościami w formie podstron z własnymi kontrolerami, widokami itd...
- **Controllers** - pliki z klasami obsługujące żądania od klientów. W kontrolerach zaszyta jest logika aplikacji.
- **Data** - kontekst bazy danych i skrypty migracyjne
- **Models** - pliki z klasami modeli danych, na których będzie operować nasza aplikacja. Katalog i jego zawartość odpowiada za strukturę danych, które będą przetwarzane przez aplikację webową.
- **obj** - (dla nas nieistotne) katalog, w którym będzie umieszczona skompilowana wersja projektu. Tu znajdują się pliki wykonywalne naszego projektu wraz z bibliotekami.
- **Properties** - (dla nas nieistotne) katalog z dodatkowymi plikami konfiguracyjnymi naszej aplikacji.
- **Views** - Katalog z widokami. W plikach widoków znajdują się szablony html, z których aplikacja buduje cały kontekst strony.
- **wwwroot** - katalog, w którym dodajemy statyczne zasoby, które będzie udostępniał serwer www. Przykładowo mogą to być skrypty javascript, pliki .css, obrazki, fonty.
- **app.db** - plik z bazą danych naszej aplikacji. Domyślnie jest to baza plikowa Sqlite.
- **appsetting.json** - główny plik konfiguracyjny naszej aplikacji, umieszczamy w nim informacje, które pozwolą uruchomić i działać naszej aplikacji w danym środowisku. Wyjaśniając. Skompilowana wersja aplikacji jest nieedytowalna, umieszczając ją np. na różnych serwerach musimy w jakiś sposób przekazać do aplikacji informacje np. adres serwera bazy danych, czy serwera poczty i właśnie robimy to poprzez pliki

konfiguracyjne.

- **Program.cs** - jest to plik z metodą **main** czyli metodą, która jako pierwsza jest uruchamiana w naszej aplikacji.
- **Startup.cs** - plik z klasą, której zadanie jest skonfigurowanie całej struktury naszej aplikacji podczas uruchomienia. W niej wiążane są wszystkie elementy (moduły) naszej aplikacji w całość. Bardzo istotny plik dla naszej aplikacji.

2.Uruchomienie projektu.

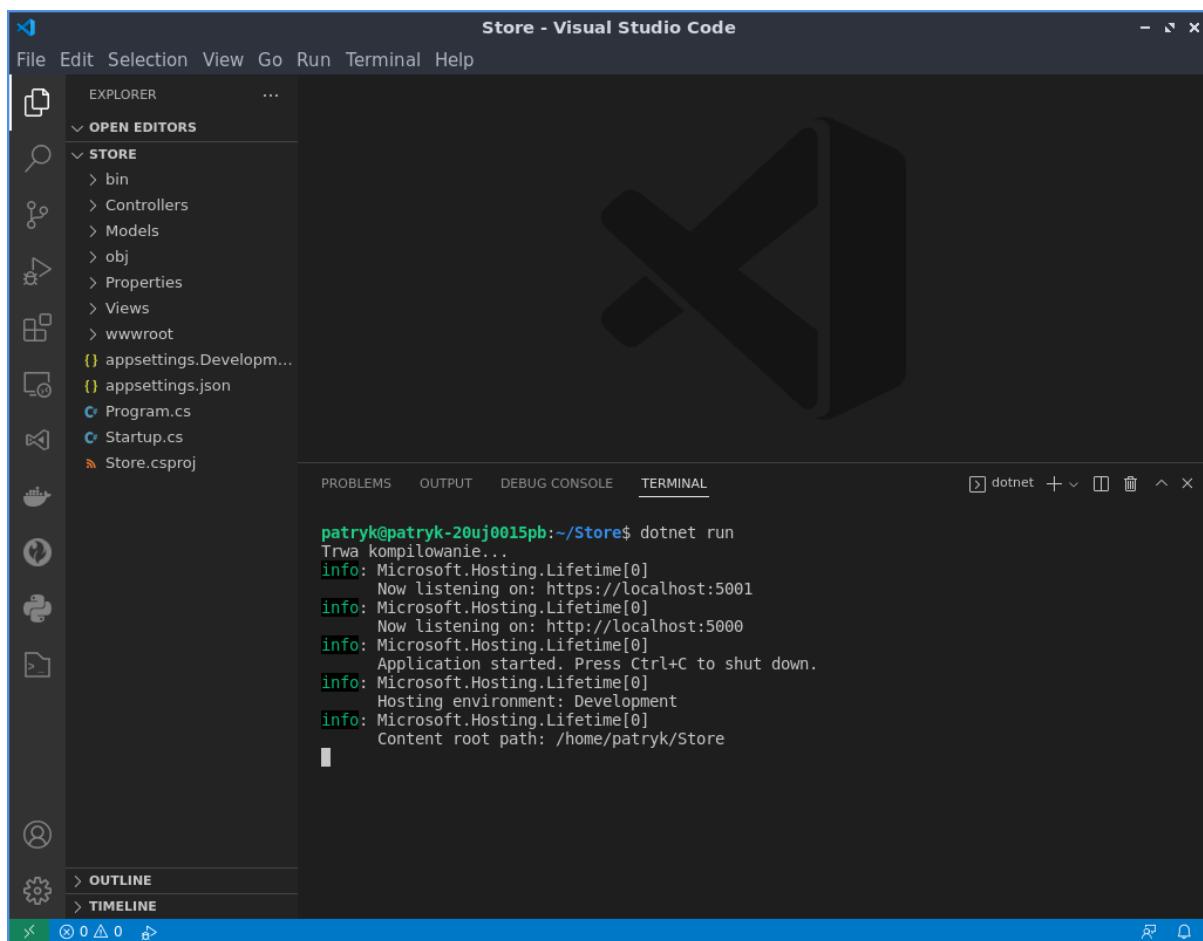
Projekt można uruchomić na kilka sposobów. Pamiętać trzeba, że w trakcie uruchomienia projektu aplikacja webowa otwiera i zajmuje porty, których nie może współdzielić z innymi aplikacjami. Czyli możemy odpalić jedynie jedną instancję na danym porcie. Domyślnie nasz projekt zajmie port :5000 dla http oraz :5001 dla https. Domyślnie protokół http obsługiwany jest na porcie 80, a https (komunikacja szyfrowana) na porcie 443 .

1. Uruchomienie z wiersza poleceń.

Wpisujemy polecenie w terminalu.

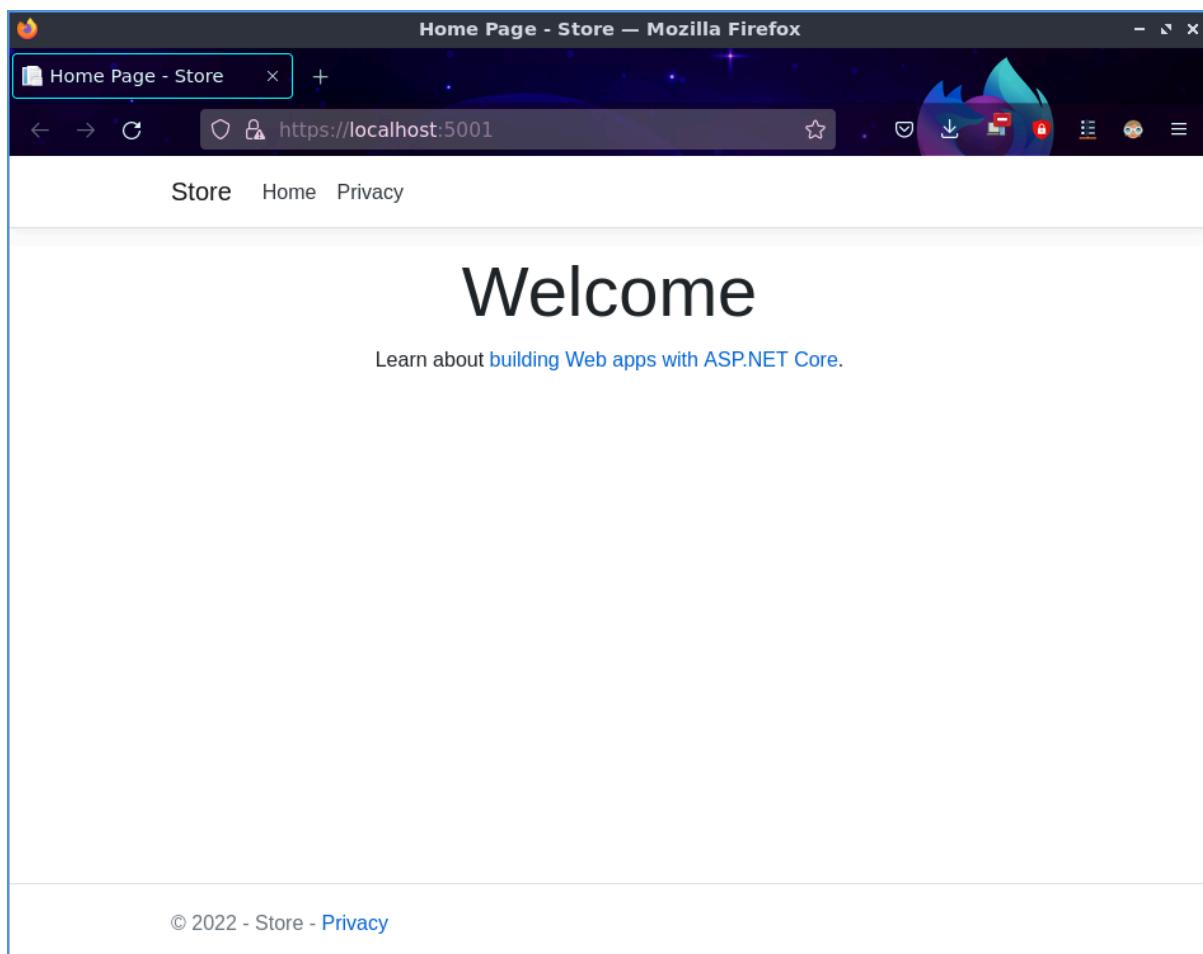
```
dotnet run
```

Po wykonaniu pojawi nam się informacja o adresie, na którym nasłuchuje serwer z naszą aplikacją. Możemy przejść do przeglądarki kliknięciem na adres z klawiszem Ctrl.



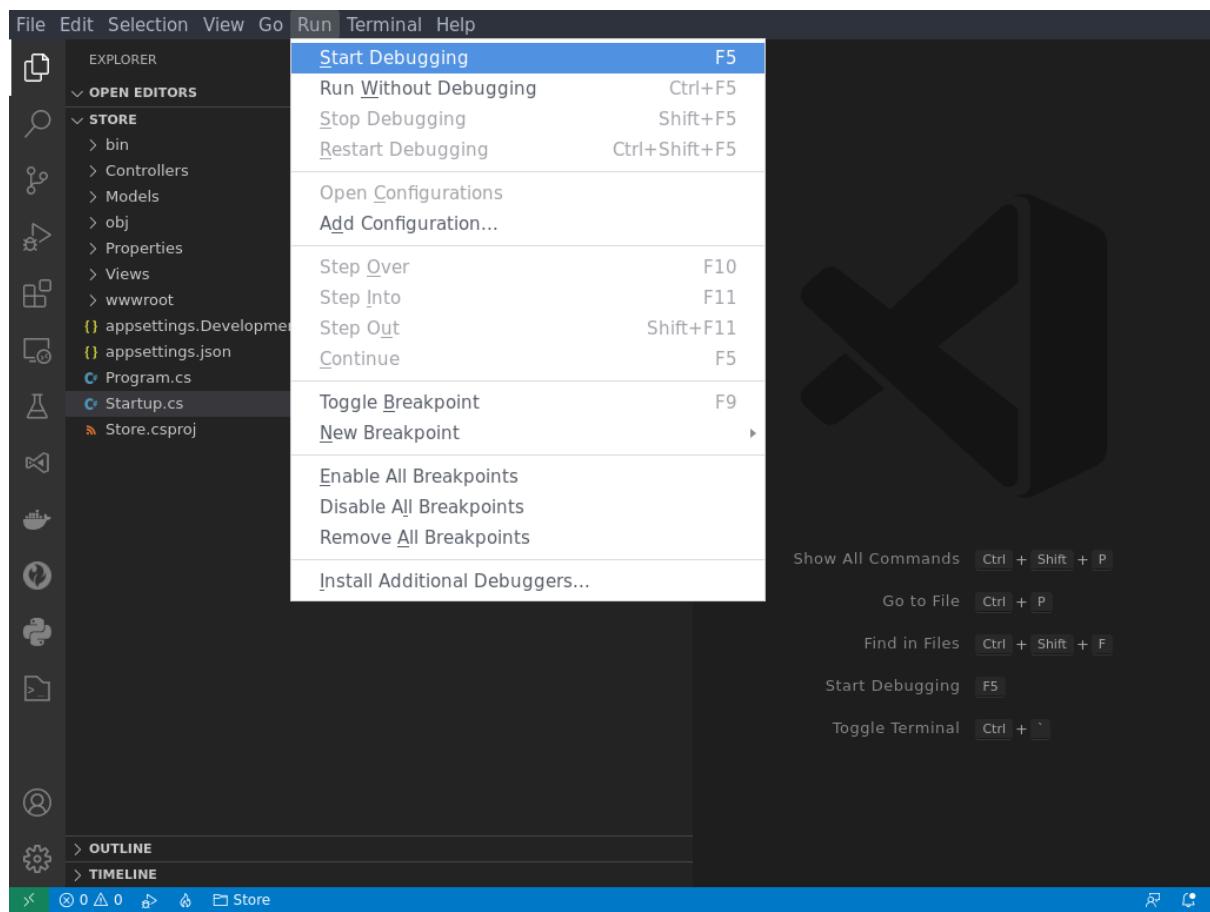
Widok aplikacji w przeglądarce:

- Nasza aplikacja praktycznie nie ma treści, jest tylko pustym szkieletem.
- U góry mamy jedynie link, który odnosi się do kontrolera HomeController w katalogu Controllers

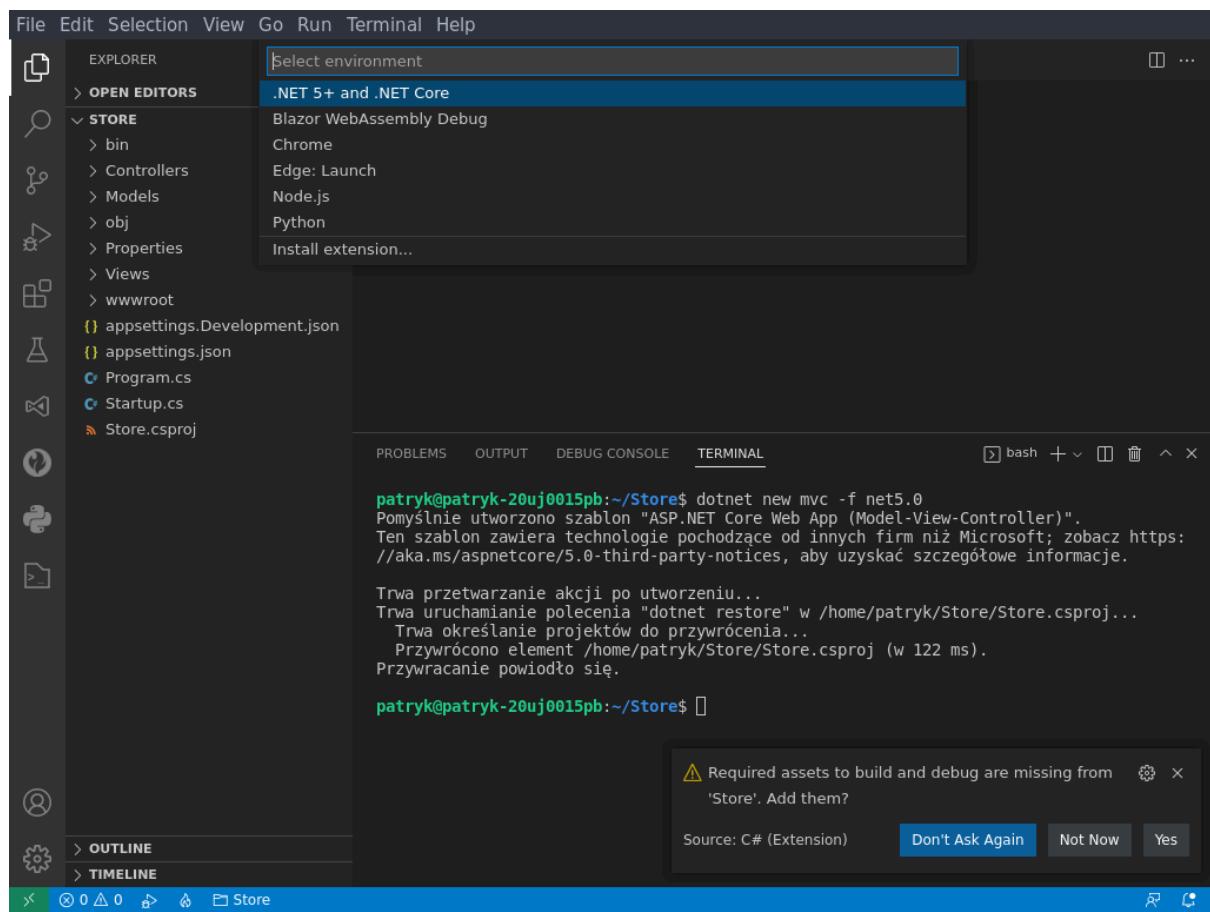


2. Uruchomienie w trybie z debugowaniem

Tryb ten pozwala debugować aplikację na żywo, zatrzymywać wykonywanie programu w miejscach ustawionych pułapek i podglądać oraz zmieniać zmienne w pamięci.
Uruchomić można za pomocą przycisku **F5**



Przy pierwszym uruchomieniu poprosi o wybór environmentu .Net Core.



Utworzy to nam dodatkowe uruchomieniowe pliki konfiguracyjne.

```
// Use IntelliSense to learn about possible attributes.  
// Hover to view descriptions of existing attributes.  
// For more information, visit: https://go.microsoft.com/fwlink/?LinkID=703952  
"version": "0.2.0",  
"configurations": [  
    {  
        "name": ".NET Core Launch (web)",  
        "type": "coreclr",  
        "request": "launch",  
        "preLaunchTask": "build",  
        "program": "${workspaceFolder}/bin/Debug/net5.0/Store.dll",  
        "args": [],  
        "cwd": "${workspaceFolder}",  
        "stopAtEntry": false,  
        "serverReadyAction": {  
            "action": "openExternally",  
            "pattern": "\\\bNow listening on:\\\$+(https?://\\\\S+)"  
        },  
        "env": {  
            "ASPNETCORE_ENVIRONMENT": "Development"  
        },  
        "sourceFileMap": {  
            "/Views": "${workspaceFolder}/Views"  
        }  
    },  
    {  
        "name": ".NET Core Attach",  
        "type": "coreclr",  
        "request": "attach"  
    }  
]
```

Jeśli konfiguracja została utworzona, możemy ponownie uruchomić aplikację w trybie debugowania, która tym razem się uruchomi.

W **vs code** przełącz się w trybie debugowania i będzie zatrzymywał wątek w trakcie wykonywania kodu w punktach oznaczonych pułapkami (**czerwone kropki** przed numeracją linii kodu).

3. Debugowanie: przykład

W poniższym przykładzie pułapka została ustawiona na metodzie **Index** w klasie **HomeController**

```
File Edit Selection View Go Run Terminal Help
EXPLORER ... Controllers > HomeController.cs ●
OPEN EDITORS 1 UNSAVED launch.json HomeController.cs
STORE ... 2 references
.<code>.vscode 12 public class HomeController : Controller
launch.json 13 {
tasks.json 14     1 reference
> bin 15     private readonly ILogger<HomeController> _logger;
Controllers 16     0 references
 HomeController.cs 17         public HomeController(ILogger<HomeController> logger)
18             {
19                 _logger = logger;
20             }
21     0 references
22     public IActionResult Index()
23     {
24         return View();
25     }
26     0 references
27     public IActionResult Privacy()
28         {
29             return View();
30         }
31     [ResponseCache(Duration = 0, Location = ResponseCacheLocation
0 references
32     public IActionResult Error()
33     {
34         return View(new ErrorViewModel { RequestId = Activity.Cu
35     }
36 }
```

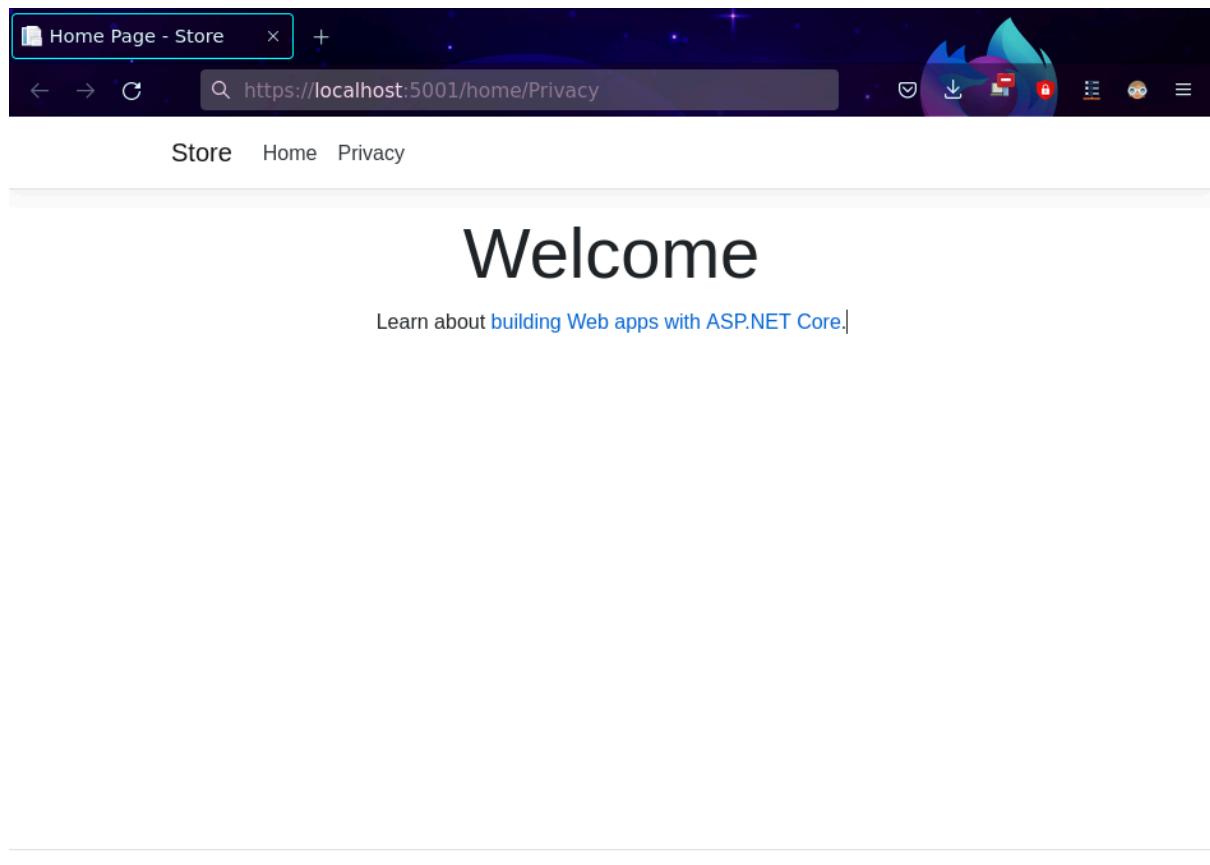
PROBLEMS OUTPUT DEBUG CONSOLE ... Filter (e.g. text, **/*.ts, !**/node_modules/...) ✎ ⌂ ⌄ ⌁

No problems have been detected in the workspace.

Ln 28, Col 9 Spaces: 4 UTF-8 with BOM CRLF C Sharp ⌂ ⌄

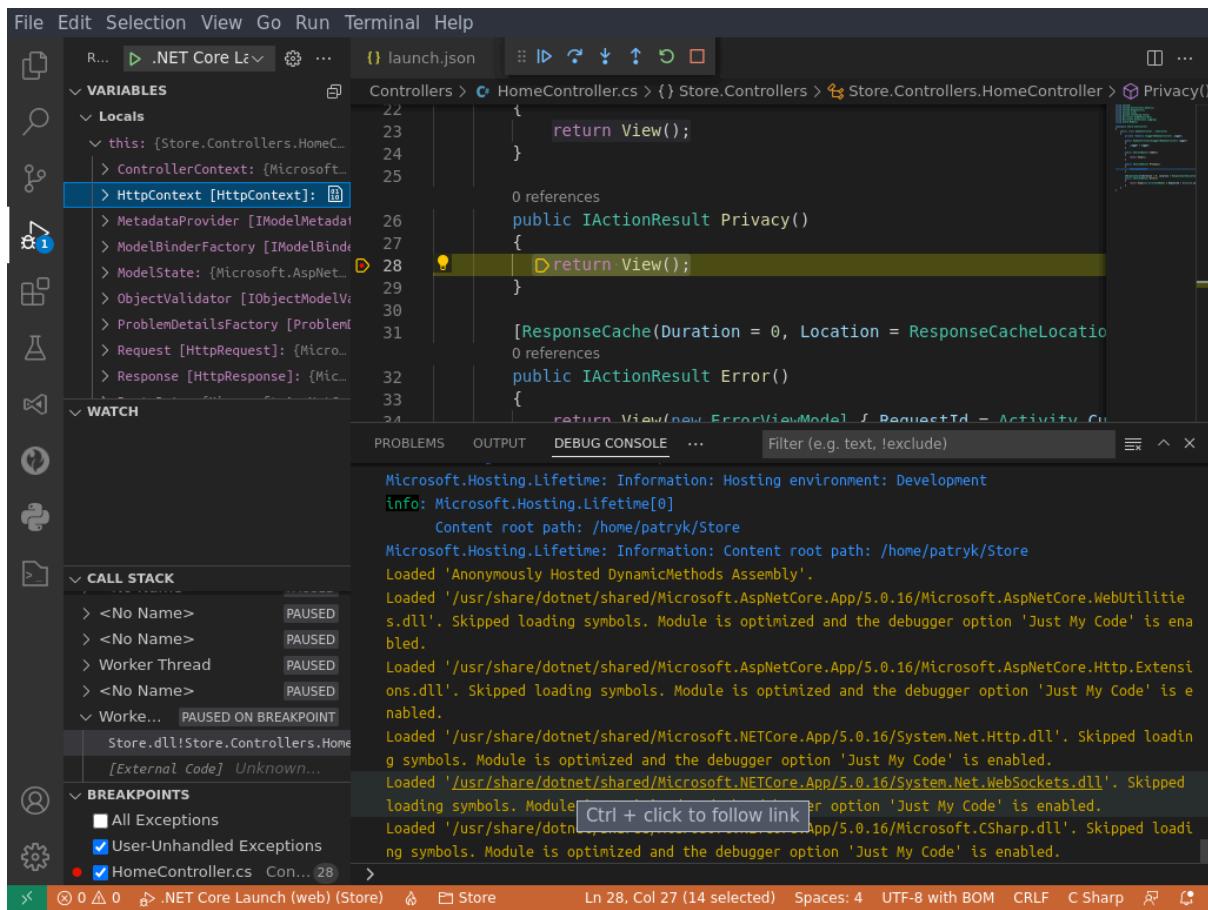
Aplikacje uruchamiamy w trybie debugowania **F5** i automatycznie uruchamia nam się przeglądarka na adresie domyślnym <https://localhost:5001>, który wywoła metodę **index** i kontroler **HomeController**.

Jeśli wpiszemy adres na <https://localhost:5001/home/privacy> uruchomi się metoda **Privacy** w klasie **HomeController**, w której umieszczona jest pułapka.



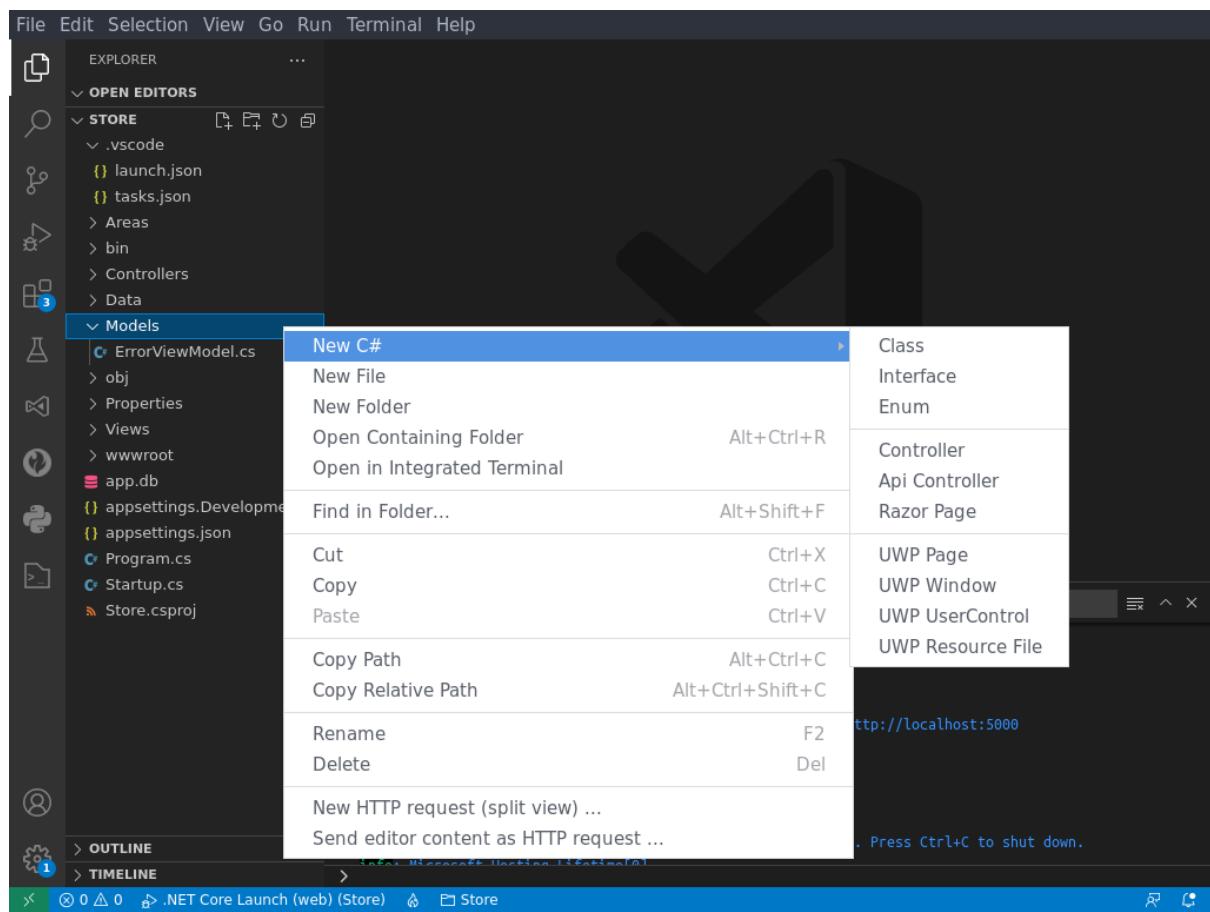
Efektem będzie zatrzymanie wątku w trakcie wykonywania żądania w miejscu pułapki. Z tego miejsca będziemy mogli podejrzeć wszystkie parametry obiektów uruchomionej aplikacji.

U góry mamy również interfejs, który może zwolnić wątek i kontynuować wykonywanie kodu.



3. Model danych

Tworzenie aplikacji najlepiej zacząć od projektu modelu danych na jakich będzie operować nasza aplikacja. W tym celu w katalogu **Models** utworzymy kilka klas z uwzględnieniem przypisania atrybutów dla poszczególnych właściwości klas, tak aby później Entity Framework mógł wygenerować dla nas bazę.



W naszym przykładzie dodamy klasy modeli danych:

- **Employee** - reprezentująca dane pracownika wystawiającego faktury.
- **Customer** - reprezentująca dane klienta, który będzie mógł podglądać swoje faktury
- **Invoice** - reprezentującego faktury
- **InvoiceItem** - reprezentująca pozycje faktury.

The screenshot shows the Visual Studio Code interface with the following details:

- Explorer:** Shows the project structure under the 'STORE' folder. The 'Invoice.cs' file is selected.
- Code Editor:** Displays the 'Invoice.cs' code:

```

1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel.DataAnnotations;
4
5  namespace Store.Models
6  {
7      2 references
8      public class Invoice
9      {
10         [Key]
11         0 references
12         public int Id { get; set; }
13         0 references
14         public DateTime Date {get;set;}
15         0 references
16         public virtual IList<InvoiceItem> Items {get;set;}
17     }

```
- Terminal:** Shows the output of a 'dotnet build' command:

```

> Executing task: dotnet build /home/patryk/Store/Store.csproj /property:GenerateFullPaths=true /consoleLoggerParameters:NoSummary <
Microsoft (R) Build Engine 17.1.1+a02f73656 dla platformy .NET
Copyright (C) Microsoft Corporation. Wszelkie prawa zastrzeżone.

Trwa określanie projektów do przywrócenia...
Wszystkie projekty są aktualne na potrzeby przywrócenia.
Store -> /home/patryk/Store/bin/Debug/net5.0/Store.dll
Store -> /home/patryk/Store/bin/Debug/net5.0/Store.Views.dll

```

Kod /Store/Models/Employee.cs

```
namespace Store.Models
{
    public class Employee : IdentityUser
    {
        public DateTime EmploymentDate { get; set; }
        public string Position { get; set; }
    }
}
```

- **IdentityUser** - jest to klasa modelu użytkownika wbudowana w aspnet, która jednocześnie integruje się z wbudowanym mechanizmem sesji i rejestracji użytkowników. Dziedzicząc po niej model **Employee**, czyli nasz pracownik będzie mógł się zarejestrować i zalogować do naszej aplikacji.
<https://docs.microsoft.com/dotnet/api/microsoft.aspnetcore.identity.entityframeworkcore.identityuser?view=aspnetcore-1.1&viewFallbackFrom=aspnetcore-5.0>

Kod /Store/Models/Customer.cs

```
using System.Collections.Generic;
using Microsoft.AspNetCore.Identity;

namespace Store.Models
{
```

```

public class Customer : IdentityUser
{
    public string FullName { get; set; }
    public string Address { get; set; }
    public virtual IList<Invoice> Invoices { get; set; }
}
}

```

- Właściwości **virtual** nie będą odwzorowane na bazie danych, tylko dynamicznie wypełniane przez Entity Framework na podstawie innych modeli. Odwołanie do tego pola spowoduje wygenerowanie dodatkowego zapytania SQL.

Kod /Store/Models/InvoiceItem.cs

```

using System.ComponentModel.DataAnnotations.Schema;
using System.ComponentModel.DataAnnotations;
namespace Store.Models
{
    public class InvoiceItem
    {
        [Key]
        public int Id { get; set; }
        [Required]
        public string Name { get; set; }
        public float Price { get; set; }
        public float Amount { get; set; }
        public float VAT { get; set; }
        public int InvoiceId { get; set; }
        [ForeignKey("InvoiceId")]
        public virtual Invoice Invoice { get; set; }
    }
}

```

- **[Key]** - klucz główny. Pole będzie traktowane jako klucz główny po stronie bazy
- **[ForeignKey("InvoiceId")]** - wskazuje właściwość, w której znajduje się klucz obcy do **Invoice**. Gdy odwołamy się do właściwości **Invoice** EF dynamicznie wyszuka w bazie **Invoice**, którego klucz główny **[Key]** będzie zgodny z właściwością **InvoiceId**.

Kod /Store/Models/Invoice.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Store.Models
{

```

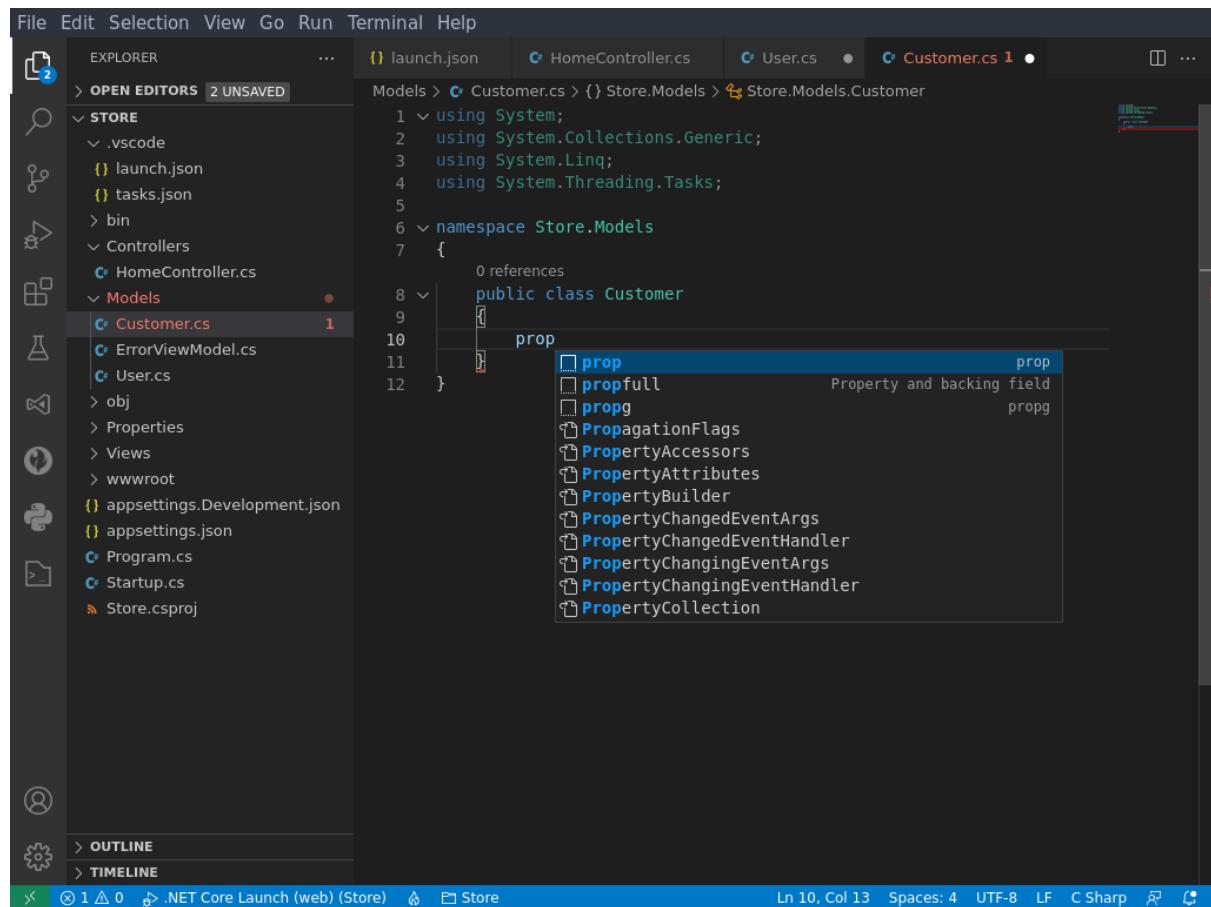
```

public class Invoice
{
    [Key]
    public int Id { get; set; }
    public DateTime Date { get; set; }
    public string IssuedForGuid { get; set; }
    public string IssuedByGuid { get; set; }
    public virtual IList<InvoiceItem> Items { get; set; }
    [ForeignKey("IssuedForGuid")]
    public virtual Customer IssuedFor { get; set; }
    [ForeignKey("IssuedByGuid")]
    public virtual Employee IssuedBy { get; set; }
}

```

- Odwołanie do lista **Items** wykona zapytanie do tabeli z listą **InvoiceItem** na bazie klucza obcego **[ForeignKey]** zdefiniowanego w **InvoiceItem** i klucza głównego **[Key]** w **Invoice**

Uwaga. Aby przyspieszyć pisanie możemy korzystać ze snippetów czyli drobnych szablonów uruchamianych w trakcie pisania. np. Wpisując **porp** następnie klikając **Tab** korzystamy z gotowego szablonu właściwości.



```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading.Tasks;
5
6  namespace Store.Models
7  {
8      public class Customer
9      {
10         public int MyProperty { get; set; }
11     }
12 }
```

1. Kontekst bazy danych

Posiadamy model danych ale potrzebujemy jeszcze obiektu, który będzie odwzorowywał **struktury tabel SQL-a**. Do tego celu tworzymy klasę dziedziczącą po klasie **DbContext**, która będzie odzwierciedlać schemat bazy danych i stanie się też punktem dostępowym do bazy danych poprzez ORM.

W naszym przypadku klasa taka już istnieje, została utworzona razem z projektem i wystarczy ją zmodyfikować.

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File Edit Selection View Go Run Terminal Help
- Explorer:** Shows a tree view of the project structure under the 'STORE' folder, including .vscode, Areas, bin, Controllers, Data, Migrations, Models, obj, Properties, Views, wwwroot, app.db, appsettings.Development.json, appsettings.json, Program.cs, Startup.cs, and Store.csproj.
- Code Editor:** Displays the `ApplicationDbContext.cs` file with the following code:

```

1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4  using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
5  using Microsoft.EntityFrameworkCore;
6
7  namespace Store.Data
8  {
9      public class ApplicationDbContext : IdentityDbContext
10     {
11         public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
12             : base(options)
13         {
14         }
15     }
16 }
```
- Terminal:** Shows the command `dotnet build /home/patryk/Store/Store.csproj /property:GenerateFullPaths=true /consoleLoggerParameters:NoSummary <` and the Microsoft Build Engine output.
- SQLite Explorer:** Shows a connection to the `app.db` database.
- Bottom Status Bar:** Shows .NET Core Launch (web) (Store), Store, Ln 9, Col 38 (20 selected), Spaces: 4, UTF-8 with BOM, CRLF, C Sharp, and a refresh icon.

- **DbContext, IdentityDbContext** znajdują się w namespace-ie **Microsoft.EntityFrameworkCore**, jeśli po dołączeniu namespaca klasa jest dalej niewidoczna to najprawdopodobniej będzie trzeba dodać referencję do brakującej biblioteki za pomocą komendy:

```
dotnet add package Microsoft.EntityFrameworkCore -v 5.0.0
```

Trzeba zwrócić uwagę na wersję biblioteki aby była zgodna z wersją net framework używaną w projekcie np., wersja 5..* zgodna jest z net. 5.0. Trzeba pamiętać aby wszystkie dodawane biblioteki były również w zbliżonej wersji np. 5.0.3*

Kod /Store/Data/ApplicationDbContext.cs

```

using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using Store.Models;

namespace Store.Data
{
    public class ApplicationDbContext : IdentityDbContext
    {
        public DbSet<Invoice> Invoices { get; set; }
        public DbSet<InvoiceItem> InvoiceItems { get; set; }
    }
}
```

```

    public
    ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    { }

    protected override void OnModelCreating(ModelBuilder builder)
    {
        base.OnModelCreating(builder);
        builder.Entity<IdentityUser>()
            .ToTable("AspNetUsers")
            .HasDiscriminator<string>("UserType")
            .HasValue<Employee>("Employee")
            .HasValue<Customer>("Customer");
    }
}
}
}

```

- Każda właściwość typu **DbSet<>** będzie traktowana przez EF jako tabela. My będziemy odwoływać się do tych właściwości jak do zwykłych list, natomiast EF w tle będzie generować zapytania do bazy i zwracać rezultaty już zmapowane na obiekty modeli.
- W naszym przypadku wprost dodajemy dwie tabele dla modeli **Invoices** i **IvoiceItems**, natomiast z dziedziczonej klasy **IdentityDbContext** dochodzi nam dodatkowa tabela **Users**.
- Natomiast modele **Employee** i **Customer** z powodu, że dziedziczą po modelu **IdentityUser** łączymy do jednej tabeli i będziemy je rozróżniać poprzez dodatkową kolumnę w tabeli **UserType**. Czyli oba modele w bazie będą jedną tabelą **AspNetUsers** i jednocześnie będą zintegrowane z mechanizmem logowania z aspnet.

2. Migracja

Posiadając gotowy kontekst możemy przeprowadzić migrację. W naszym przypadku to będzie już kolejna migracją ponieważ pierwsza została wykonana wraz z utworzeniem projektu na potrzeby systemu autentykacji użytkowników.

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Explorer:** Shows the project structure for "STORE".
- Code Editor:** Displays the file "00000000000000_CreatIdentitySchema.cs" which contains C# code for Entity Framework Core migrations. The code defines two tables: "AspNetRoles" and "AspNetUsers".
- Terminal:** Shows the message "Terminal will be reused by tasks, press any key to close it."
- Bottom Status Bar:** .NET Core Launch (web) (Store), Store, Ln 1, Col 1, Spaces: 4, UTF-8 with BOM, CRLF, C Sharp.

Uwaga. Gdyby komenda `dotnet-ef` nie była rozpoznawana to musimy zainstalować pakiet narzędzi `dotnet-ef`.

```
dotnet tool install --global dotnet-ef
```

Aby wykonać migrację wystarczy wpisać komendę:

```
dotnet-ef migrations add StoreModel
```

- `StoreModel` - to unikatowa nazwa dla migracji, każda migracja powinna nazywać się inaczej.
- migracja tworzy skrypt dla bazy danych w oparciu o poprzednie migracje i zmiany jakie dokonaliśmy w kontekście bazy danych. Czyli skrypt nie tworzy bazy od nowa ale tylko ją modyfikuje do aktualnych zmian w modelach i kontekście bazy danych.

The screenshot shows the Visual Studio Code interface. The left sidebar contains the Explorer, Outline, Timeline, and SQLite Explorer sections. The code editor displays the file `20220514064212_StoreModel.cs` which contains C# migration code for creating a `StoreModel`. The terminal at the bottom shows the command `dotnet ef migrations add StoreModel` being run, with the output indicating a successful build. The SQLite Explorer section shows a database named `app.db`.

- W naszym przypadku mamy już dwa skrypty migracyjne, pierwszy który tworzy bazę dla systemu autentykacji użytkowników i drugi który go modyfikuje pod modele **Employee** i **Customer** oraz dodaje nowe tabele **Invoices** i **IvoicelItems**.

3. Aktualizacja bazy

Mając skrypty migracyjne możemy zaktualizować bazę. W naszym przypadku chcemy dodać do niej jedynie zmiany aby była zgodna z kodem kontekstu i modeli. Mechanizmu aktualizacji bazy używamy również w przypadku przenoszenia projektu na serwer www i zainicjowania bazy danych pod naszą aplikację.

Do przeniesienia zmian na bazę wystarczy wywołać linijkę:

```
dotnet-ef database update
```

The screenshot shows the Visual Studio Code interface. The code editor displays C# migration code for Entity Framework. The terminal window shows the command `dotnet-ef database update` being run, with a message indicating that the Entity Framework tools version is older than the runtime.

```

migrationBuilder.AddColumn<string>(
    name: "Position",
    table: "AspNetUsers",
    type: "TEXT",
    nullable: true);

migrationBuilder.AddColumn<string>(
    name: "UserType",
    table: "AspNetUsers",
    type: "TEXT",
    nullable: false,
    defaultValue: "");

migrationBuilder.CreateTable(
    name: "Invoices",
    columns: table => new
    {
        Id = table.Column<int>(type: "INTEGER", nullable: false).Annotation("Sqlite:Autoincrement", true),
        Date = table.Column<DateTime>(type: "TEXT", nullable: false),
        IssuedForGuid = table.Column<string>(type: "TEXT", nullable: false),
        IssuedByGuid = table.Column<string>(type: "TEXT", nullable: false)
    },
);

```

- zmiany zostaną wyeksportowane na bazę powiązaną projektem poprzez konfigurację, która znajduje się w **Startup.cs** oraz **appsettings.json**.
- w **Startup.cs** znajduje się rejestracja naszego kontekstu bazy danych jako serwis oraz wskazany jest typ bazy.

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlite(
            Configuration.GetConnectionString("DefaultConnection")));
    services.AddDatabaseDeveloperPageExceptionFilter();

    services.AddDefaultIdentity<IdentityUser>(options =>
        options.SignIn.RequireConfirmedAccount = true)
        .AddEntityFrameworkStores<ApplicationDbContext>();
    services.AddControllersWithViews();
}

```

- w **appsettings.json** natomiast jest konfiguracja, którą możemy modyfikować już po skompilowaniu projektu.

```
{
  "ConnectionStrings": {
    "DefaultConnection": "DataSource=app.db;Cache=Shared"
  },
  "Logging": {

```

```
"LogLevel": {  
    "Default": "Information",  
    "Microsoft": "Warning",  
    "Microsoft.Hosting.Lifetime": "Information"  
}  
,  
"AllowedHosts": "*"  
}
```

4. Generowanie kodu

W naszym przypadku na podstawie modeli danych możemy wygenerować pozostałą najważniejszą część kodu (kontrolery, widoki) narzędziem **dotnet-aspnet-codegenerator**.

Gdyby narzędzie nie było dostępne (terminal) należy je doinstalować.

```
dotnet tool install -g dotnet-aspnet-codegenerator
```

Prawdopodobnie będzie wymagane dodanie referencji do projektu

```
dotnet add package Microsoft.VisualStudio.Web.CodeGeneration.Design -v  
5.0.0
```

Jeśli wszystko zostało poprawnie zainstalowane i projekt się kompliluje, to możemy przystąpić do generowania gotowych kontrolerów za pomocą narzędzia **dotnet-aspnet-codegenerator**.

Terminal:

```
dotnet-aspnet-codegenerator controller -udl -dc ApplicationDbContext  
-sqlite -outDir Controllers -m Invoice -name InvoiceController
```

- parametr **-m** określa klasę modelu danych, w tym przypadku odnosi się do klasy **Store.Models.Invoice.cs**
- parametr **-name** określa nazwę tworzonego kontrolera w przykładzie jest to **InvoiceController**
- opis reszty parametrów oraz możliwości narzędzia znajdziecie na stronie:
<https://docs.microsoft.com/aspnet/core/fundamentals/tools/dotnet-aspnet-codegenerator?view=aspnetcore-5.0>

The screenshot shows the Visual Studio Code interface during the creation of an ASP.NET Core application. The Explorer sidebar on the left lists project files like .vscode, Areas, bin, Controllers (HomeController.cs, InvoiceController.cs), Data (Migrations, ApplicationDbContext.cs), Models (Customer.cs, Employee.cs, ErrorViewModel.cs, Invoice.cs, InvoiceItem.cs), Properties, Views (Home, Invoice), and Shared. The Editor tab at the top shows the code for `Invoice.cs` under the `Store.Models` namespace. The code defines a `Invoice` class with properties `Id`, `Date`, `IssuedForGuid`, and `IssuedByGuid`. The `Invoice` item is selected in the editor. The Terminal tab at the bottom displays the command-line output of the dotnet-aspnet-codegenerator command, showing the generation of controllers, views, and their associated HTML partials for the `Invoice` model.

```
patryk@patryk-20uj0015pb:~/Store$ dotnet-aspnet-codegenerator controller -m Invoice -dc ApplicationDbContext -sqlite -name InvoiceController -outDir Controllers
Building project ...
Finding the generator 'controller'...
Running the generator 'controller'...
Attempting to compile the application in memory.
Attempting to figure out the EntityFramework metadata for the model and DbContext: 'Invoice'.
Added Controller : '/Controllers/InvoiceController.cs'.
Added View : /Views/Invoice/Create.cshtml
Added View : /Views/Invoice/Edit.cshtml
Added View : /Views/Invoice/Details.cshtml
Added View : /Views/Invoice/Delete.cshtml
Added View : /Views/Invoice/Index.cshtml
RunTime 00:00:12.75
patryk@patryk-20uj0015pb:~/Store$
```

- Komenda tworzy klasę kontroler w katalogu `Controllers`, wraz z podstawowymi akcjami do edycji i przeglądania danych.
- Tworzone są również Widoki do akcji, które generują kod html.

Możemy teraz wykonać tą samą operację dla modeli **Customer**, **Invoice** i **InvoiceItem**.

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading.Tasks;
5  using Microsoft.AspNetCore.Mvc;
6  using Microsoft.AspNetCore.Mvc.Rendering;
7  using Microsoft.EntityFrameworkCore;
8  using Store.Data;
9  using Store.Models;
10 namespace Store.Controllers
11 {
12     public class InvoiceController : Controller
13     {
14         private readonly ApplicationDbContext _context;
15
16         public InvoiceController(ApplicationDbContext context)
17         {
18             _context = context;
19         }
20
21         // GET: Invoice
22         public async Task<IActionResult> Index()
23         {
24             var applicationDbContext = _context.Invoices.Include(i =>
25                 i
26             );
27             return View(await applicationDbContext.ToListAsync());
28         }
29     }
30 }
```

RunTime 00:00:10.10
patrick@patrick-20uj0015pb:~/Store\$

4. Kontrolery

Kontrolery są klasami obsługującymi żądania użytkowników. Domyślnie projekt posiada jedną klasę HomeController. Adres URL, którym odwołujemy się do strony jest parsowany domyślnie.

```
example.com/{controller}/{action}/{id?}
```

- **controller** - odwołuje się do nazwy klasy kontrolera
np. home -> HomeController
- **action** - odwołuje się do nazwy metody w klasie kontrolera
np. index -> HomeController.Index
- **id?** - paruje parametr id dla metody, znak ? oznacza, że nie jest wymagany

Parsery URL są zdefiniowane w **Startup.cs**

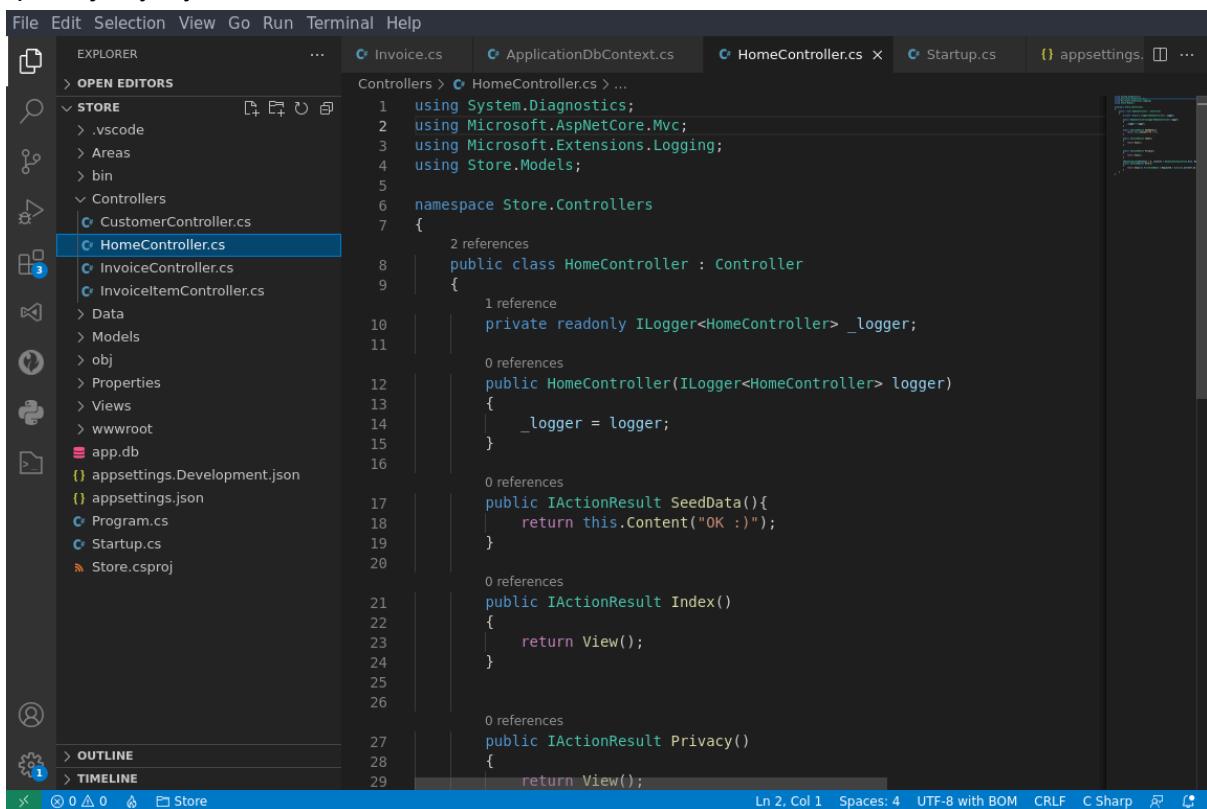
```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}");
    endpoints.MapRazorPages();
```

```
} );
```

1. Akcje

Akcje w kontrolerach to tak naprawdę metody, które można wywołać wpisując w przeglądarce url odpowiadający nazwie kontrolera oraz jego metody.

Dla przykładu stworzymy sobie nową metodę **SeedData** w klasie **HomeController** i ją spróbujemy wywołać.



The screenshot shows the Visual Studio code editor with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Explorer:** Shows the project structure with files like Invoice.cs, ApplicationDbContext.cs, HomeController.cs (selected), Startup.cs, and appsettings.json.
- Code Editor:** Displays the HomeController.cs code:

```
using System.Diagnostics;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using Store.Models;

namespace Store.Controllers
{
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;

        public HomeController(ILogger<HomeController> logger)
        {
            _logger = logger;
        }

        public IActionResult SeedData()
        {
            return this.Content("OK :)");
        }

        public IActionResult Index()
        {
            return View();
        }

        public IActionResult Privacy()
        {
            return View();
        }
    }
}
```
- Status Bar:** Ln 2, Col 1, Spaces: 4, UTF-8 with BOM, CRLF, C Sharp, etc.

Kod: /Store/Controllers/HomeController.cs

```
using System.Diagnostics;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using Store.Models;

namespace Store.Controllers
{
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;

        public HomeController(ILogger<HomeController> logger)
        {
            _logger = logger;
        }

        public IActionResult SeedData()
        {
            return this.Content("OK :)");
        }

        public IActionResult Index()
        {
            return View();
        }

        public IActionResult Privacy()
        {
            return View();
        }
    }
}
```

```

    }

    // -----
    public IActionResult SeedData(int? arg1, int? arg2){
        return this.Content($"OK arg1={arg1}, arg2={arg2}");
    }
    // -----


    public IActionResult Index()
    {
        return View();
    }

    public IActionResult Privacy()
    {
        return View();
    }

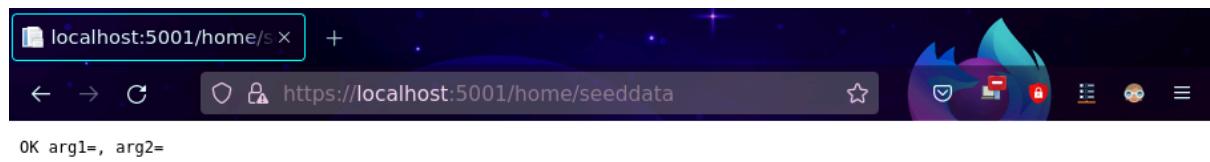
    [ResponseCache(Duration = 0, Location =
ResponseCacheLocation.None, NoStore = true)]
    public IActionResult Error()
    {
        return View(new ErrorViewModel { RequestId =
Activity.Current?.Id ?? HttpContext.TraceIdentifier });
    }
}

```

- Dopuszczamy jedna akcje **SeedData**
- Akcja przyjmuje dwa parametry arg1, arg2 typu int lub null
- Akcja musi zwracać jakiś kontekst, który zostanie wysłany do przeglądarki, najczęściej zwracany jest widok czyli szablon, który generuje kod html, w naszym przypadku obecnie to stała wartość w postaci stringa.

Żeby teraz wywołać naszą metodę musimy uruchomić aplikację i w przeglądarce wpisać url

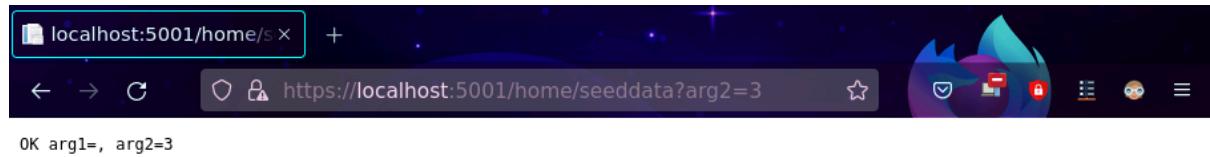
localhost:5001/home/seeddata



Wpisując :

`localhost:5001/home/seeddata?arg2=3`

przekażemy wartość 3 do parametru **arg2**



2. Seriwisy

Servisy jak było wspomniane wcześniej są to klasy rejestrowane jako tzw. globalne api w naszej aplikacji. Servisy są w jednym miejscu (**Startup.cs**) konfigurowane i dołączane do aplikacji, a potem możemy z nich dowolnie korzystać.

kod. /Store/Startup.cs

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ApplicationContext>(options =>
        options.UseSqlite(
            Configuration.GetConnectionString("DefaultConnection")));
    services.AddDatabaseDeveloperPageExceptionFilter();

    services.AddDefaultIdentity<IdentityUser>(options =>
        options.SignIn.RequireConfirmedAccount = true)
        .AddEntityFrameworkStores<ApplicationContext>();
    services.AddControllersWithViews();
}
```

- **services.AddDbContext<ApplicationContext>** - dodaje kontekst bazy danych jako serwis więc będzie dostępny globalnie również w kontrolerach
- **services.AddDefaultIdentity<IdentityUser>** - dodaje servis obsługi sesji logowania, rejestracji itp..., jednocześnie ten serwis wykorzystuje serwis kontekstu bazy danych.
- serwisami są również same kontrolery, a pomiędzy serwisami następuje płynne przekazywanie ich instancji.

I teraz chcemy korzystać z kontekstu bazy danych (ApplicationContext), oraz menadżera użytkowników (UserManager) w HomeController, to musimy je wskazać w konstruktorze kontrolera i przenieść do jakiś pól w klasie, żeby użyć ich później w akcji.

Fragment: /Store/Controllers/HomeController.cs

```
private ApplicationContext db;
private UserManager<IdentityUser> userManger;
public HomeController(ILogger<HomeController> logger,
                      ApplicationDbContext db,
                      UserManager<IdentityUser> userManger)
{
    _logger = logger;
    this.db = db;
    this.userManger = userManger;
}
```

W ten sposób w trakcie rejestracji serwisów aspnet powiąże nam wzajemnie servisy, po typach argumentów przekazywanych do konstruktorów.

Uwaga. Jeśli Klasa **ApplicationContext**, **UserManager**, **IdentityUser** nie jest widoczna prawdopodobnie brakuje using, możemy go dodać z pomocą coda lub ręcznie dodając linie **using Store.Data; itd...**:

```
File Edit Selection View Go Run Terminal Help
Invoice.cs ApplicationController.cs HomeController.cs 1 x Startup.cs appsettings.json ...
s > HomeController.cs > ()> Store.Controllers > HomeController > HomeController(logger, ApplicationDbContext db)
1  using System.Diagnostics;
2  using Microsoft.AspNetCore.Mvc;
3  using Microsoft.Extensions.Logging;
4  using Store.Models;
5
6  namespace Store.Controllers
7  {
8      public class HomeController : Controller
9      {
10         private readonly ILogger<HomeController> _logger;
11
12         public HomeController(ILogger<HomeController> logger, ApplicationDbContext db)
13             ApplicationDBContext - using Store.Data;
14             Generate type 'ApplicationDBContext' -> Generate class 'ApplicationDBContext' in new file
15             Generate type 'ApplicationDBContext' -> Generate class 'ApplicationDBContext'
16             Generate type 'ApplicationDBContext' -> Generate nested class 'ApplicationDBContext'
17
18         public IActionResult SeedData(int? arg1, int? arg2){
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL Filter (e.g. text, !exclude)
LN 12, COL 83 (20 selected) Spaces: 4 UTF-8 with BOM CRLF C Sharp
Loaded '/usr/share/dotnet/shared/Microsoft.NETCore.App/5.0.16/System.Transactions.Local.dll'. Skipped loading symbols. Module is optimized and the debugger option 'Just My Code' is enabled.
Loaded '/usr/share/dotnet/shared/Microsoft.NETCore.App/5.0.16/System.Net.Http.dll'. Skipped loading symbols. Module is optimized and the debugger option 'Just My Code' is enabled.
Loaded '/usr/share/dotnet/shared/Microsoft.NETCore.App/5.0.16/System.Net.WebSockets.dll'. Skipped loading symbols. Module is optimized and the debugger option 'Just My Code' is enabled.
Loaded '/usr/share/dotnet/shared/Microsoft.NETCore.App/5.0.16/System.Buffers.dll'. Skipped loading symbols. Module is optimized and the debugger option 'Just My Code' is enabled.
The program '[7759] Store.dll' has exited with code 0 (0x0).

```

3. Operacje na danych

Mamy podpięte serwisy i stworzoną akcję **SeedData** więc możemy wykonać pierwsze operacje na danych. Wsumie to stworzona metoda wykorzystana zostanie do wypełniania bazy danymi.

Zaczniemy od użytkowników, bo do nich dowiązane są inne modele. Użytkownicy są związani z aspnet więc najlepiej użyć managera zamiast próbować wpisywać ich ręcznie do bazy.

Kod: /Store/Controllers/HomeController.cs

```
using System.Collections.Generic;
using System.Threading.Tasks;
using System;
using System.Diagnostics;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using Store.Data;
using Store.Models;
```

```
using System.Linq;

namespace Store.Controllers
{
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;

        private ApplicationDbContext db;
        private UserManager<IdentityUser> userManger;
        public HomeController(ILogger<HomeController> logger,
            ApplicationDbContext db,
            UserManager<IdentityUser> userManger)
        {
            _logger = logger;
            this.db = db;
            this.userManger = userManger;
        }

        // -----
        public IActionResult SeedData(int? arg1, int? arg2)
        {
            if(db.Users.FirstOrDefault(u=>u.Id == "001") == null)
                userManger.CreateAsync(new Customer() {
                    Id = "001", UserName = "customer1@example.com",
                    EmailConfirmed = true }, "Ws9Pp59TjWW6kN8:").Wait();
            if(db.Users.FirstOrDefault(u=>u.Id == "002") == null)
                userManger.CreateAsync(new Customer() {
                    Id = "002", UserName = "customer2@example.com",
                    EmailConfirmed = true }, "Ws9Pp59TjWW6kN8:").Wait();
            if(db.Users.FirstOrDefault(u=>u.Id == "003") == null)
                userManger.CreateAsync(new Employee {
                    Id = "003", UserName = "employee@example.com",
                    EmailConfirmed = true }, "Ws9Pp59TjWW6kN8:").Wait();
            if(db.Invoices.FirstOrDefault(u=>u.Id == 1) == null){
                db.Add( new Invoice(){ Id = 1, Date = DateTime.Now,
                    IssuedByGuid = "003", IssuedForGuid="001",
                    Items = new List<InvoiceItem>(){
                        new InvoiceItem(){Id=1, Name="Product 1",
                            Amount=2f, Price=23.4f, VAT=23},
                        new InvoiceItem(){Id=2, Name="Product 2",
                            Amount=1.3f, Price=3.4f, VAT=8}
                    }
                });
            }
        }
    }
}
```

```

        } );
        db.SaveChanges();
    }

    return this.RedirectToAction("Index", "Customer");
}
// ----

public IActionResult Index()
{
    return View();
}

public IActionResult Privacy()
{
    return View();
}

[ResponseCache(Duration = 0, Location =
ResponseCacheLocation.None, NoStore = true)]
public IActionResult Error()
{
    return View(new ErrorViewModel { RequestId =
Activity.Current?.Id ?? HttpContext.TraceIdentifier });
}
}

```

- Metoda dodaje trzech użytkowników: customer1@example.com, customer2@example.com, employee@example.com z identycznym hasłem: Ws9Pp59TjWW6kN8:
- Przed dodaniem odpytywana jest baza, czy taki użytkownik przypadkiem już nie istnieje.
- **RedirectToAction("Index", "Customer")** - zwraca adres przekierowania na jaki przeglądarka automatycznie przejdzie (HTTP response codes = 303) <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>

Jeśli uruchomimy aplikację i w przeglądarce wpisujemy adres:

```
localhost:5001/home/seeddata
```

to wywołamy naszą metodę, a jeśli wszystko będzie ok to zwróci przeglądarce odpowiedź aby ta zmieniła adres. Przy wykonaniu się kodu przeglądarka przejdzie na adres:

```
localhost:5001/customer/index
```

Akcja tak naprawdę wywoła dwa żądania do serwera, a w efekcie zobaczymy listę modeli Customer z bazy danych.

FullName	Address	UserName	NormalizedUserName	Email	NormalizedEmail	EmailConfirmed	PasswordHash
customer1@example.com		CUSTOMER1@EXAMPLE.COM		customer1@example.com	CUSTOMER1@EXAMPLE.COM	<input checked="" type="checkbox"/>	AQAAAAEAAACcQ
customer2@example.com		CUSTOMER2@EXAMPLE.COM		customer2@example.com	CUSTOMER2@EXAMPLE.COM	<input checked="" type="checkbox"/>	AQAAAAEAAACcQ

Będziemy mogli jednocześnie się zalogować na dodanych użytkowników.

Log in

Use a local account to log in.

Email

There are no external authentication services configured. See this [article](#) about setting up this ASP.NET application to support logging in via external services.

Password

Remember me?

Log in

[Forgot your password?](#)

W tym momencie możemy również przejrzeć resztę kontrolerów, tylko że musimy wpisywać adresy ręcznie.

<https://localhost:5001/invoiceitem>

Name	Price	Amount	VAT	Invoice	
Product 1	23,4	2	23	1	Edit Details Delete
Product 2	3,4	1,3	8	1	Edit Details Delete

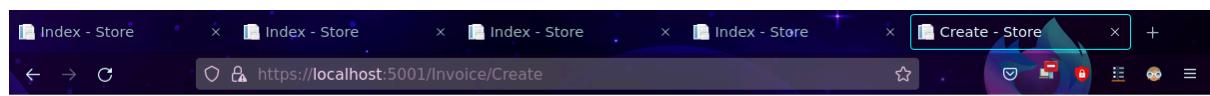
© 2022 - Store - [Privacy](#)

<https://localhost:5001/invoice>

Date	IssuedFor	IssuedBy	
14.05.2022 15:48:59	001	003	Edit Details Delete

© 2022 - Store - [Privacy](#)

Klikając **Create New** możemy nawet dodawać nowe pozycje do bazy, jednak jest to robione nielogicznie ponieważ kod wygenerowany został automatycznie i wymaga korekty.



Create

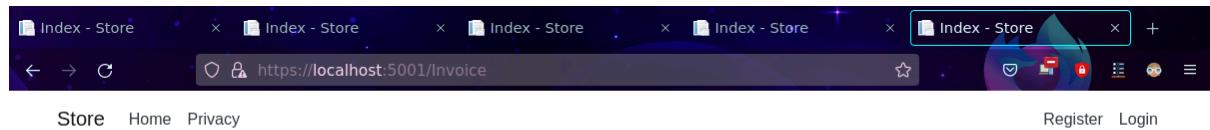
Invoice

Date

IssuedForGuid

IssuedByGuid

[Create](#)



Index

[Create New](#)

Date	IssuedFor	IssuedBy	
14.05.2022 15:48:59	001	003	Edit Details Delete
18.05.2022 01:21:00	002	003	Edit Details Delete

© 2022 - Store - [Privacy](#)

4. Podstawowa logika w kontrolerach

Jak widzimy dane są wyświetlane chaotycznie np. kontroler faktur wyświetla wszystkie faktury, nie można przejść pomiędzy danymi np. z klienta do jego listy faktur, z faktury do listy jej pozycji.

Aby uporządkować trochę przejrzystość danych spróbujemy wprowadzić hierarchię. Użytkownik będzie mógł najpierw wskazać klienta i wyświetlić jego faktury, następnie wskazać fakturę i wyświetlić jej pozycję.

Zaczniemy od **CustomerController** i zmienimy tylko linijkę kodu, żeby nie modyfikować na tym etapie widoków. Zmienimy sposób działania akcji **Details**.

Fragment: /Store/Controllers/CustomerController.cs

```

public async Task<IActionResult> Details(string id)
{
    if (id == null)
    {
        return NotFound();
    }

    var customer = await _context.Customer
        .FirstOrDefaultAsync(m => m.Id == id);
    if (customer == null)
    {
        return NotFound();
    }

    //return View(customer);
    return RedirectToAction("Index", "Invoice",
        new {CustomerId=customer.Id});
}

```

- Zamiast widoku akcja zwraca przekierowanie do kontrolera **Invoice** na metodę **Index** i przekaże parametr **CustomerId** odpowiadający użytkownika w którego klikniemy.

Następnie zmiany wprowadzamy w **InvoiceController** i zmienimy kodu akcji **Indeks**.

Fragment: /Store/Controllers/InvoiceController.cs

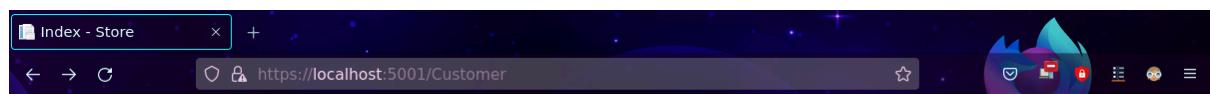
```

// GET: Invoice
public async Task<IActionResult> Index(string CustomerId)
{
    if(CustomerId != null)
        return View(await _context.Invoices
            .Where(i=>i.IssuedForGuid == CustomerId)
            .Include(i => i.IssuedBy).Include(i => i.IssuedFor)
            .ToListAsync());
    var applicationDbContext = _context.Invoices
        .Include(i => i.IssuedBy).Include(i => i.IssuedFor);
    return View(await applicationDbContext.ToListAsync());
}

```

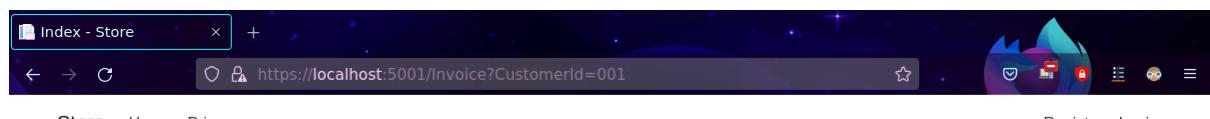
- metoda **Index** przyjmuje parametr **CustomerId** przekazywany z przekierowania z klasy **Customer**.
- w kodzie metody dopisano jeden warunek, który w przypadku gdy podano parametr **CustomerId** pobiera z bazy listę faktur wskazanego klienta i przekazuje ją do widoku.

Efekt: Po kliknięciu Details przeskakujemy do listy faktur.



ConcurrencyStamp	PhoneNumber	PhoneNumberConfirmed	TwoFactorEnabled	LockoutEnd	LockoutEnabled	AccessFailedCount	
02d271f0-3897-4ed5-b47a-8776c82ec1da	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	0	Edit Details Delete
1a560750-ed4b-4dc3-b67a-130d192619c5	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	0	Edit Details Delete

<https://localhost:5001/Customer/Details/001>



Store Home Privacy Register Login

Index

[Create New](#)

Date	IssuedFor	IssuedBy	
14.05.2022 15:48:59	001	003	Edit Details Delete

© 2022 - Store - [Privacy](#)

Jeszcze pozostała modyfikacja metody **Details** w **InvoiceController** i **Index** w **InvoiceIndexController**.

Fragment: /Store/Controllers/InvoiceController.cs

```
// GET: Invoice/Details/5
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }
```

```

var invoice = await _context.Invoices
    .Include(i => i.IssuedBy)
    .Include(i => i.IssuedFor)
    .FirstOrDefaultAsync(m => m.Id == id);
if (invoice == null)
{
    return NotFound();
}

return RedirectToAction("Index", "InvoiceItem",
    new { invoice_id=invoice.Id});
//return View(invoice);
}

```

Fragment: /Store/Controllers/InvoiceItemController.cs

```

// GET: InvoiceItem
public async Task<IActionResult> Index(int? invoice_id)
{
    if (invoice_id != null)
        return View(await _context.IvoiceItems
            .Where(i => i.InvoiceId == invoice_id)
            .Include(i => i.Invoice)
            .ToListAsync());
    var applicationDbContext = _context.IvoiceItems
        .Include(i => i.Invoice);
    return View(await applicationDbContext.ToListAsync());
}

```

- teraz mamy możliwość wyboru użytkownika -> fakturę tego użytkownika -> wyświetlić listę pozycji faktury.

Date	IssuedFor	IssuedBy
14.05.2022 15:48:59	001	003

Index - Store

https://localhost:5001/InvoiceItem?invoice_id=1

Store Home Privacy Register Login

Index

Create New

Name	Price	Amount	VAT	Invoice	
Product 1	23,4	2	23	1	Edit Details Delete
Product 2	3,4	1,3	8	1	Edit Details Delete

© 2022 - Store - [Privacy](#)

5. Widoki

Tak jak było wspomniane wcześniej domyślnie każdej akcji kontrolera przypisany jest jeden widok. Widoki są trzymane w katalogu **Views/<controllername>/<action>.cshtml**. Widok to szablon html z osadzonym kodem c#.

```

File Edit Selection View Go Run Terminal Help
EXPLORER ... HomeController.cs InvoiceController.cs InvoiceltemController.cs Startup.cs { app ... }
> OPEN EDITORS Controllers > InvoiceController.cs > {} Store.Controllers > Store.Controllers.InvoiceController > Details(int? id)
> STORE .vscode 54 }
> Areas 55
> bin 56 // GET: Invoice/Create
> Controllers 57 0 references
> Data 58 public IActionResult Create()
> Models 59 {
> obj 60 ViewData["IssuedByGuid"] = new SelectList(_context.Set<Employee>());
> Properties 61 ViewData["IssuedForGuid"] = new SelectList(_context.Customer, "Id"
62 return View();
> Views 63
> Customer 64 // POST: Invoice/Create
> Home 65 // To protect from overposting attacks, enable the specific properties
66 // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598
67 [HttpPost]
68 [ValidateAntiForgeryToken]
69 0 references
70 public async Task<IActionResult> Create([Bind("Id,Date,IssuedForGuid,I
71 {
72 if (ModelState.IsValid)
73 {
74 _context.Add(invoice);
75 await _context.SaveChangesAsync();
76 return RedirectToAction(nameof(Index));
77 }
78 ViewData["IssuedByGuid"] = new SelectList(_context.Set<Employee>())
79 ViewData["IssuedForGuid"] = new SelectList(_context.Customer, "Id"
80 return View(invoice);
81 }
82 // GET: Invoice/Edit/5
83 0 references
84 public async Task<IActionResult> Edit(int? id)
85 {
86 if (id == null)
87 }
```

Ln 51, Col 47 (11 selected) Spaces: 4 UTF-8 CRLF C Sharp ⌂ ⌂

1. Wywołanie widoku

Html generowany jest z szablonu jeśli w akcji zwrócimy wywołanie metody **View**.

```
public class InvoiceController : Controller
{
    // GET: Invoice/Create
    public IActionResult Create()
    {
        return View();
        return View("Delete");
        return View(new Invoice() {});
        return View("Delete", new Invoice() {});
    }
}
```

- **View()** - zwraca domyślny widok np. [/Store/Views/Invoice/Create.cshtml](#)
- **View("Delete")** - wskazujemy nazwę widoku
np. [/Store/Views/Invoice/Delete.cshtml](#)
- **View(new Invoice() {})** - domyślny widok ale przekazujemy tam model danych np. Invoice. Typ modelu danych musi być zbieżny z modelem zdefiniowanym dla widoku (o tym później)

2. Przekazywanie danych do widoków

Widok najczęściej generuje kod Html dynamicznie na podstawie danych przekazanych przeważnie z bazy danych. Dane możemy przekazywać na trzy sposoby.

```
public class InvoiceController : Controller
{
    // GET: Invoice/Create
    public IActionResult Create()
    {
        ViewBag.TestData = "Test";
        ViewData["Test"] = "Test";
        return View(new Invoice() {});
    }
}
```

- **View(object model)** - w tym wypadku widok musi być zorientowany (stworzony) podany model.
- **ViewBag** - dynamiczny obiekt niezwiązany z widokiem. Pola tworzone są w chwili przypisania i będą dostępne w kodzie widoku.
- **ViewData** - wersja ViewBag w wersji tablicy asocjacyjnej

3. Kod widoku

Najłatwiej będzie przedstawić przykłady, a opis umieścić dalej.

Kod /Store/Views/Invoice/Create.cshtml

```
@model Store.Models.Invoice

@{
    ViewData["Title"] = "Create";
}

<h1>Create</h1>

<h4>Invoice</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Create">
            <div asp-validation-summary="ModelOnly" class="text-danger">
            </div>
            <div class="form-group">
                <label asp-for="Date" class="control-label"></label>
                <input asp-for="Date" class="form-control" />
                <span asp-validation-for="Date" class="text-danger">
                </span>
            </div>
            <div class="form-group">
                <label asp-for="IssuedForGuid" class="control-label"></label>
                <select asp-for="IssuedForGuid" class ="form-control"
                        asp-items="ViewBag.IssuedForGuid"></select>
            </div>
            <div class="form-group">
                <label asp-for="IssuedByGuid" class="control-label"></label>
                <select asp-for="IssuedByGuid" class ="form-control"
                        asp-items="ViewBag.IssuedByGuid"></select>
            </div>
            <div class="form-group">
                <input type="submit" value="Create"
                      class="btn btn-primary" />
            </div>
        </form>
    </div>
</div>
```

```

        </div>
    </form>
</div>
</div>

<div>
    <a asp-action="Index">Back to List</a>
</div>

```

- <div **asp-validation-summary**="ModelOnly" /> - wyświetli błędy walidacji

Zagnieżdżanie kodu

- @<kod C#> - kod którego zwracana wartość zostanie wkomponowana w kod html
- @{{<kod C#>}} - blok kodu nie musi nic zwracać
- @foreach (var item in Model) { } - iteracja, pętla.
- @if (item != null) { } - warunek.
- @model - określa model dla widoku, głównie na potrzeby uzupełniania składni. dodatkowo funkcje i atrybuty z **for** (asp-for="...", DisplayFor(...)) będą odnosić się do właśnie tego modelu.
- @using - dołączanie namespace
- @{{Layout = "_Layout";}} - zmienna określająca layout czyli globalny szablon całej strony, w których nasze szablony widoków są osadzane. Domyślny layout znajduje się w **/Views/Shared/_Layout.cshtml**, a globalnie jest ustawiony w pliku **/Views/_ViewStart.cshtml**. Layout możemy dowolnie zmienić w widoku i w kontrolerze.

Helpery z for, generują kod HTML powiązany z modelem danych. Mogą być w formie tagów HTML lub tradycyjnie wywoływanie z metod

- <label **asp-for**="Date" /> - opis dla właściwości Date z modelu.
- <input **asp-for**="Date" /> - pole edycji dla właściwości Date z modelu.
- - pole wyświetla błędy walidacji dla wskazanego pola modelu.
- **@Html.DisplayFor**(modelItem => item.Date) - wyświetla dane z modelu w tym przypadku wskazuje na właściwość Date.
- **@Html.DisplayNameFor**(model => model.Date) - wyświetla nazwę właściwości pola z modelu danych.

Kod /Store/Views/Invoice/Index.cshtml

```

@model IEnumerable<Store.Models.Invoice>

@{
    ViewData["Title"] = "Index";
}

<h1>Index</h1>

<p>

```

```

<a asp-action="Create">Create New</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Date)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.IssuedFor)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.IssuedBy)
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Date)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.IssuedFor.Id)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.IssuedBy.Id)
        </td>
        <td>
            <a asp-action="Edit" asp-route-id="@item.Id">Edit</a> |
            <a asp-action="Details" asp-route-id="@item.Id">
                Details</a> |
            <a asp-action="Delete" asp-route-id="@item.Id">
                Delete</a>
            </td>
        </tr>
}
    </tbody>
</table>

```

Helpy URL, generują adresy URL i mogą być w formie tagów z atrybutami lub w formie metod.

- <form asp-action="Create"> - tworzy adres url na jaki ma zostać wysłany formularz, w przykładzie to akcja Create w bieżącym kontrolerze.
 - **asp-route-id="@item.Id"** - definiowanie dowolnych parametrów w trakcie tworzenia urla. np. **asp-route-arg1**, **asp-route-userId**, atrybuty te są spręgniête z tagami w których występuje adres URL np. <a>, <form>
 - **asp-controller="Invoice"** - wskazuje kontroler, do którego ma być wygenerowany adres URL.
 - **@Url.Action("Index", "Invoice", new {id=21, arg1="abc"})** - generuje sam URL np. "/Invoice/Index/21?arg1=abc"
 - **@Html.ActionLink("link", "Index", "Invoice", new {id=21, arg1="abc"})** - generuje tag np. link
 - <a asp-action="Index" asp-controller="Invoice" asp-route-id="21" asp-route-arg1="abc">"link" - alternatywa do poprzedniego przykładu

4. Kod Widoku główne - Layout

Layout to globalnym szablonem strony w którym osadzony jest kod widoków generowanych w kontrolerach. Domyślny layout znajduje się w `/Views/Shared/_Layout.cshtml`, a globalnie jest ustawiony w pliku `/Views/_ViewStart.cshtml`. Layout możemy dowolnie zmienić w widoku i w kontrolerze. Możemy np mieć inny layout dla klienta inny dla pracownika.

Kod /Views/Shared/_Layout.cshtml

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width,
        initial-scale=1.0" />
    <title>@ ViewData["Title"] - Store</title>
    <link rel="stylesheet"
        href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" />
</head>
<body>
    <header>
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm
            navbar-light bg-white border-bottom box-shadow mb-3">
            <div class="container">
                <a class="navbar-brand" asp-area=""
                    asp-controller="Home" asp-action="Index">Store</a>
                <button class="navbar-toggler" type="button"
                    data-toggle="collapse"
                    data-target=".navbar-collapse">
```

```
        aria-controls="navbarSupportedContent"
        aria-expanded="false"
        aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
</button>
<div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
    <ul class="navbar-nav flex-grow-1">
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area=""
               asp-controller="Home"
               asp-action="Index">
                Home</a>
            </li>
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area=""
               asp-controller="Customer"
               asp-action="Index">
                Customers</a>
            </li>
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area=""
               asp-controller="Home"
               asp-action="Privacy">
                Privacy</a>
            </li>
        </ul>
        <partial name="_LoginPartial" />
    </div>
</div>
</nav>
</header>
<div class="container">
    <main role="main" class="pb-3">
        @RenderBody()
    </main>
</div>

<footer class="border-top footer text-muted">
    <div class="container">
        &copy; 2022 - Store - <a asp-area="" asp-controller="Home"
               asp-action="Privacy">Privacy</a>
    </div>

```

```

</footer>
<script src="~/lib/jquery/dist/jquery.min.js"></script>
<script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js">
</script>
<script src="~/js/site.js" asp-append-version="true"></script>
@await RenderSectionAsync("Scripts", required: false)
</body>
</html>

```

- **@RenderBody()** - punkt w którym zostanie umieszczony nasz szablon widoku zwracany z kontrolera.
- Przy okazji dodaliśmy link do kontrolera Customers

5. Dostosowywanie wygenerowanego kodu

A więc teraz zmodyfikujemy wygenerowane widoki aby widoki były bardziej przejrzyste. Zaczniemy od widoku listy klientów.

Kod Views/Customer/Index.cshtml

```

@model IEnumerable<Store.Models.Customer>

@{
    ViewData["Title"] = "Index";
}

<h1>Index</h1>

<p>
    <a asp-action="Create">Create New</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.UserName)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Address)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Email)
            </th>
    
```

```

<th>
    @Html.DisplayNameFor(model => model.PhoneNumber)
</th>
<th></th>
</tr>
</thead>
<tbody>
@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.UserName)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Address)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Email)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.PhoneNumber)
        </td>
        <td>
            <a asp-action="Index" asp-controller="Invoice"
                asp-route-CustomerId="@item.Id">Invoices</a> |
            <a asp-action="Edit" asp-route-id="@item.Id">Edit</a> |
            <a asp-action="Details"
                asp-route-id="@item.Id">Details</a> |
            <a asp-action="Delete"
                asp-route-id="@item.Id">Delete</a>
            </td>
        </tr>
}
</tbody>
</table>

```

- usunęliśmy część pól i dodaliśmy bezpośredni odnośnik do faktur

przed zmianami:

The screenshot shows a browser window with three tabs: 'Index - Store', 'Home Page - Store', and 'Home Page - Store'. The active tab is 'Index - Store' at the URL <https://localhost:5001/Customer>. The page title is 'Index'. Navigation links include 'Store', 'Home', 'Privacy', 'Register', and 'Login'. The main content is a table with columns: FullName, Address, UserName, NormalizedUserName, Email, NormalizedEmail, EmailConfirmed, and PasswordHash. Two rows of data are shown:

FullName	Address	UserName	NormalizedUserName	Email	NormalizedEmail	EmailConfirmed	PasswordHash
customer1@example.com		CUSTOMER1@EXAMPLE.COM				<input checked="" type="checkbox"/>	AQAAAAEAAACcQ
customer2@example.com		CUSTOMER2@EXAMPLE.COM				<input checked="" type="checkbox"/>	AQAAAAEAAACcQ

po zmianach:

The screenshot shows a browser window with three tabs: 'Index - Store', 'Home Page - Store', and 'Home Page - Store'. The active tab is 'Index - Store' at the URL <https://localhost:5001/Customer>. The page title is 'Index'. Navigation links include 'Store', 'Home', 'Privacy', 'Register', and 'Login'. The main content is a table with columns: UserName, Address, Email, and PhoneNumber. Two rows of data are shown, each with a link to 'Invoices' and other actions:

UserName	Address	Email	PhoneNumber	
customer1@example.com				Invoices Edit Details Delete
customer2@example.com				Invoices Edit Details Delete

© 2022 - Store - [Privacy](#)

Teraz zmienimy kontroler i widoki aby można było dodawać faktury z widoku faktur wskazanego klienta

Kod Views/Invoice/Index.cshtml

```
@model IEnumerable<Store.Models.InvoiceItem>

@{
    ViewData["Title"] = "Index";
}

<h1>Invoices @ViewData["customer_name"]</h1>
```

```
<p>
    <a asp-action="Create"
        asp-route-customer_id='@ViewData["customer_id"]'>Create New</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Name)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Price)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Amount)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.VAT)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Invoice)
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Name)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Price)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Amount)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.VAT)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Invoice.Id)
        </td>
    </tr>
}
```

```

        </td>
        <td>
            <a asp-action="Edit" asp-route-id="@item.Id">Edit</a> |
            <a asp-action="Details"
                asp-route-id="@item.Id">Details</a> |
            <a asp-action="Delete"
                asp-route-id="@item.Id">Delete</a>
        </td>
    </tr>
}
</tbody>
</table>

```

- Dodaliśmy link przejścia do pozycji faktury
- zmieniliśmy sposób wyświetlania właściwości Customer i Employee
- zmieniliśmy również link **Create New** tak aby można było dodawać faktury bezpośrednio do klienta. Dane klienta dodamy do **ViewData** w kontrolerze.

Fragment Controllers/InvoiceController.cs

```

// GET: Invoice
public async Task<IActionResult> Index(string CustomerId)
{
    if (CustomerId != null) {
        var customer = await _context.Users
            .FirstOrDefaultAsync(u=>u.Id == CustomerId);
        if(customer == null)
            return NotFound();
        ViewData["customer_id"] = CustomerId;
        ViewData["customer_name"] = customer.UserName;
        return View(await _context.Invoices
            .Where(i => i.IssuedForGuid == CustomerId)
            .Include(i => i.IssuedBy).Include(i => i.IssuedFor)
            .ToListAsync());
    }
    var applicationDbContext = _context.Invoices
        .Include(i => i.IssuedBy)
        .Include(i => i.IssuedFor);
    return View(await applicationDbContext.ToListAsync());
}

```

- dodano zapytanie do bazy o dane klienta i umieszczamy je w ViewData, którego użyliśmy w widoku.
- jeśli klient nie zostanie znaleziony zwracamy błąd not found 404

Kod Views/Invoice/Create.cshtml

```
@model Store.Models.Invoice
```

```

@{
    ViewData["Title"] = "Create";
}

<h1>Create</h1>

<h4>Invoice</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Create">
            <div asp-validation-summary="ModelOnly"
                class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Date" class="control-label"></label>
                <input asp-for="Date" class="form-control" />
                <span asp-validation-for="Date"
                    class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="IssuedFor"
                    class="control-label"></label>
                <input readonly class ="form-control"
                    value="@Model.IssuedFor.UserName" />
                @Html.HiddenFor(i=>i.IssuedForGuid)
            </div>
            <div class="form-group">
                <label asp-for="IssuedBy" class="control-label"></label>
                <input readonly class ="form-control"
                    value="@Model.IssuedBy.UserName" />
                @Html.HiddenFor(i=>i.IssuedByGuid)
            </div>
            <div class="form-group">
                <input type="submit" value="Create"
                    class="btn btn-primary" />
            </div>
        </form>
    </div>
</div>

<div>
    <a asp-action="Index">Back to List</a>

```

```
</div>
```

- W formularzu teraz wyświetlają się nazwy użytkowników, a ich id jest ukryte.

Fragment Controllers/InvoiceController.cs

```
// GET: Invoice/Create
public IActionResult Create(string customer_id)
{
    var empl = _context.Set<Employee>()
        .FirstOrDefault();
    var cust = _context.Set<Customer>()
        .FirstOrDefault(u=>u.Id == customer_id);
    if(cust == null || empl == null)
        return NotFound();
    return View(new Invoice(){Date=DateTime.Now,
        IssuedBy=empl, IssuedFor=cust,
        IssuedByGuid=empl.Id, IssuedForGuid=cust.Id});
}

// POST: Invoice/Create
// To protect from overposting attacks, enable the specific properties
// you want to bind to.
// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult>
Create([Bind("Id,Date,IssuedForGuid,IssuedByGuid")] Invoice invoice)
{
    var empl = _context.Set<Employee>()
        .FirstOrDefault();
    var cust = _context.Set<Customer>()
        .FirstOrDefault(u=>u.Id == invoice.IssuedForGuid);
    if(cust == null || empl == null)
        return NotFound();
    invoice.IssuedByGuid = empl.Id;
    invoice.IssuedForGuid = cust.Id;

    if (ModelState.IsValid)
    {
        _context.Add(invoice);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(invoice);
```

}

- Do metody **Create** obsługującej żądania **GET** przekazujemy argument **customer_id**, który dodaliśmy do linku **Create New** w widoku. Na podstawie **customer_id** aplikacja rozpoznaje, do którego chcemy dodać nową fakturę. Robi odpowiednie zapytania do bazy danych i tworzy nowy obiekt **Invoice**, który przekazuje do widoku generującego formularz.
- Metoda **Create** obsługująca żądanie **POST** odbiera dane z formularza, waliduje, po czym dodaje do bazy. Dodaliśmy sprawdzenie, czy wskazani użytkownicy w formularzu istnieją.

Po zmianach:

Invoices customer1@example.com

Date	IssuedFor	IssuedBy	
14.05.2022 15:48:59	customer1@example.com	employee@example.com	Items Edit Details Delete
15.05.2022 12:26:06	customer1@example.com	employee@example.com	Items Edit Details Delete

© 2022 - Store - [Privacy](#)

Create

Invoice

Date
15.05.2022, 12:28:30.886

IssuedFor
customer1@example.com

IssuedBy
employee@example.com

Create

[Back to List](#)

Date	IssuedFor	IssuedBy	
14.05.2022 15:48:59	customer1@example.com	employee@example.com	Items Edit Details Delete
18.05.2022 01:21:00	customer2@example.com	employee@example.com	Items Edit Details Delete
15.05.2022 12:26:06	customer1@example.com	employee@example.com	Items Edit Details Delete
15.05.2022 12:28:30	customer1@example.com	employee@example.com	Items Edit Details Delete

© 2022 - Store - [Privacy](#)

6. Modele pomocnicze ViewModels

Tak jak można zobaczyć, czasami model danych z kontekstu bazy danych może przeszkadzać, zwłaszcza gdy pracujemy z narzędziami, które generują kod. W niektórych sytuacjach jak no model Customer zawiera bardzo dużo właściwości, których nie chcemy przekazywać do widoków, a nawet nie chcemy pobierać z bazy danych. Do tego celu możemy tworzyć dodatkowe modele danych w praktyce zwykłych klas, które po prostu nie są powiązane z DBContext.

Dlatego stworzymy nowy model widoku **CustomerVM** i opiszymy go atrybutami, które będą pomocne w trakcie generowania nowego widoków.

Kod /Models/CustomerVM.cs

```
using System.ComponentModel.DataAnnotations;
using Microsoft.AspNetCore.Mvc;

namespace Store.Models
{
    public class CustomerVM
    {
        [Key]
        [HiddenInput(DisplayValue = false)]
        public string Id { get; set; }

        [Required]
        [Display(Name = "Customer Name")]
        public string FullName { get; set; }

        [Required]
        public string Address { get; set; }
    }
}
```

```
}
```

- Model **CustomerVM** posiada tylko trzy właściwości i po żadnej innej klasie nie dziedziczy tak jak ma to miejsce z modelem **Customer**
- **[HiddenInput]** - atrybut sprawia, że w formularzach pole będzie ukryte nieedytowalne
- **[Required]** - formularz będzie wymagał uzupełnienia pola
- **[Display]** - zmiana sposobu wyświetlania w naszym przypadku zmieniliśmy opis właściwości

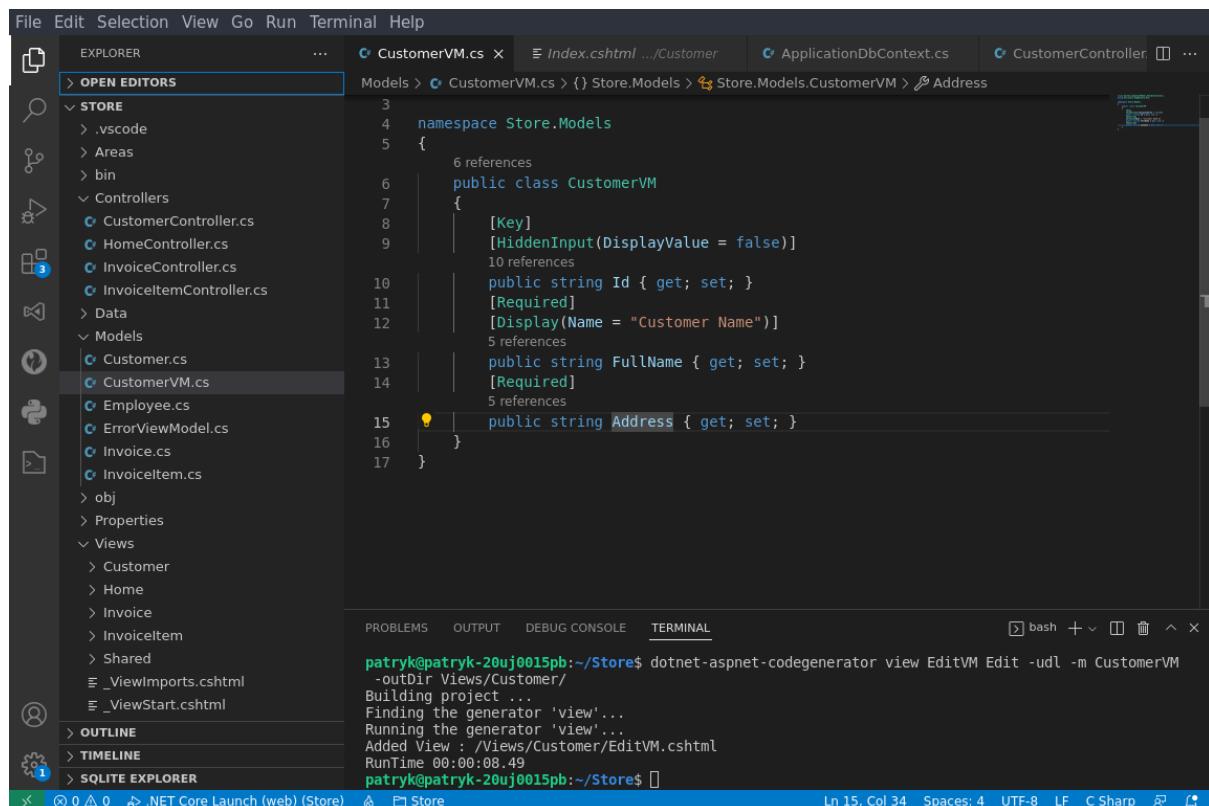
Atrybuty są obszernie opisane w dokumentacji:

- <https://docs.microsoft.com/dotnet/api/system.componentmodel.dataannotations?view=net-5.0>
- <https://docs.microsoft.com/dotnet/api/microsoft.aspnetcore.mvc.hiddeninputattribute?view=aspnetcore-5.0>

Dla dowolnego modelu możemy wygenerować automatycznie widok za pomocą komendy:

```
dotnet-aspnet-codegenerator view EditVM Edit -udl -m CustomerVM -outDir Views/Customer/
```

- **EditVM** - nazwa widoku
- **Edit** - typ widoku ten bedzie z formularzem do edycji
- **-outDir Views/Customer/** - lokalizacja gdzie zostaną pliki umieszczone
- **-m CustomerVM** - wskazanie modelu danych dla którego generujemy widok
- opis reszty parametrów oraz możliwości narzędzia znajdziecie na stronie:
<https://docs.microsoft.com/aspnet/core/fundamentals/tools/dotnet-aspnet-codegenerator?view=aspnetcore-5.0>



The screenshot shows the Visual Studio Code interface. The code editor displays the `CustomerVM.cs` file with its generated code. The terminal at the bottom shows the command being run:

```
patryk@patryk-20uj0015pb:~/Store$ dotnet-aspnet-codegenerator view EditVM Edit -udl -m CustomerVM -outDir Views/Customer/
```

The output of the command shows the generator building the project, finding the view, running the generator, adding the view, and providing runtime information.

Kod Views/Customer/EditVM.cshtml

```
@model Store.Models.CustomerVM

@{
    ViewData["Title"] = "EditVM";
}

<h1>EditVM</h1>

<h4>CustomerVM</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="EditVM">
            <div asp-validation-summary="ModelOnly"
class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Id" class="control-label"></label>
                <input asp-for="Id" class="form-control" />
                <span asp-validation-for="Id"
class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="FullName" class="control-label"></label>
                <input asp-for="FullName" class="form-control" />
                <span asp-validation-for="FullName"
class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Address" class="control-label"></label>
                <input asp-for="Address" class="form-control" />
                <span asp-validation-for="Address"
class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Save" class="btn
btn-primary" />
            </div>
        </form>
    </div>
</div>
```

```
<div>
    <a asp-action="Index">Back to List</a>
</div>
```

- <form asp-action="EditVM"> - trzeba zmienić adres wygenerowany "EditVM" na "Edit"

Pozostaje tylko dodanie zmian w kontrolerze.

Kod Controllers/CustomerController.cs

```
// GET: Customer/Edit/5
public async Task<IActionResult> Edit(string id)
{
    if (id == null)
    {
        return NotFound();
    }

    var customer = await _context.Customer.FindAsync(id);
    if (customer == null)
    {
        return NotFound();
    }
    var model = new CustomerVM(){Id = customer.Id,
        Address = customer.Address,
        FullName = customer.FullName};
    return View("EditVM", model);
}

// POST: Customer/Edit/5
// To protect from overposting attacks, enable the specific properties
// you want to bind to.
// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(string id,
[Bind("FullName,Address,Id")] CustomerVM customer)
{
    if (id != customer.Id)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
```

```

{
    try
    {
        var model = _context.Customer
            .FirstOrDefault(c=>c.Id == id);
        if (model == null)
            return NotFound();
        model.FullName = customer.FullName;
        model.Address = customer.Address;
        _context.Update(model);
        _context.SaveChanges();
    }
    catch (DbUpdateConcurrencyException)
    {
        if (!CustomerExists(customer.Id))
        {
            return NotFound();
        }
        else
        {
            throw;
        }
    }
    return RedirectToAction(nameof(Index));
}
return View("EditVM", customer);
}

```

- Wskazaliśmy nowy widok "EditVM", który wcześniej wygenerowaliśmy.
- Widok przyjmuje model **CustomerVM** natomiast w bazie mamy model **Customer** z tego powodu musimy przepisywać dane do nowego modelu.
- **Trzeba zaznaczyć że kopiowanie pomiędzy modelami można zastąpić automapperem. Natomiast zapytania do bazy danych przenieść z kontrolerów do serwisów.**

Przed

The screenshot shows a web browser window with the URL <https://localhost:5001/Customer/Edit/001>. The page title is "Edit - Store". The navigation bar includes links for "Store", "Home", "Customers", "Privacy", "Register", and "Login". The main content area has a heading "Edit Customer". It contains four input fields: "FullName" (empty), "Address" (empty), "UserName" (customer1@example.com), and "NormalizedUserName" (CUSTOMER1@EXAMPLE.COM).

The screenshot shows a web browser window with the URL <https://localhost:5001/Customer/Edit/001>. The page title is "EditVM - Store". The navigation bar includes links for "Home Page - Store", "CustomerVM", and "EditVM". The main content area has a heading "EditVM CustomerVM". It contains two input fields: "Customer Name" (empty) and "Address" (empty). Below the address field is a blue "Save" button. At the bottom left is a link "Back to List".

© 2022 - Store - [Privacy](#)