

Model ASP.NET MVC Core 3.1 Web App - Deel 1

1 Inleiding

In dit document wordt beschreven hoe je stap voor stap een Web App in MVC .Net Core 3.1 maakt.

Dit is een model- en voorbeeldproject die je kan gebruiken als leidraad voor het eindwerk.

Dit voorbeeldproject illustreert de verschillende onderdelen die we hebben behandeld in de lessen:

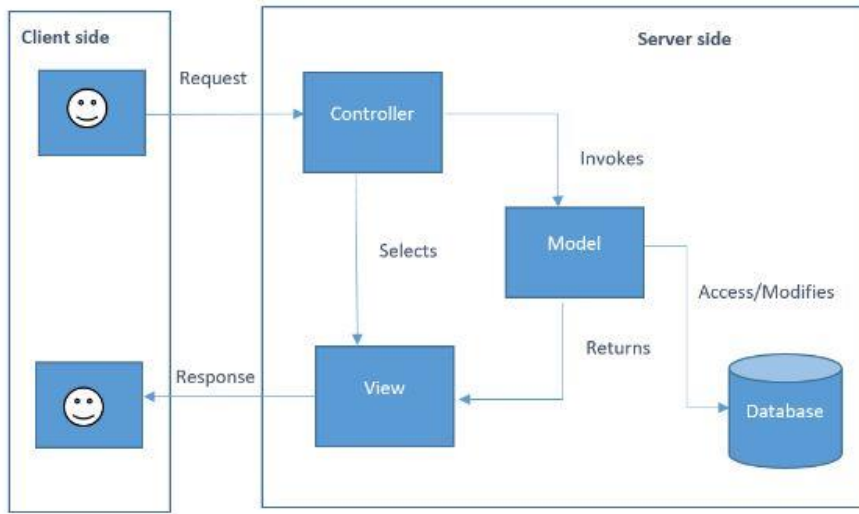
- ASP.Net Core Web applicatie met MVC architectuur design pattern
- Sql Server database aanspreken en gebruik van ORM Entity Framework Core
- Gebruik van LINQ to Entities en lambda expressions
- Authenticatie en Authorisatie met ASP.Net Identity en beheren van Individuele User Accounts in SQL Server.
- Facebook en Google authenticatiemogelijkheden
- Gebruik van HTML/Javascript/Razor syntax voor de UI
- Mogelijkheid om online te bestellen en te betalen via kredietkaart.

Het project is een eenvoudige web applicatie voor een restaurant met ingebouwde content management om de inhoud van de website te beheren. Daarnaast is er authenticatie op verschillende manieren mogelijk: via google, facebook of via in db beheerde email/wachtwoord.

De autorisatie wordt voorzien door toekenning van verschillende rollen (bv manager rol kan alles: content management; personeelsbeheer, het toekennen van rollen aan personeel en bestellingen beheren en opvolgen).

In de volgende stappen maak je de MVC Web applicatie met Authenticatie en een SQL Server database

1.1 Overzicht MVC Architectuur



MVC - is een architecturaal pattern

Structuur bestaat uit 3 componenten (models / controllers / views)

3 componenten die los aan elkaar hangen (loose coupling) (voordelig voor maintenance/testing)

Dit is het **SoC principe (Separation of Concerns)**

Elke component heeft zijn eigen verantwoordelijkheid:

Model : data (lezen/schrijven/wijzigen) bv via een ORM (Object Relational Mapper) zoals EF Core

View : verzorgt de UI (representatie van gegevens) die hij meekrijgt van Controller

(naar User HTTP response request) - web pagina in browser

Controller: Requests afhandelen (HttpGet/HttpPost)

via (action) methodes

daarvoor gaat hij Model vragen om data en ev.

Zelf data aan model leveren om naar de db te schrijven

Geeft gegevens door aan View die moeten worden getoond aan eindgebruiker

1.2 Routing in mvc

Bv <https://www.website.com/Customers/Edit/101>

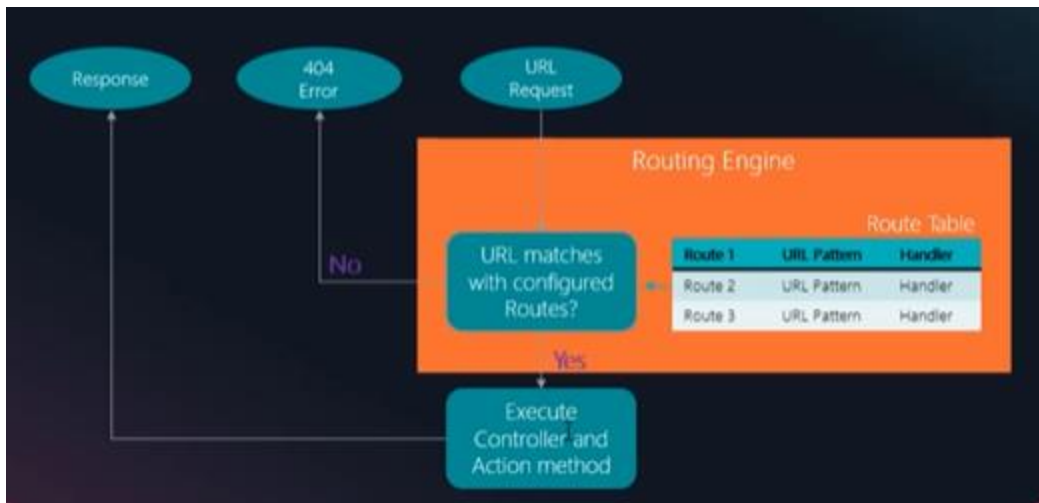
Standaard gedrag: Zal een methode Edit(...) aanroepen in de CustomersController class en een parameter (meestal een id van type int) met waarde 101 meegeven aan deze methode Edit

Routing systeem:

Request wordt doorgestuurd naar de CustomersController

En Methode Edit in CustomersController wordt dan uitgevoerd

Indien de gevraagde request url niet wordt herkend, wordt een Foutcode 404-Page not found teruggegeven als http response naar de browser van de eindgebruiker



2 Aanmaken van Sql Server database

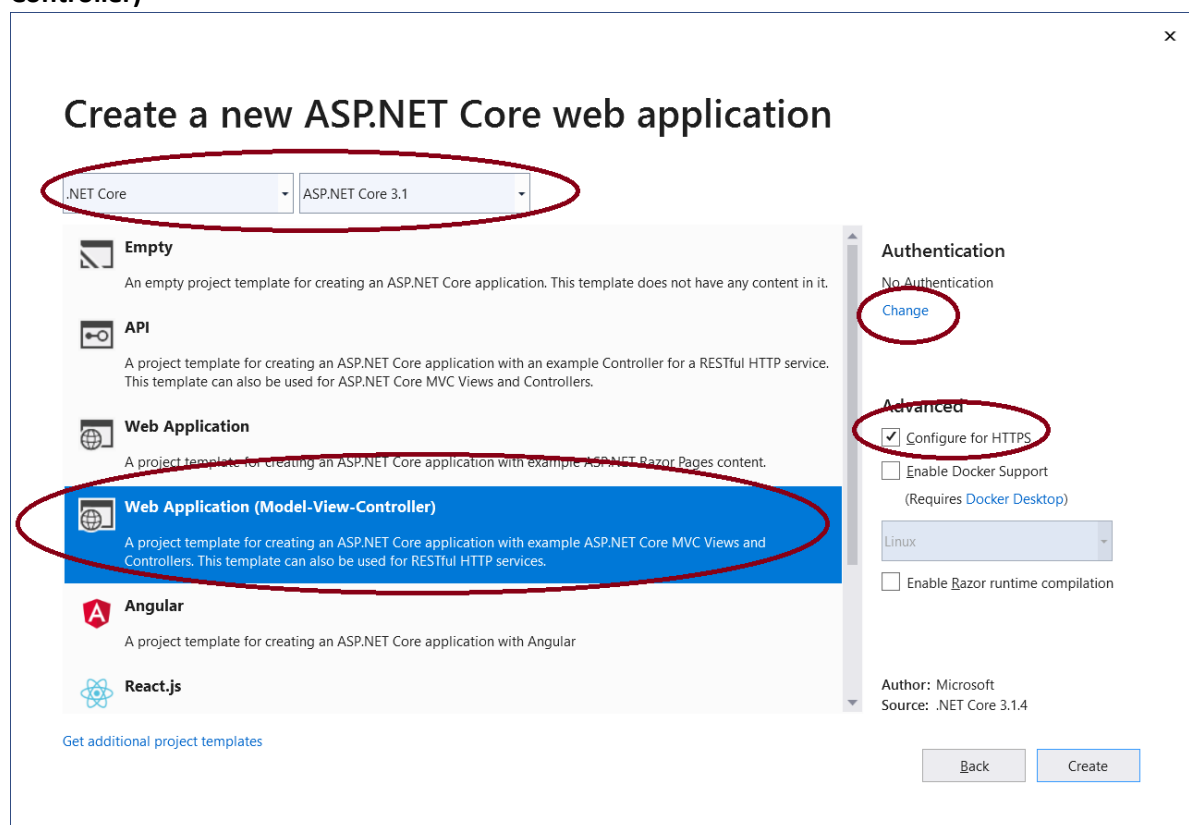
Open Sql Server Management studio en maak lokaal een nieuwe database met naam **PittigRestoMVC**. Deze database gaan we gebruiken om alle gegevens bij de houden van de web app.

3 Web App MVC PittigRestoMVC aanmaken

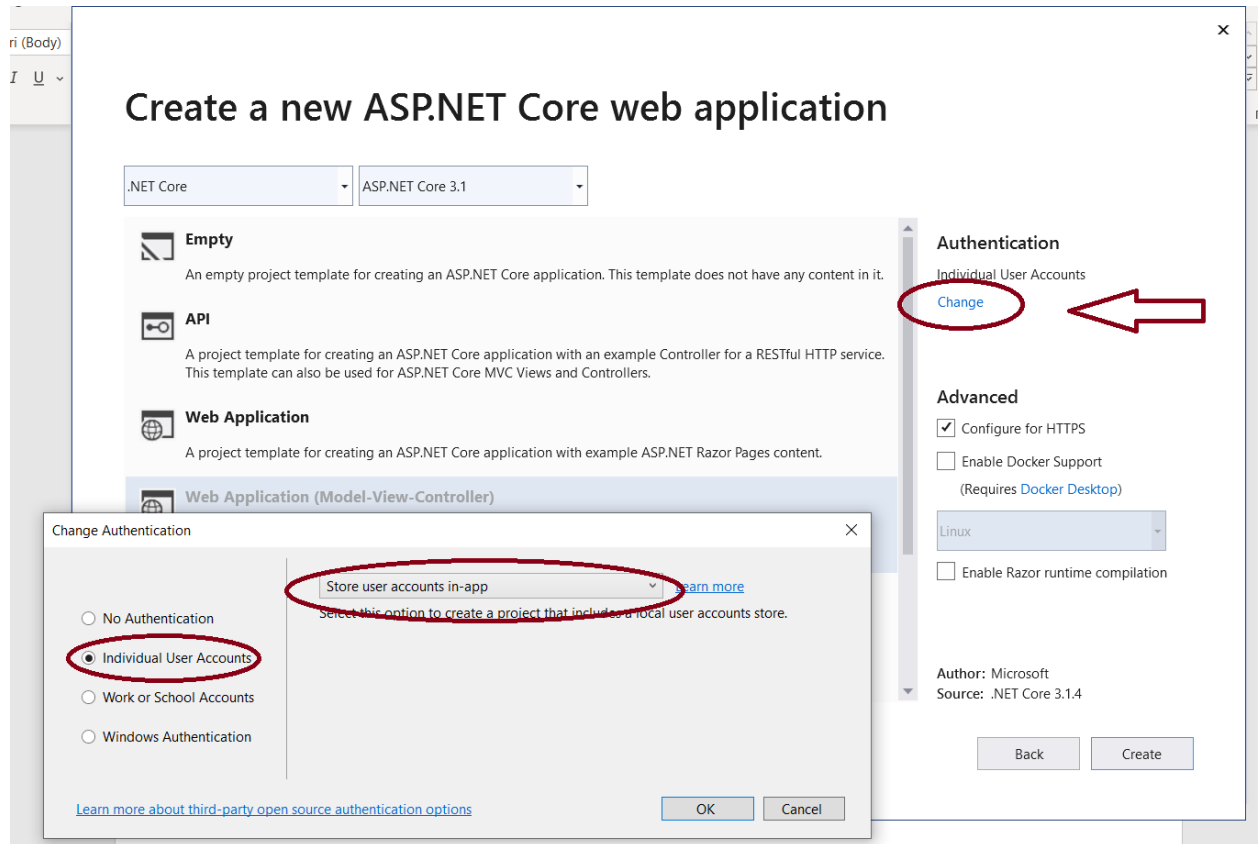
Aanmaken project in VS 2019

Maak in VS2019 een nieuw ASP.NET Core Web Application aan
Geef als naam **PittigRestoMVC**

Controleer dat je **ASP.NET Core 3.x** hebt geselecteerd en **Web Application (Model-View-Controller)**



Klik op de link change.. onder Authenticatie



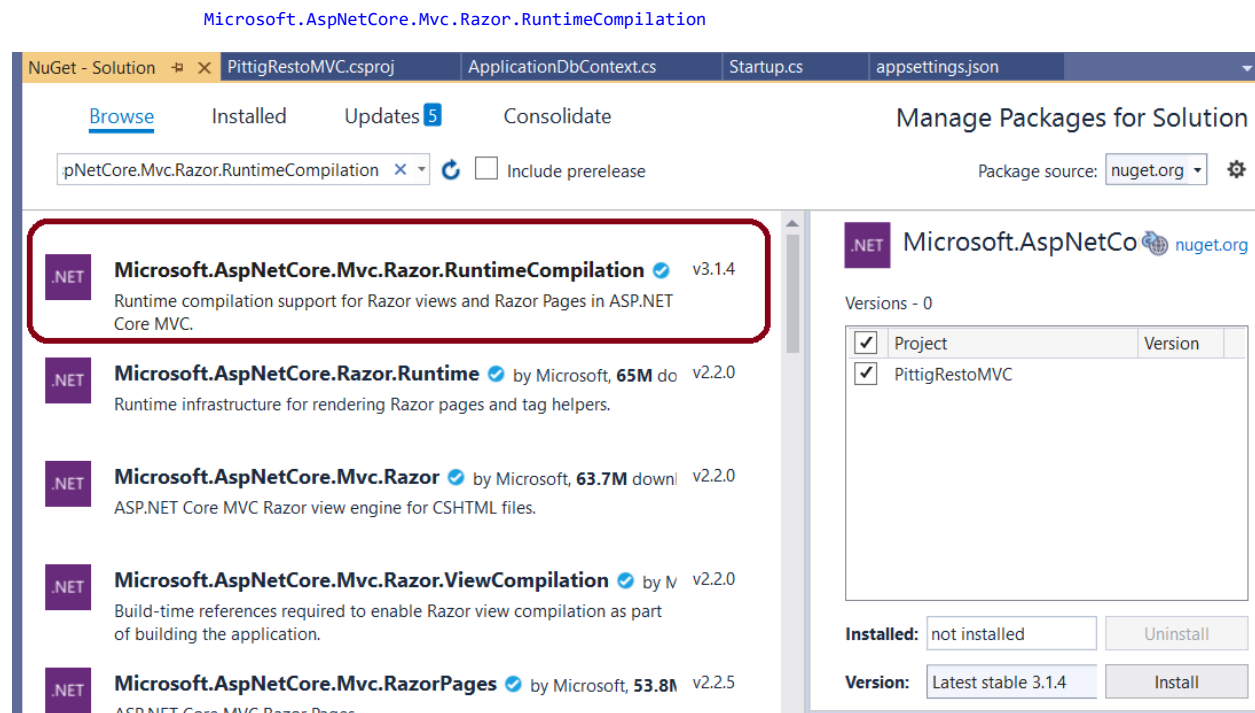
In het popup-venster, kies voor de optie “Individual User Accounts” en dan “Store user accounts in-app”

Klik op **OK button** in het popup-venster en daarna op de **button Create** in het hoofdvenster

Opmerking: Bij de creatie van de MVC Web App met Authenticatie met optie “Individual User Accounts” en dan “Store user accounts in-app” worden de volgende NuGet packages automatisch geïnstalleerd:

```
<PackageReference Include="Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore" Version="3.1.3" />
<PackageReference Include="Microsoft.AspNetCore.Identity.EntityFrameworkCore" Version="3.1.3" />
<PackageReference Include="Microsoft.AspNetCore.Identity.UI" Version="3.1.3" />
<PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="3.1.3" />
<PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="3.1.3" />
```

Installeer bijkomend de volgende NuGet package:

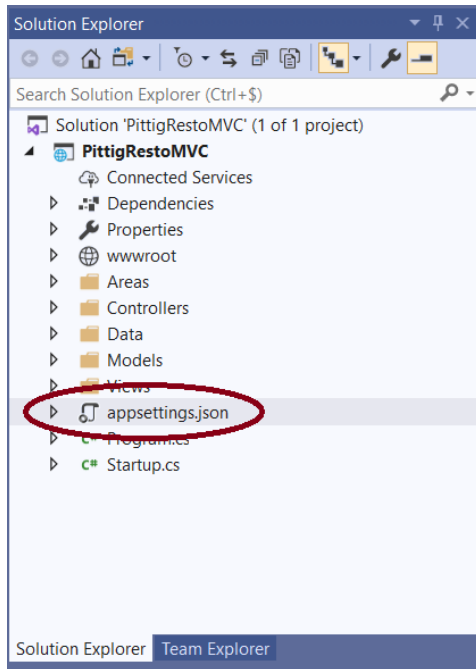


Deze Package zorgt dat je razor pages kan aanpassen at runtime (dus tijdens het runnen van je project zie je onmiddellijk de wijzigingen aan een razor page)

Open startup.cs na de installatie van de NuGet package en voeg de Runtimecompilaton van Razor pages toe als service:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(
            Configuration.GetConnectionString("DefaultConnection")));
    services.AddDefaultIdentity<IdentityUser>(options =>
options.SignIn.RequireConfirmedAccount = true)
        .AddEntityFrameworkStores<ApplicationDbContext>();
    services.AddControllersWithViews();
    services.AddRazorPages().AddRazorRuntimeCompilation();
}
```

Het project is aangemaakt en hierin wordt de ASP.NET Identity reeds voorzien voor Authenticatie en autorisatie. Ook verschillende folders voor de structuur van MVC worden voorzien (Models, controllers en views)



Open het bestand **appsettings.json** en pas de connectionstring aan voor de SQL Database PittigRestoMVC dat je hiervoor hebt aangemaakt

```
"ConnectionStrings": {  
  "DefaultConnection":  
    "Server=(localdb)\\mssqllocaldb;Database=PittigRestoMVC;Trusted_Connection=True;MultipleActiveResultSets=true"  
}
```

Open **startup.cs**. Bekijk de **ConfigureServices** en **Configure** methoden

1. **ConfigureServices** voorziet reeds via de DI container van ASP.NET een connectie met de SQL server db via de class **ApplicationDbContext**. De Connectionstring wordt uit de **appsettings.json** gelezen via **Configuration.GetConnectionString("DefaultConnection")**

```
public void ConfigureServices(IServiceCollection services)  
{  
    services.AddDbContext<ApplicationDbContext>(options =>  
        options.UseSqlServer(  
            Configuration.GetConnectionString("DefaultConnection")));  
    services.AddDefaultIdentity<IdentityUser>(options =>  
        options.SignIn.RequireConfirmedAccount = true);  
    .AddEntityFrameworkStores<ApplicationDbContext>();  
    services.AddControllersWithViews();  
    services.AddRazorPages().AddRazorRuntimeCompilation();  
}
```

Verwijder de parameter van **AddDefaultIdentity** (dit is om het login-proces eenvoudiger te maken tijdens development)

Er is eveneens een mogelijkheid voorzien om gebruikers in te loggen en de registreren via **services.AddDefaultIdentity<IdentityUser>**. Er is nog geen mogelijkheid om IdentityRoles te gebruiken. Deze gaan we later hier nog moeten toevoegen.

Verder zien we **services.AddControllersWithViews()**. Dit zorgt dat we MVC met Models, Controllers en Views kunnen gebruiken in deze App

Als laatste zien we **services.AddRazorPages()**. De ASP.Identity registratie en login van IdentityUsers gebeurt via Razor pages. Dit zijn pagina's die Razor-code bevatten. Deze bevat HTML-achtige code van HelperTags en eveneens c-sharp code bij deze tekens `@{}`. Het is de Razor-engine die deze Razor code zal omzetten naar HTML pagina's en deze via een HTTP response aan de (browser van de) Eindgebruiker bezorgen als antwoord op zijn HTTP request naar de Web App.

2. We bekijken nu ook de methode Configure in startup.cs:

```
// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseDatabaseErrorPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
        app.UseHsts(); /*security: om de HTTP downgrade attack te vermijden waarbij een HTTPS
            request wordt "gedowngraded" tot een niet-beveiligde HTTP request*/
    }
    app.UseHttpsRedirection();
    app.UseStaticFiles();

    app.UseRouting();

    app.UseAuthentication(); //Wie ben je?
    app.UseAuthorization(); //Wat mag je doen?

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

Elke HTTP request doorloopt alle de 'middleware' componenten die hier toegevoegd worden aan het app object:

-UseStaticFiles(): is een middleware component die zorgt dat de HTML, CSS, JS en image files onder de wwwroot folder kunnen worden gebruikt en aangesproken door de app

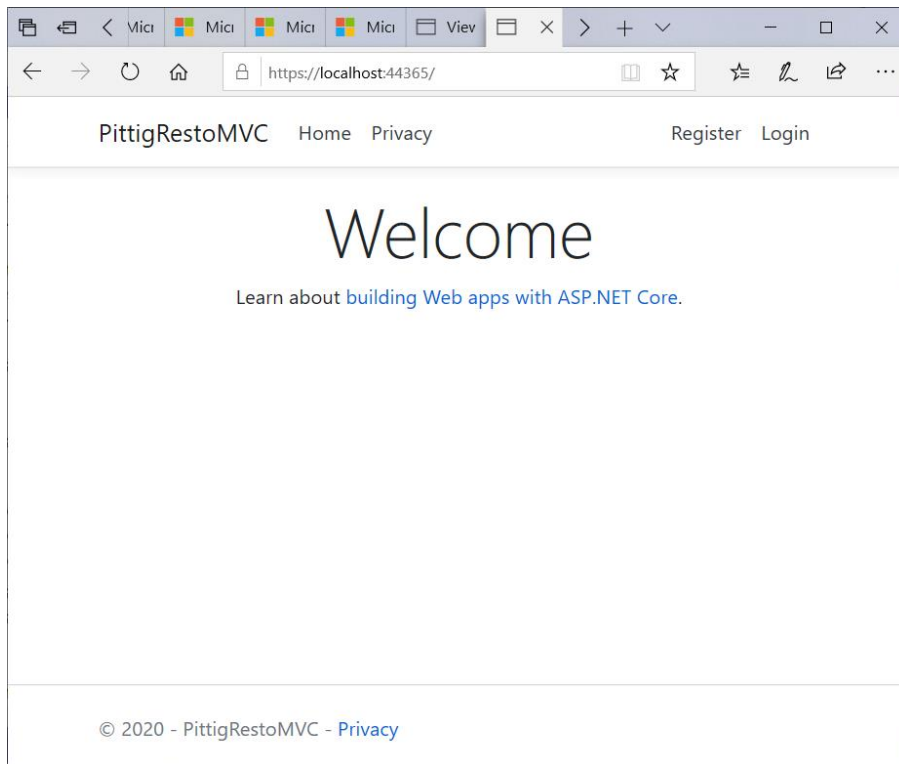
-UseRouting(): Zorgt dat elke request door het MVC routing systeem zal worden gestuurd

-UseAuthentication(): Zorgt dat een login/logout en registratie via HTTP request mogelijk wordt

-UseEndpoints(): hier wordt de default Route gedefinieerd. De url van elke request verwacht zal hieraan moeten voldoen om te kunnen worden verwerkt. Indien de url enkel <https://localhost:xxxx/> bevat wordt

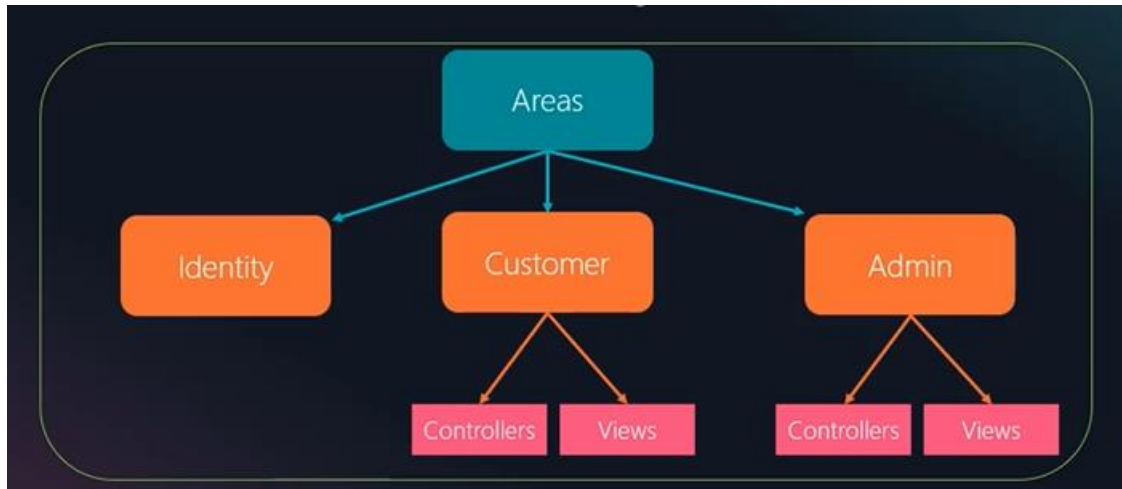
bij default de request gestuurd naar <https://localhost:xxxx/Home/Index>. Het is de action methode Index van de HomeController die deze HTTP Get request zal afhandelen.

Start het project eens op Je krijgt bij default deze (Home/Index) pagina te zien:

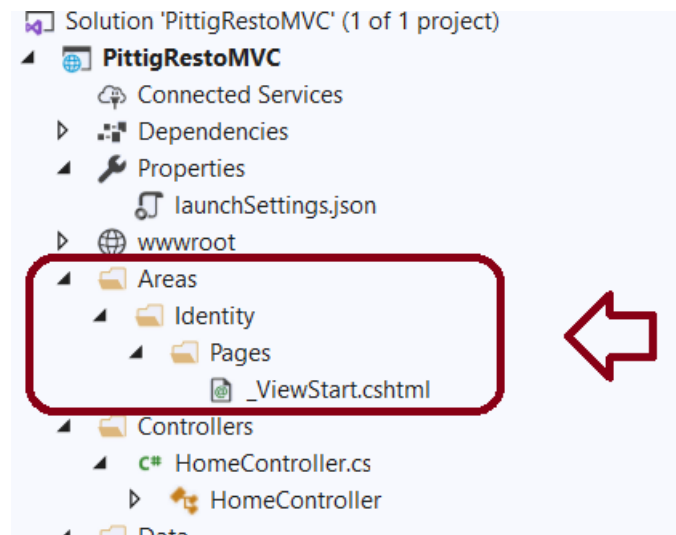


4 Toevoegen van Areas

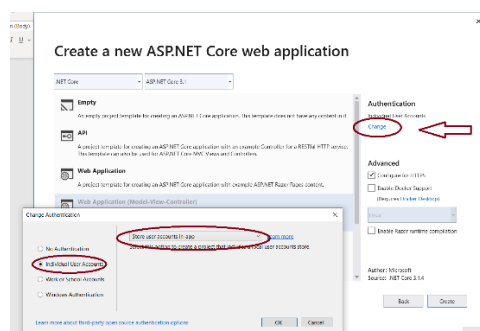
we gaan de volgende structuur met Area folders in het project aanmaken:



Onder de folder Areas is er reeds een Area folder met naam Identity aangemaakt (bij creatie van het project wordt dit voorzien aangezien we de authenticatie optie hebben aangegeven bij creatie)

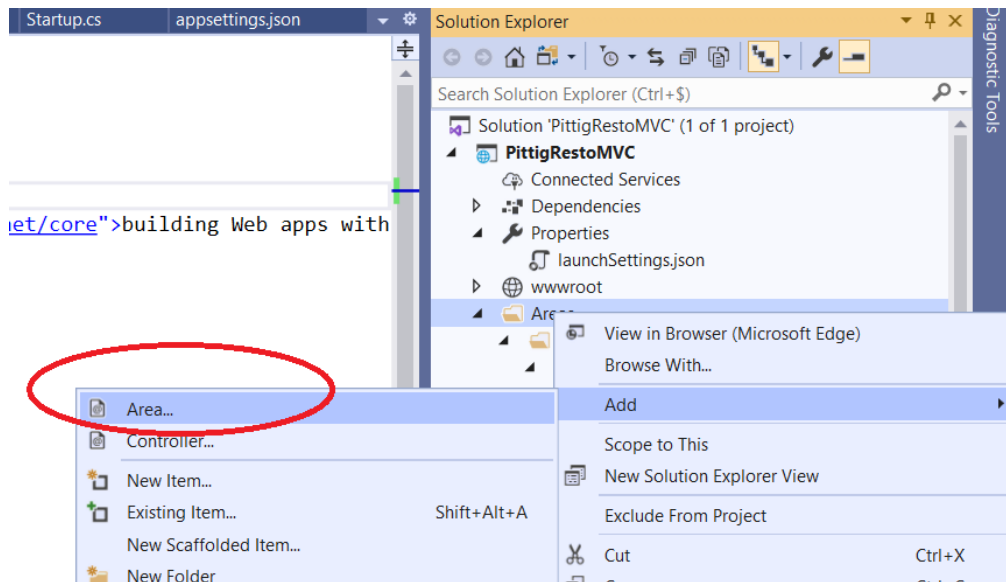


De Area Identity is aangemaakt, omdat we bij creatie van het project hadden we deze opties hadden gekozen:



We gaan nu 2 Areas toevoegen: Area Customer en Area Admin:

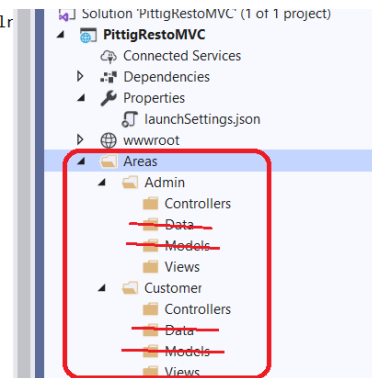
Rechtsklik op de Area folder in de Solution explorer en Kies Add/Area:



Onder de Folder Area zijn nu een reeks nieuwe folders aangemaakt. We verwijderen de Folders met naam Model en Data, want we gaan alle models en Data samenzetten in de root-folders Models en Data (niet apart). De Controllers en Views gaan we wel apart voorzien in de 2 nieuwe Areas

following code to the Configure method in your Application's Startup class if not already

```
app.UseMvc(routes =>
{
    routes.MapRoute(
        name : "areas",
        template : "{area:exists}/{controller=Home}/{action=Index}/{id?}"
    );
});
```



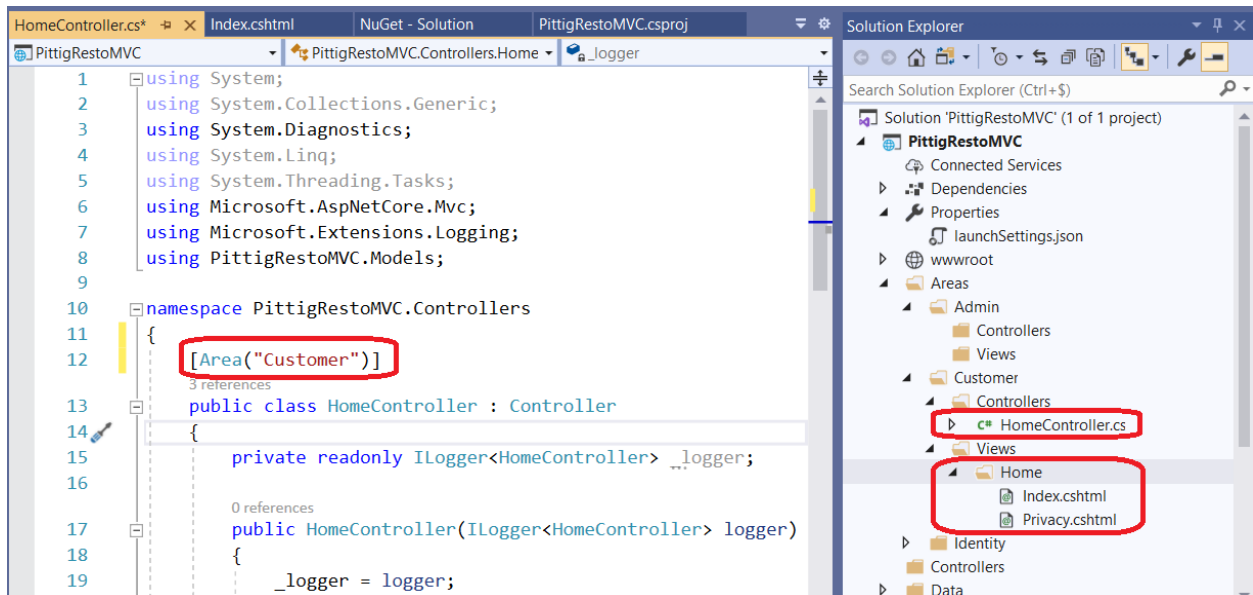
In het linkse venster zie je een instructie-bestand dat aangeeft dat de Routing pattern bij het gebruik van Areas moet worden aangepast (dit is voor Core 2.x). We gaan dit doen, maar de code is iets verschillend aangezien we in Core 3.1 werken:

Open startup.cs en wijzig de volgende lijn in de methode Configure(...)

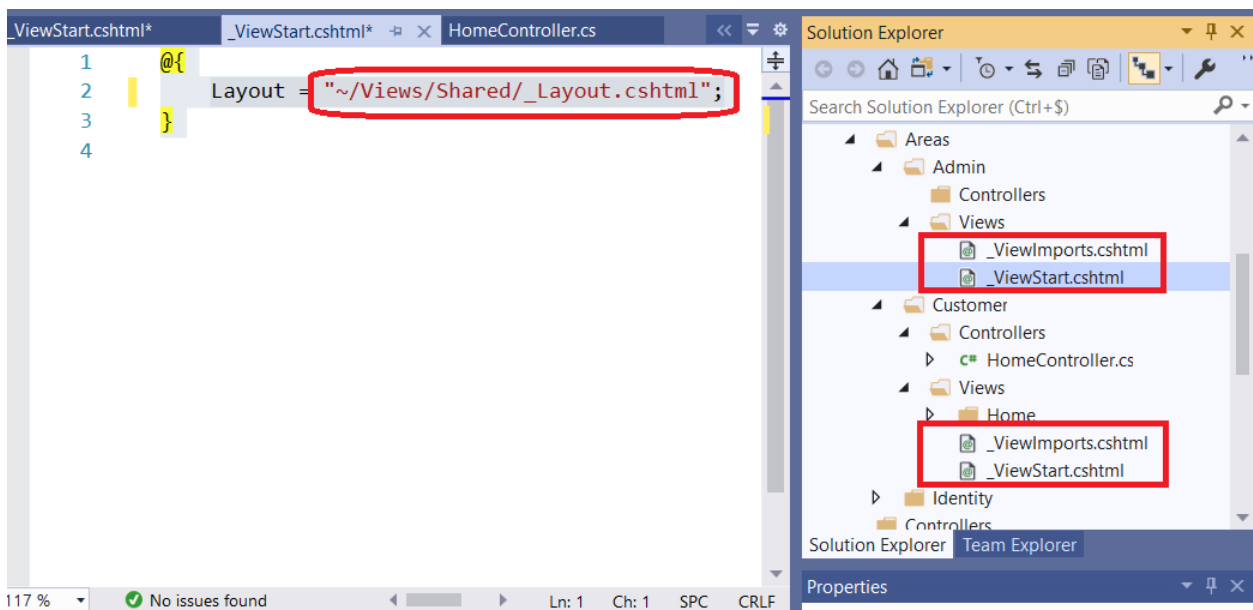
```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{area=Customer}/{controller=Home}/{action=Index}/{id?}");

    endpoints.MapRazorPages();
});
```

- **Verplaats** de Controllers/HomeController.cs file naar de folder Areas/Customer/Controllers:
- **Verplaats** de folder Views/Home (met de 2 bestanden) naar de folder Areas/Customer/Views:
- Open HomeController.cs en voeg het attribuut **[Area("Customer")]** toe boven de class definitie:



Kopieer de bestanden `Views/_ViewImports.cshtml` en `Views/_ViewStart.cshtml` naar de folders `Areas/Admin/Views` en `Areas/Customer/Views`



Pas de inhoud van de bestanden `Areas/Admin/Views/_ViewStart.cshtml` en `Areas/Customer/Views/_ViewStart.cshtml` aan (opm: de oorsponkelijke file `Views/_ViewStart.cshtml` **NIET** aanpassen) naar:

```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}
```

Start de applicatie en test nu de applicatie met deze url (Area customer/Controller Home en Action method Index):

<https://localhost:xxxxx/Customer/Home/index>

5 Toevoegen navigatie in shared Layout

Open de Shared razor layout file van de home-pagina: **Views/Shared/_Layout.cshtml**

Wijzig deze code

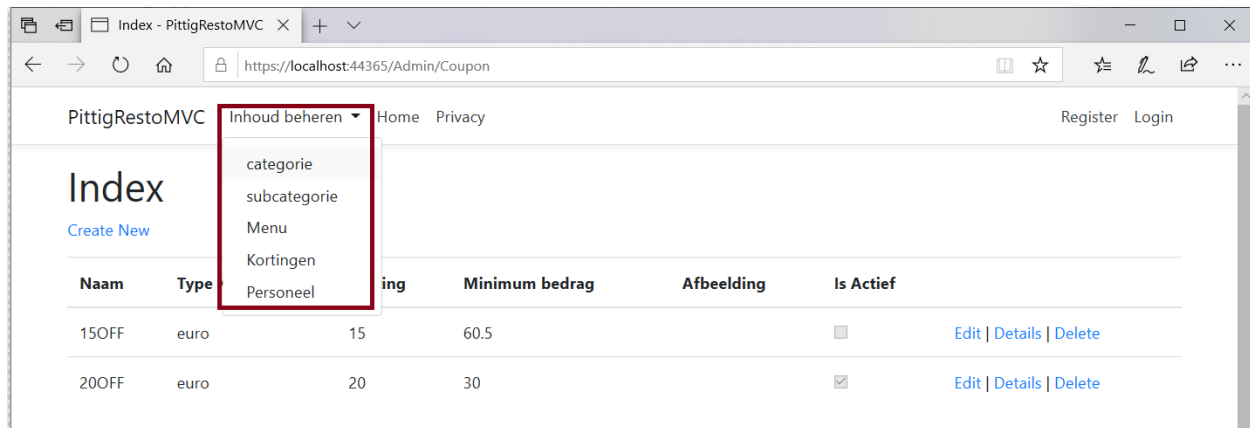
```
<partial name="_LoginPartial" />
<ul class="navbar-nav flex-grow-1">
  <li class="nav-item">
    <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
  </li>
  <li class="nav-item">
    <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
  </li>
</ul>
```

Door

```
<partial name="_LoginPartial" />
<ul class="navbar-nav flex-grow-1">
  <li class="nav-item dropdown text-white-50">
    <a class="nav-link dropdown-toggle" href="#" id="navbarDropDownMenuLink" role="button"
      data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
      Inhoud beheren
    </a>
    <div class="dropdown-menu" aria-labelledby="navbarDropDownMenuLink">
      <a class="dropdown-item" asp-action="Index" asp-controller="Category" asp-area="Admin">categorie</a>
      <a class="dropdown-item" asp-action="Index" asp-controller="SubCategory" asp-area="Admin">subcategorie</a>
      <a class="dropdown-item" asp-action="Index" asp-controller="MenuItem" asp-area="Admin">Menu</a>
      <a class="dropdown-item" asp-action="Index" asp-controller="Coupon" asp-area="Admin">Kortingen</a>
      <a class="dropdown-item" asp-action="Index" asp-controller="User" asp-area="Admin">Personeel</a>
    </div>
  </li>
  <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="Home" asp-action="Privacy">Privacy</a>
  </li>
</ul>
```

Deze code zal een dropdown list met navigatie links in de header sectie tonen van elke pagina die deze shared layout gebruikt

Start de app op en test de dropdown list met navigatie links



Aanpassen van layout: Navigatie-bar met zwarte achtergrond met bleke hyperlinks:

Open Views/Shared/_Layout.cshtml

voor een zwarte achtergrond voor navigatie-bar, pass de class aan (bootstrap css style) in deze code:

```
<header>
  <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white
border-bottom box-shadow mb-3">
    <div class="container">
```

naar:

```
<header>
  <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-dark bg-dark
border-bottom box-shadow mb-3">
    <div class="container">
```

Voor bleek-gekleurde hyperlinks, pas deze code aan:

```
<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="Home" asp-
action="Privacy">Privacy</a>
</li>
```

De andere hyperlinks van de navigatie-bar staan in een Partial Razor view gedefinieerd.

Open deze Partial Razor View Views/Shared/_LoginPartial.cshtml en wijzig overal de code

```
class="nav-link text-dark"
```

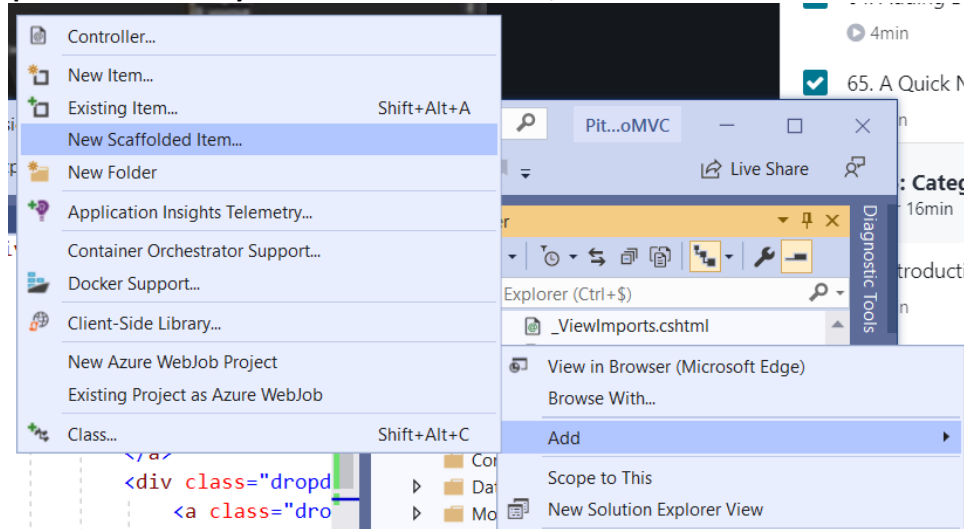
naar

```
class="nav-link"
```

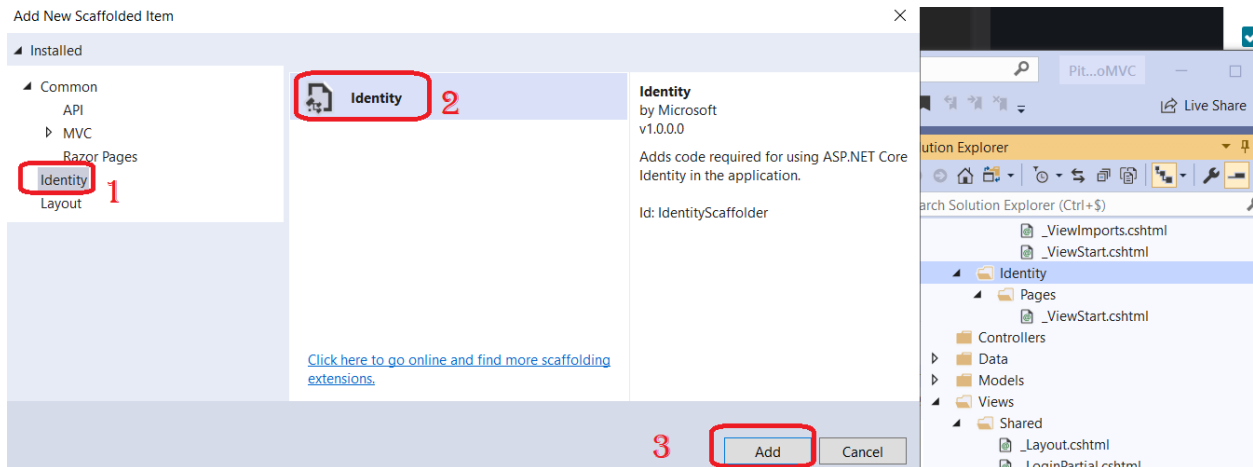
6 Scaffolding Identity Razor pages

We gaan Razor pages toevoegen om In te loggen, Registration,... functionaliteit te voorzien in ons project.

Rechtsklik op de folder Identity onder Areas en kies Add/New Scaffolded item...



In het popup-venster, kies in de lijst aan de linkerkant “Identity” en in het midden Identity
Klik op



Klik op de button “Add”

In het volgende popup-venster, zet de volgende opties en gegevens:

Add Identity ×

Select an existing layout page, or specify a new one:

...

(Leave empty if it is set in a Razor _viewstart file)

☒ Override all files

Choose files to override

<input checked="" type="checkbox"/> Account\StatusMessage	<input checked="" type="checkbox"/> Account\AccessDenied	<input checked="" type="checkbox"/> Account\ConfirmEmail
<input checked="" type="checkbox"/> Account\ConfirmEmailChange	<input checked="" type="checkbox"/> Account\ExternalLogin	<input checked="" type="checkbox"/> Account\ForgotPassword
<input checked="" type="checkbox"/> Account\ForgotPasswordConfirmation	<input checked="" type="checkbox"/> Account\Lockout	<input checked="" type="checkbox"/> Account>Login
<input checked="" type="checkbox"/> Account>LoginWith2fa	<input checked="" type="checkbox"/> Account>LoginWithRecoveryCode	<input checked="" type="checkbox"/> Account\Logout
<input checked="" type="checkbox"/> Account\Manage\Layout	<input checked="" type="checkbox"/> Account\Manage\ManageNav	<input checked="" type="checkbox"/> Account\Manage\StatusMessage
<input checked="" type="checkbox"/> Account\Manage\ChangePassword	<input checked="" type="checkbox"/> Account\Manage\DeletePersonalData	<input checked="" type="checkbox"/> Account\Manage\Disable2fa
<input checked="" type="checkbox"/> Account\Manage\DownloadPersonalData	<input checked="" type="checkbox"/> Account\Manage\Email	<input checked="" type="checkbox"/> Account\Manage\EnableAuthenticator
<input checked="" type="checkbox"/> Account\Manage\ExternalLogins	<input checked="" type="checkbox"/> Account\Manage\GenerateRecoveryCodes	<input checked="" type="checkbox"/> Account\Manage\Index
<input checked="" type="checkbox"/> Account\Manage\PersonalData	<input checked="" type="checkbox"/> Account\Manage\ResetAuthenticator	<input checked="" type="checkbox"/> Account\Manage\SetPassword
<input checked="" type="checkbox"/> Account\Manage\ShowRecoveryCodes	<input checked="" type="checkbox"/> Account\Manage\TwoFactorAuthentication	<input checked="" type="checkbox"/> Account\Register
<input checked="" type="checkbox"/> Account\RegisterConfirmation	<input checked="" type="checkbox"/> Account\ResendEmailConfirmation	<input checked="" type="checkbox"/> Account\ResetPassword
<input checked="" type="checkbox"/> Account\ResetPasswordConfirmation		

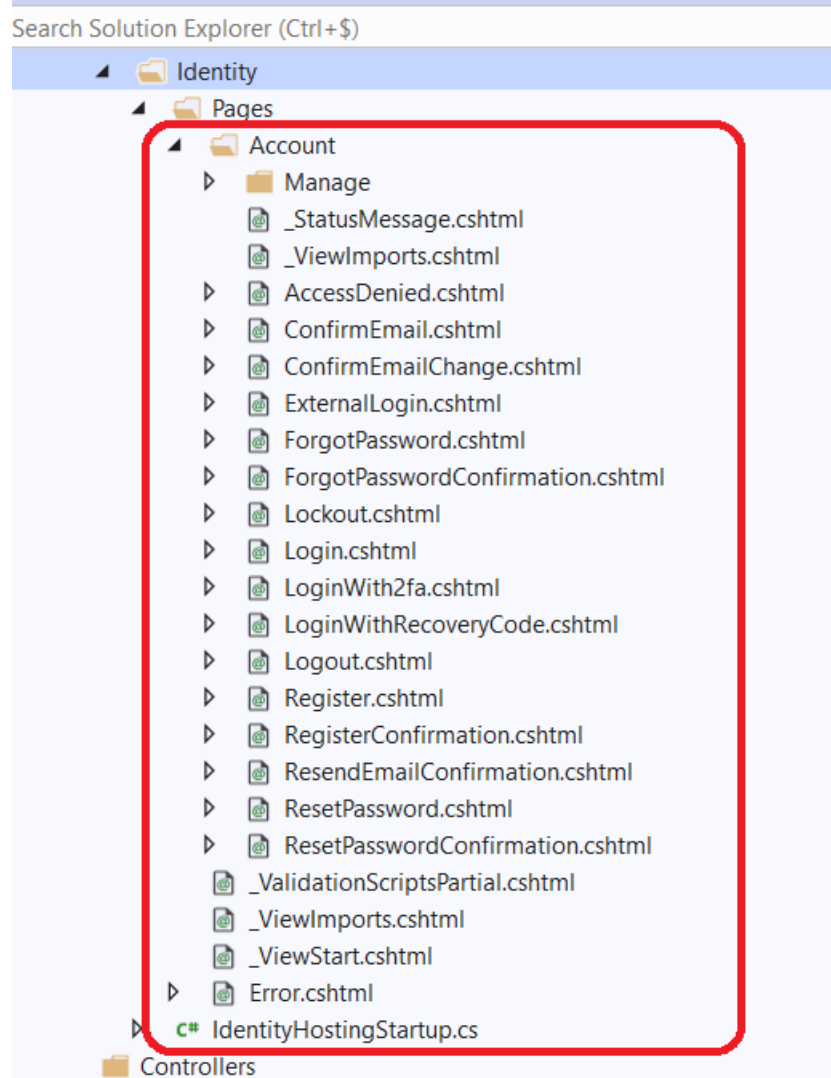
Data context class: +

☐ Use SQLite instead of SQL Server

User class: +

Klik op de button Add

Alle geselecteerde Razor pages worden aangemaakt onder Identity/Pages onder een nieuwe folder Account:



7 ASP.NET Identity Core data tables aanmaken

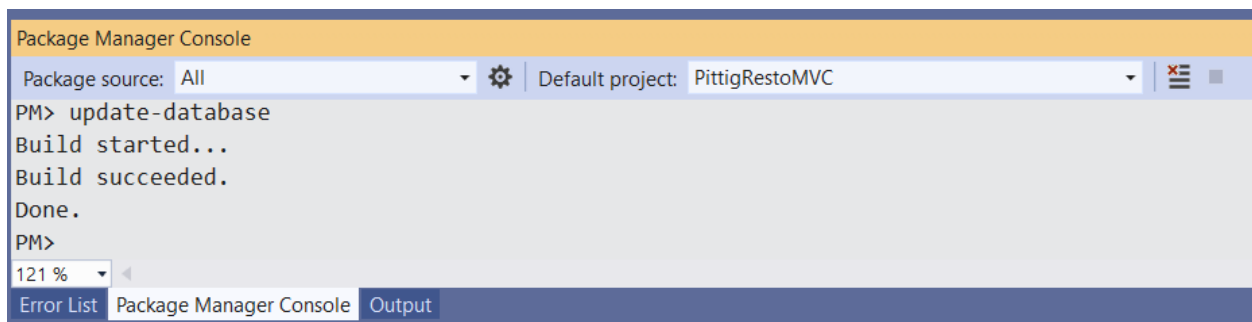
ASP.Net Identity Core heeft reeds een `ApplicationDbContext` class voorzien. Deze class is afgeleid van `IdentityDbContext`. Deze ApplicationDbContext class kan worden gebruikt voor zowel de Identity data tables als de web content data tables aan te spreken.

Er is eveneens bij aanmaak van de MVC Web app een **Migrations folder** voorzien.

In de **000...CreateIdentitySchema.cs** file staat de `CreateIdentitySchema` class gedefinieerd. Daarin zien we de **Up()** methode die de Identity datatables (`AspNetRoles`, `AspNetUsers`,...) aanmaakt via **Fluent API** code.

We gaan de Methode **Up()** laten uitvoeren d.m.v een **commando in de Package Manager Console**:

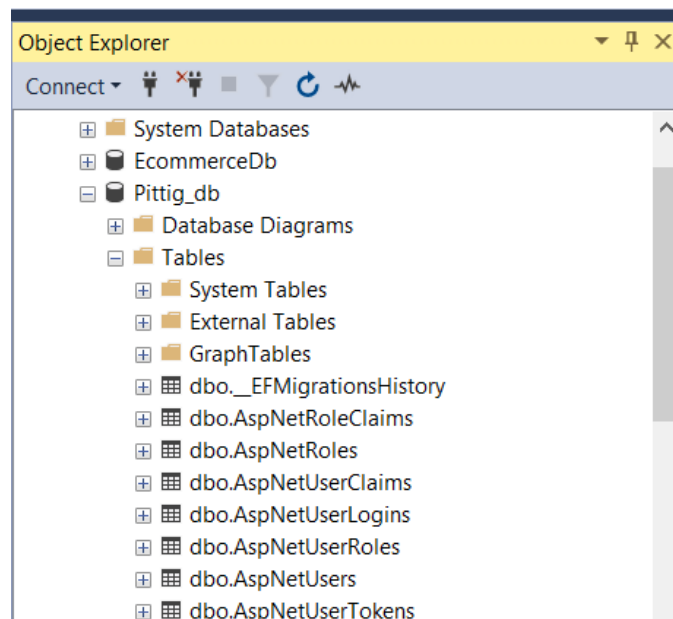
Update-database



```
Package Manager Console
Package source: All | Default project: PittigRestoMVC
PM> update-database
Build started...
Build succeeded.
Done.
PM>
```

Indien je geen foutmeldingen krijgt, zijn nu de **Identity Tabellen aangemaakt in de database PittigRestoMVC**.

Controleer dit:



8 Aanmaken van Model classes

Voor ons content management functionaliteit gaan we o.a. de volgende gegevens bewaren in de database:

-**Personeelsgegevens** (Naam, Adres, IsActief,...) en welke rol vervullen ze (Balie-,Keukenmedewerker of manager)

-**Menu's** (Kip Curry, Salade César,...)

-**Categoriën** en **SubCategorieën** om de Menu's te catalogeren (bv Voorgerecht/salade,...)

We gaan classes aanmaken onder de folder Models en deze mappen (via data-annotatie-attributen) op database tabellen

Maak onder de folder Models de volgende classes:

```
namespace PittigRestoMVC.Models
{
    public class Category
    {
        [Key]
        public int Id { get; set; }

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

namespace PittigRestoMVC.Models
{
    public class SubCategory
    {
        [Key]
        public int Id { get; set; }

        [Display(Name = "Naam Sub categorie")]
        [Required]
        public string Name { get; set; }

        [Required]
        [Display(Name = "Categorie")]
        public int CategoryId { get; set; }

        [ForeignKey("CategoryId")]
        public virtual Category Category { get; set; }
    }
}
```

```

namespace PittigRestoMVC.Models
{
    public class MenuItem
    {
        public int Id { get; set; }

        [Required]
        [Display(Name = "Naam")]
        public string Name { get; set; }
        [Display(Name = "Beschrijving")]
        public string Description { get; set; }
        [Display(Name = "Pittigheid")]
        public string Spicyness { get; set; }
        public enum ESpicy { NA = 0, Geen = 1, Weinig = 2, Sterk = 3 }

        [Display(Name = "Foto")]
        public string Image { get; set; }

        [Display(Name = "Categorie")]
        public int CategoryId { get; set; }

        [ForeignKey("CategoryId")]
        public virtual Category Category { get; set; }

        [Display(Name = "Sub categorie")]
        public int SubCategoryId { get; set; }

        [ForeignKey("SubCategoryId")]
        public virtual SubCategory SubCategory { get; set; }

        [Range(1, int.MaxValue, ErrorMessage = " Prijs moet hoger zijn dan €{1}")]
        [Display(Name = "Prijs (EUR)")]
        public double Price { get; set; }
    }
}

namespace PittigRestoMVC.Models
{
    public class ApplicationUser : IdentityUser
    {
        public string Name { get; set; }
        public string StreetAddress { get; set; }
        public string City { get; set; }
        public string State { get; set; }
        public string PostalCode { get; set; }
    }
}

```

De class ApplicationUser is afgeleid van IdentityUser. Deze zorgt ervoor dat de extra gegevens over een geregistreerde gebruiker/personneelslid zal worden bijgezet in de datatabel **AspNetUsers**.

9 Aanpassen van ApplicationDbContext class

Voeg de volgende Public properties aan de ApplicationDbContext class toe

```
public DbSet<Category> Category { get; set; }  
public DbSet<SubCategory> SubCategory { get; set; }  
public DbSet<MenuItem> MenuItem { get; set; }  
public DbSet<ApplicationUser> ApplicationUser { get; set; }
```

10 Zet migrations op voor het toevoegen van de nieuwe tabellen:

Voer het volgende commando uit in de Package Manager Console

Add-Migration AddCatSubCatMenuAppUser

Onder de folder **Migrations** is een bestand **xxxxxxxxxx_AddCatSubCatMenuAppUser.cs** bijgekomen

Open het bestand Migrations/xxxxxxxxxx_AddCatSubCatMenuAppUser.cs

Vervang in de Fluent API gegenereerde code voor de tabellen SubCategory en MenuItem bij de onDelete action van de ForeignKey-relatie

ReferentialAction.Cascade Door ReferentialAction.NoAction

(Dit is om geen foutmeldingen te krijgen bij het uitvoeren van deze code wanneer we in de volgende stap het commando update-database gaan uitvoeren)

```
migrationBuilder.CreateTable(
    name: "SubCategory",
    columns: table => new
    {
        Id = table.Column<int>(nullable: false)
            .Annotation("SqlServer:Identity", "1, 1"),
        Name = table.Column<string>(nullable: false),
        CategoryId = table.Column<int>(nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_SubCategory", x => x.Id);
        table.ForeignKey(
            name: "FK_SubCategory_Category_CategoryId",
            column: x => x.CategoryId,
            principalTable: "Category",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
    });

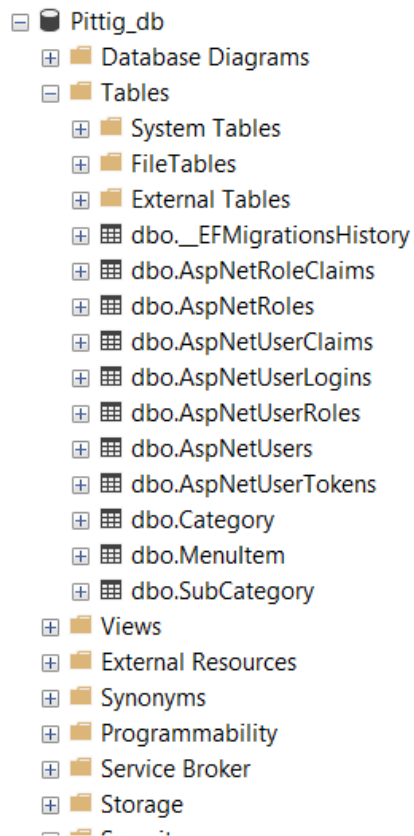
migrationBuilder.CreateTable(
    name: "MenuItem",
    columns: table => new
    {
        Id = table.Column<int>(nullable: false)
            .Annotation("SqlServer:Identity", "1, 1"),
        Name = table.Column<string>(nullable: false),
        Description = table.Column<string>(nullable: true),
        Spicyness = table.Column<string>(nullable: true),
        Image = table.Column<string>(nullable: true),
        CategoryId = table.Column<int>(nullable: false),
        SubCategoryId = table.Column<int>(nullable: false),
        Price = table.Column<double>(nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_MenuItem", x => x.Id);
        table.ForeignKey(
            name: "FK_MenuItem_Category_CategoryId",
            column: x => x.CategoryId,
            principalTable: "Category",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
        table.ForeignKey(
            name: "FK_MenuItem_SubCategory_SubCategoryId",
            column: x => x.SubCategoryId,
            principalTable: "SubCategory",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
    });
```

11 Maak de tabellen Category, SubCategory, MenuItem aan en Update tabelAspNetUsers

Build de code en voer het volgende commando uit in de **Package Manager Console**:

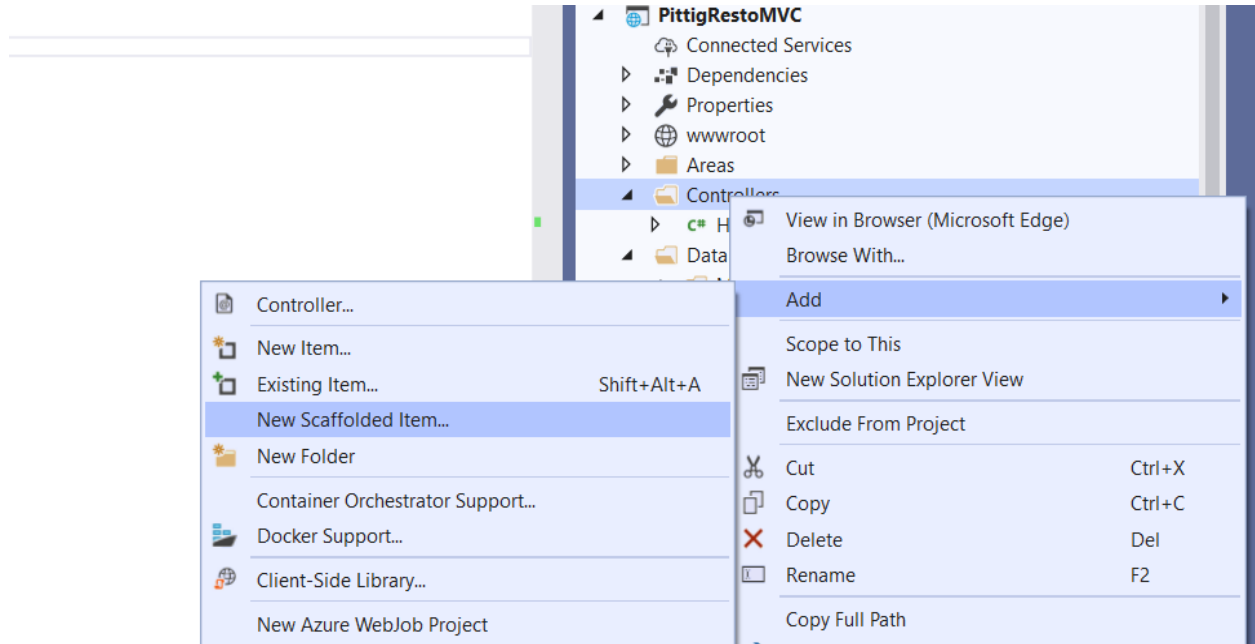
Update-database

De tabellen Category, SubCategory en MenuItem zijn nu aangemaakt in de database en AspNetUsers tabel is aangepast met extra kolommen

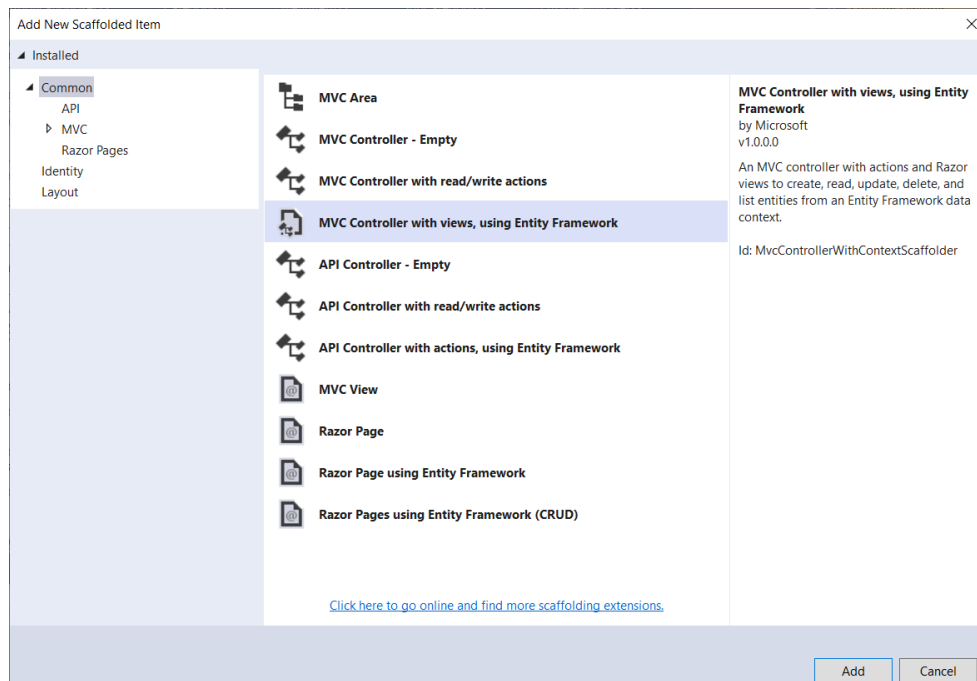


12 Maak de CategoryController aan met CRUD acties en bijhorende Views

Rechtsklik op de **folder Controllers** en Kies **Add/New Scaffolded Item...**



In het popup-venster, Kies **MVC Controller with views, using Entity Framework**



Klik op button Add

In het volgende popup-venster, Kies het model (**Category**) en de **ApplicationDbContext** classes. Zorg dat de Generate views en Use a layout page aangevinkt staan.

Geef als naam aan de Controller **CategoryController**

Add MVC Controller with views, using Entity Framework

Model class:

Category (PittigRestoMVC.Models)

Data context class:

ApplicationDbContext (PittigRestoMVC.Data)

+

Views:

☒ Generate views

☒ Reference script libraries

☒ Use a layout page:

...

(Leave empty if it is set in a Razor _viewstart file)

Controller name:

CategoryController

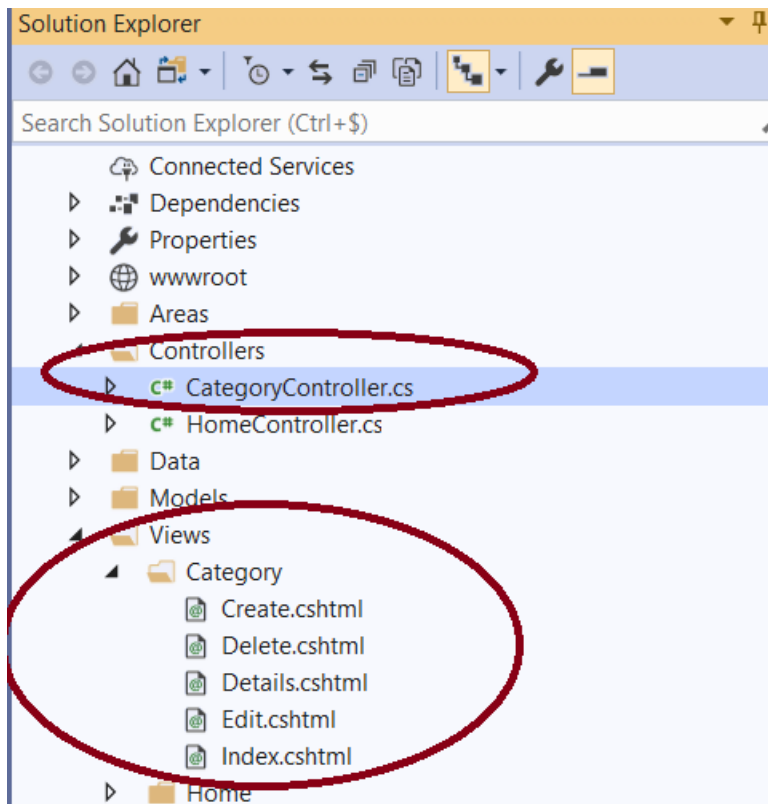
Add

Cancel

Klik op Add Button

Er is een **CategoryController** class aangemaakt onder de controllers folder

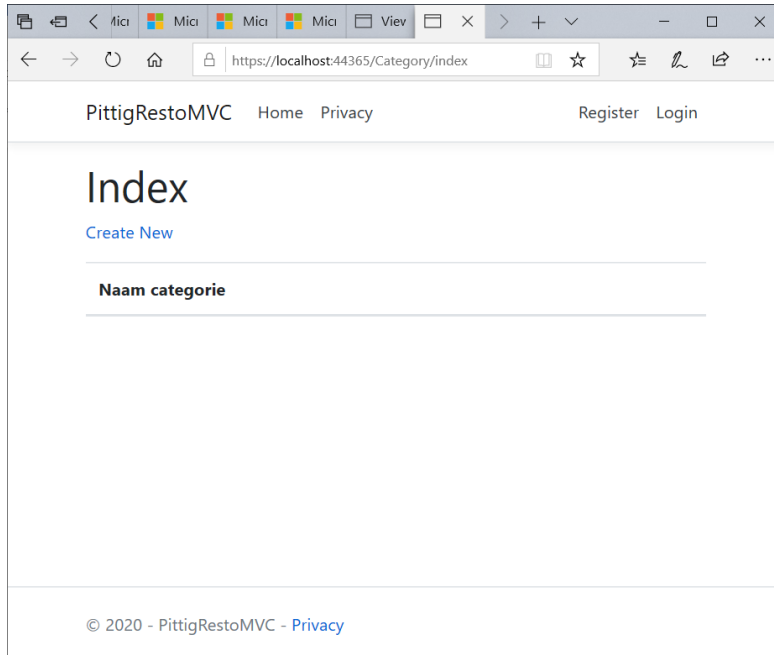
Onder de folder **Views** is een nieuwe folder **Category** aangemaakt met verschillende Razor View files:



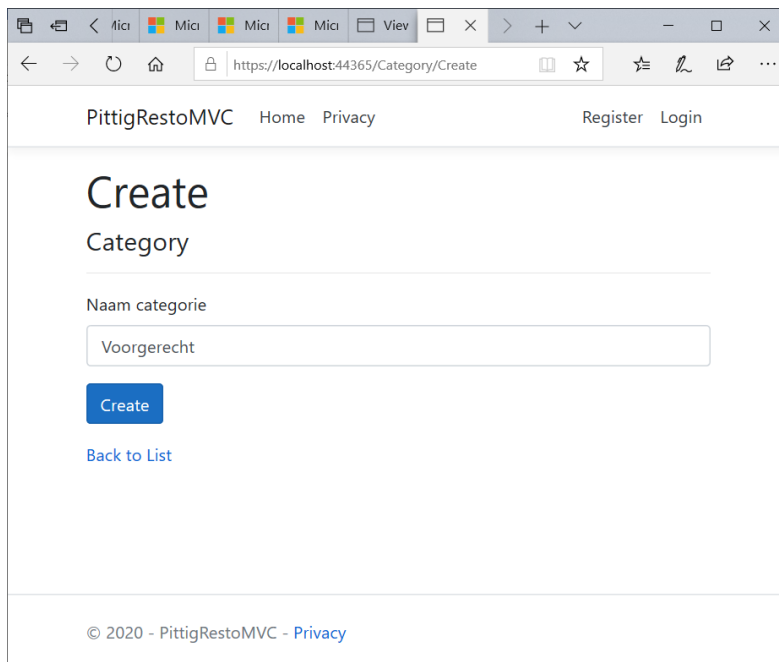
We testen nu de app. Run de app en typ als url in de browser:

<https://localhost:xxxxx/Category/index>

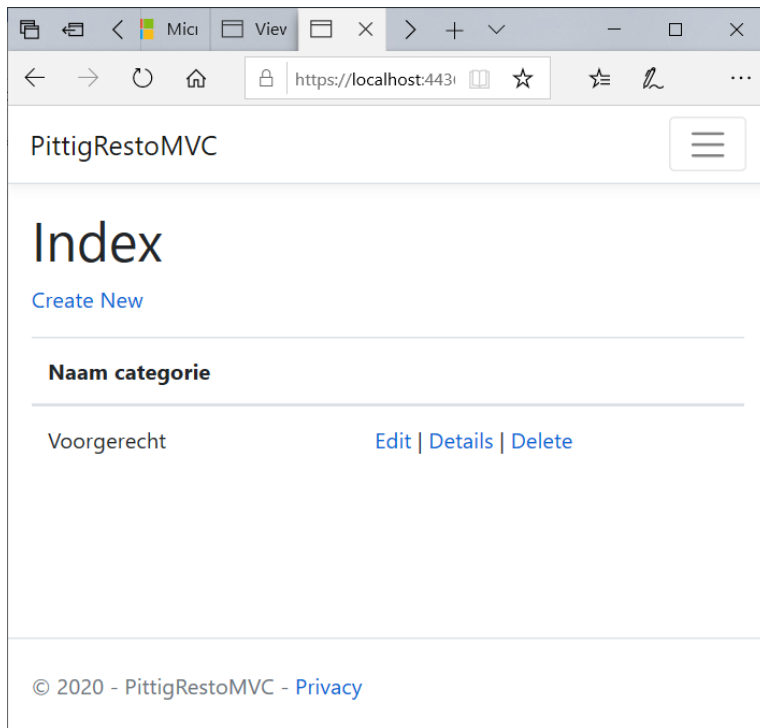
Je ziet de volgende pagina



Test de CRUD operaties en Razor Views: klik op [Create New](#) link



Klik op de **Create** button



Test ook de functionaliteit voor Edit/Details/Delete via de links op de Index view