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

<https://github.com/SyntraWestKortrijk/PittigRestoMVC-DEEL1>

# 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.

Hier kan je het resultaat van deze MVC Web app zien: <https://pittig20200419090403.azurewebsites.net/>

Een login/wachtwoord voor de rol van Manager:

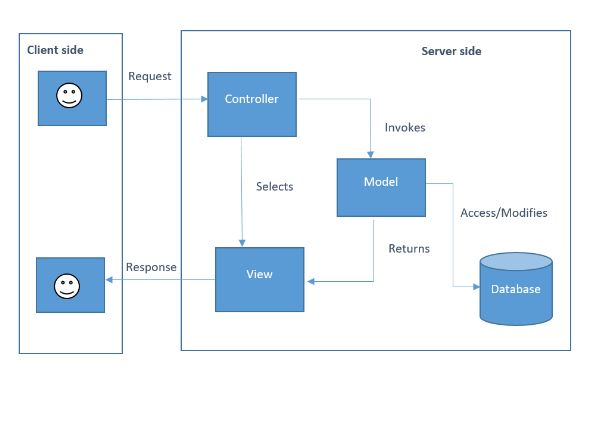
login: admin@gmail.com

pwd: Admin123\*

source code: <https://github.com/SyntraWestKortrijk/PittigResto>

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

## 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

## Routing in mvc

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

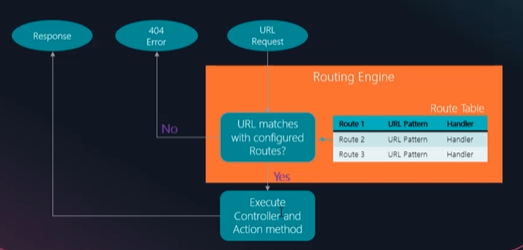
Standaard gedrag: Zal een methode Edit(…) aanroepen in de CustomersController class en een parameter (meestal een id van type int ) met waarde 101 meegeven aan deze methode Edit

**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



# Aanmaken van Sql Server database

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

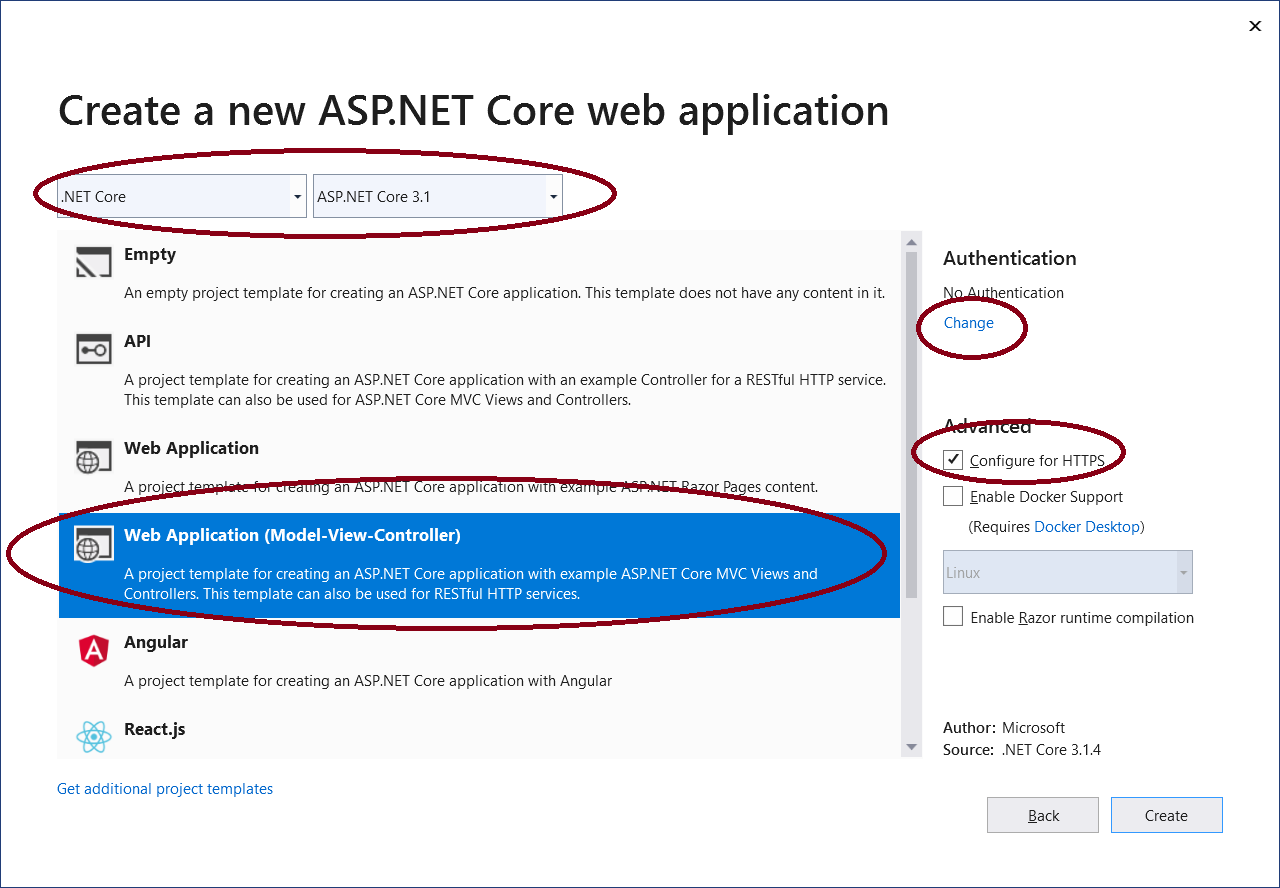
# Web App MVC PittigRestoMVC aanmaken

**Aanmaken project in VS 2019**

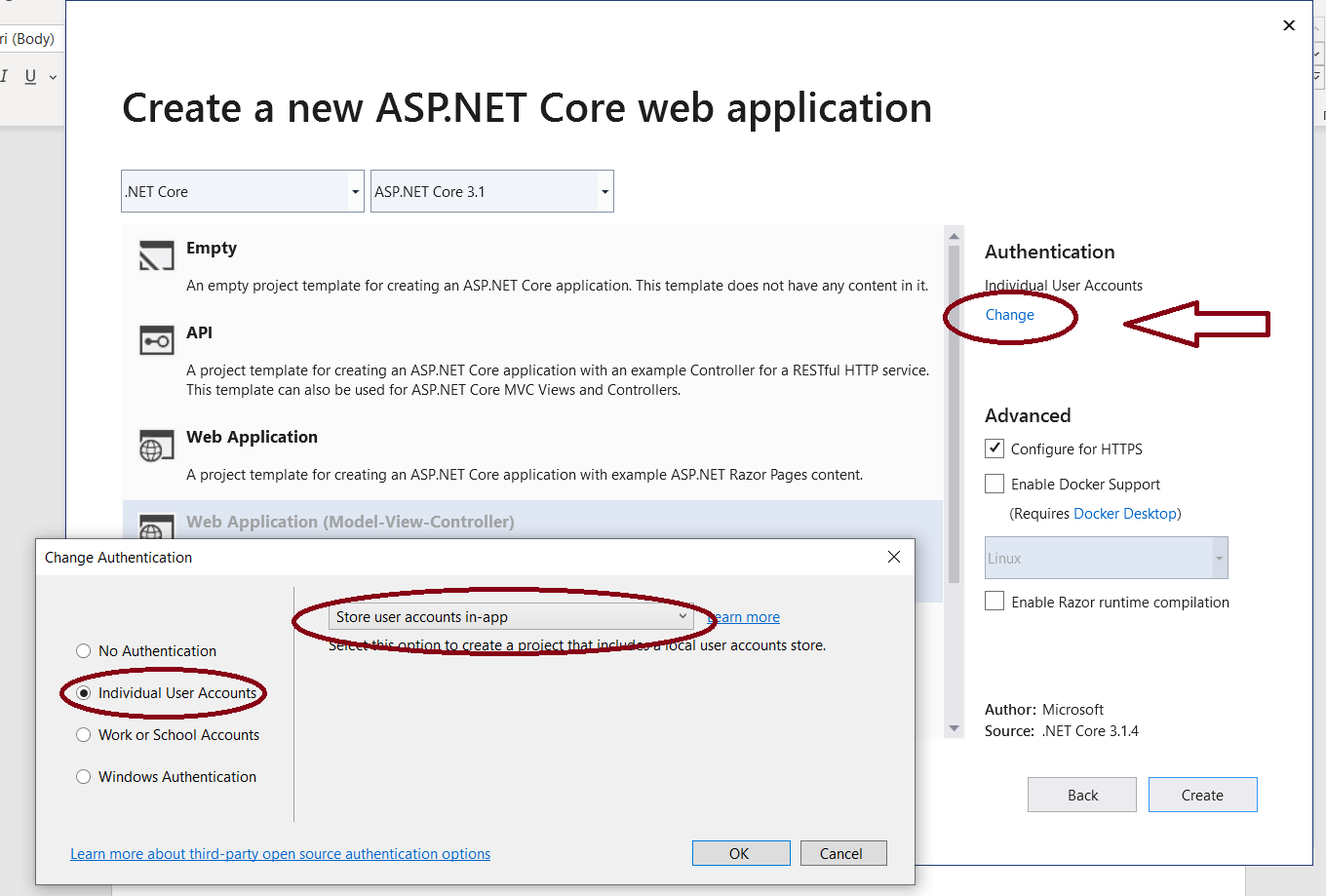
Maak in VS2019 een nieuw ASP.NET Core Web Application aan

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 Authentication 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" />

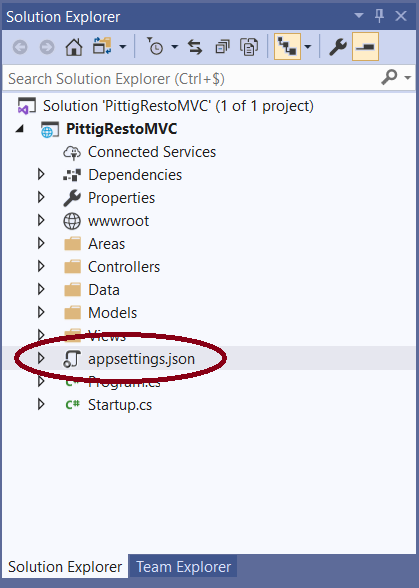
<PackageReference Include="Microsoft.AspNetCore.Identity.EntityFrameworkCore" Version="3.1.3" />

<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" />

Het project wordt 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 Pittig\_db dat je hiervoor hebt aangemaakt

"ConnectionStrings": {

"DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=**Pittig\_db**;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**();

}

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();

}

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:

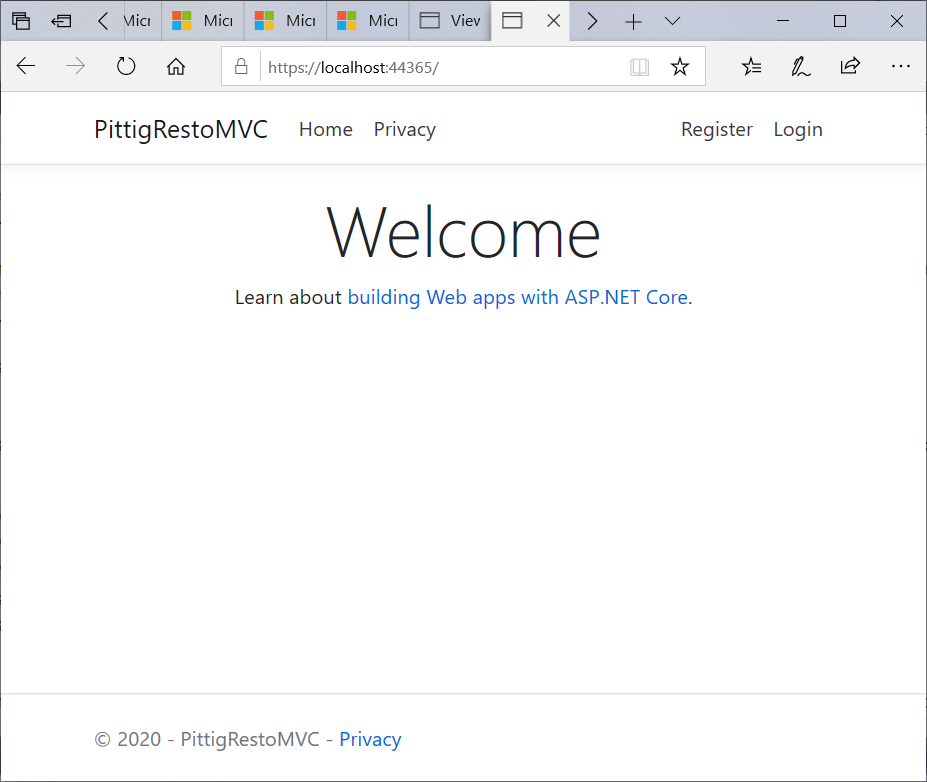
-**UseStaticFiles()**: is een middleware component die zorgt dat de HTML, CSS, JS en image files onder de wwwroot folder kunnen worden gebruikt en aangesproken door de app

-**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:



# ASP.NET Identity Core data tables aanmaken

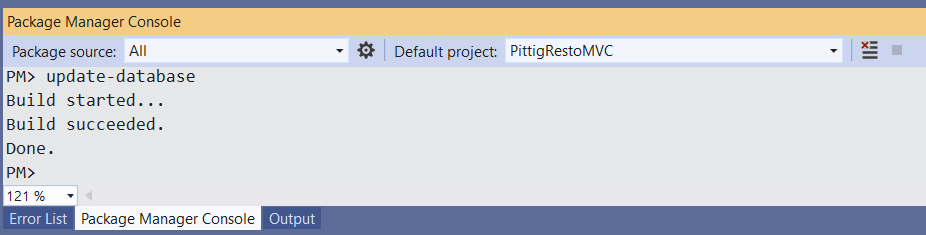
ASP.Net Identity Core heeft reeds een ApplicationDbContext class voorzien. Deze class is afgeleid van IdentityDbContext. Deze ApplicationDbContext class kan worden gebruikt voor zowel de Identity data tables als de web content data tables aan te spreken.

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.

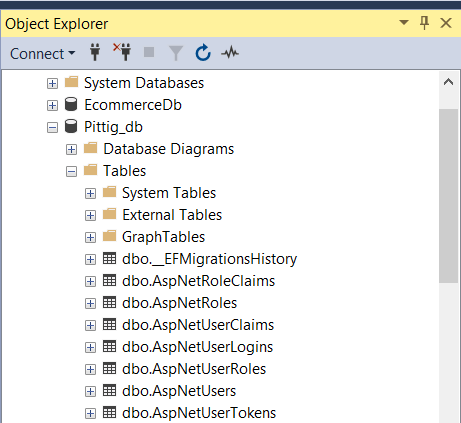
We gaan de Methode **Up**() laten uitvoeren d.m.v een **commando in de Package Manager Console**:

***Update-database***



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

Controleer dit:



# Aanmaak 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/personeelslid zal worden bijgezet in de datatabel AspNetUsers**.**

# 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; }

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

Voer het volgende commando uit in de Package Manager Console

***Add-Migration AddCatSubCatMenuAppUser***

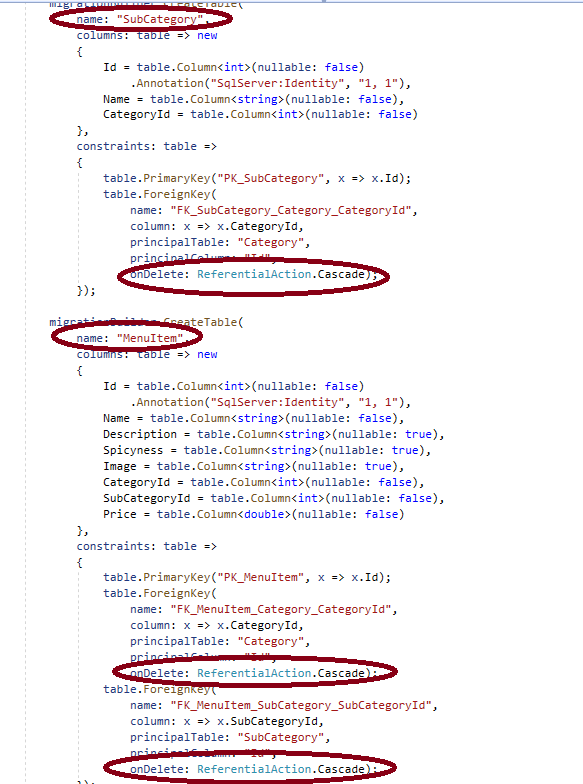
Onder de folder **Migrations** is een bestand **xxxxxxxxxx\_AddCatSubCatMenuAppUser.cs** bijgekomen

**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)***

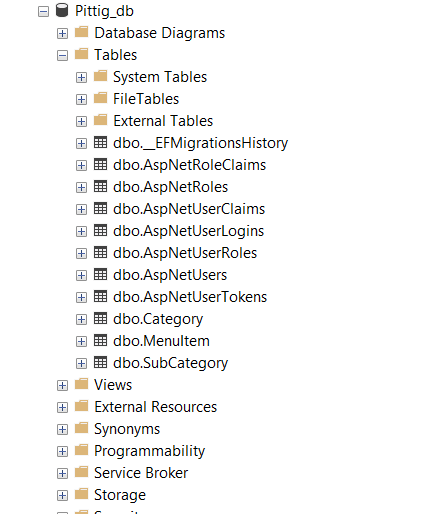
****

# Maak de tabellen Category,SubCategory, MenuItem aan en Update tabel AspNetUsers

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

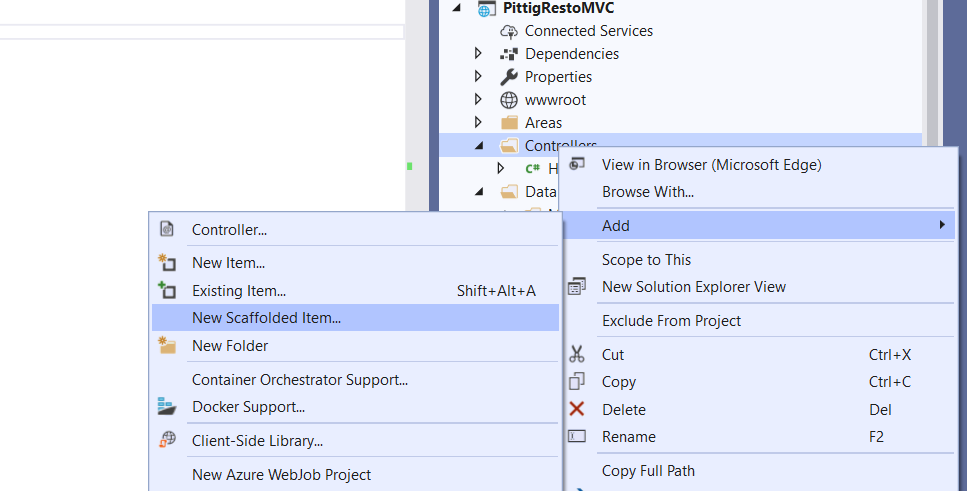
***Update-database***

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

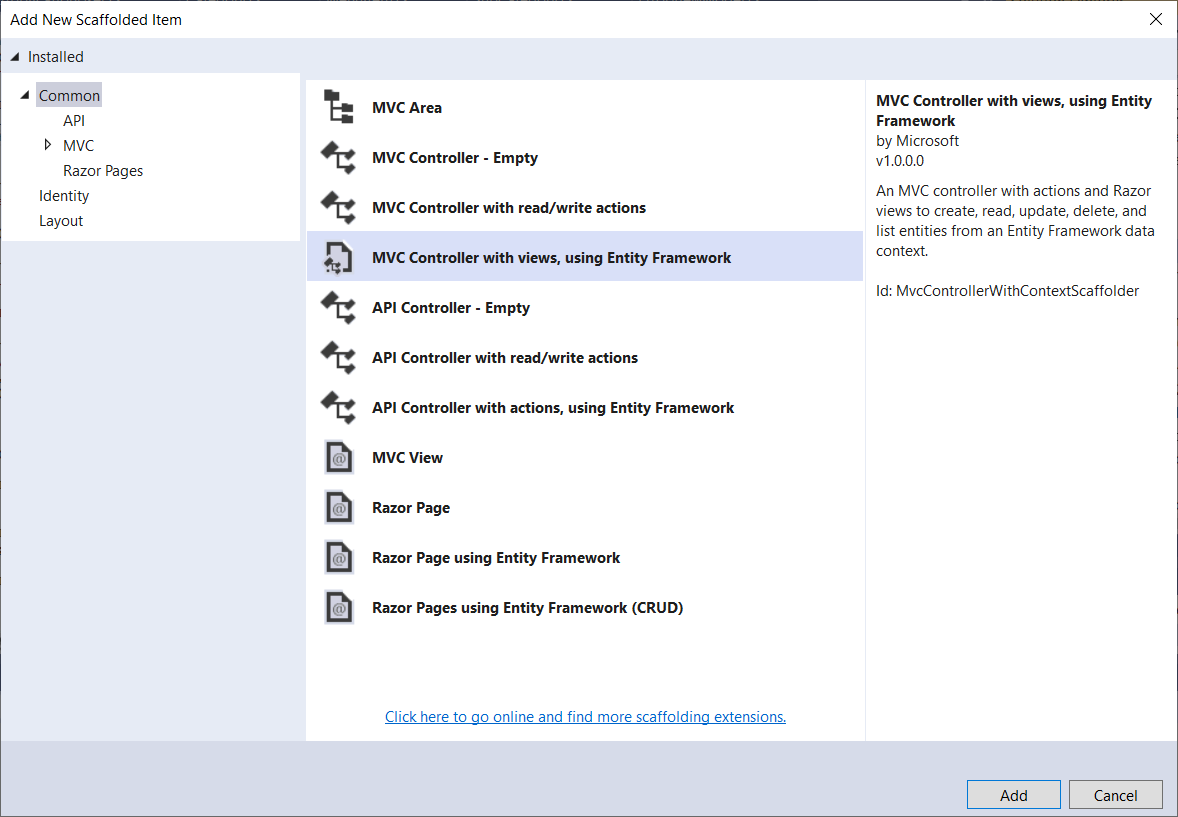


# Maak de CategoryController aan met CRUD acties en bijhorende Views

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



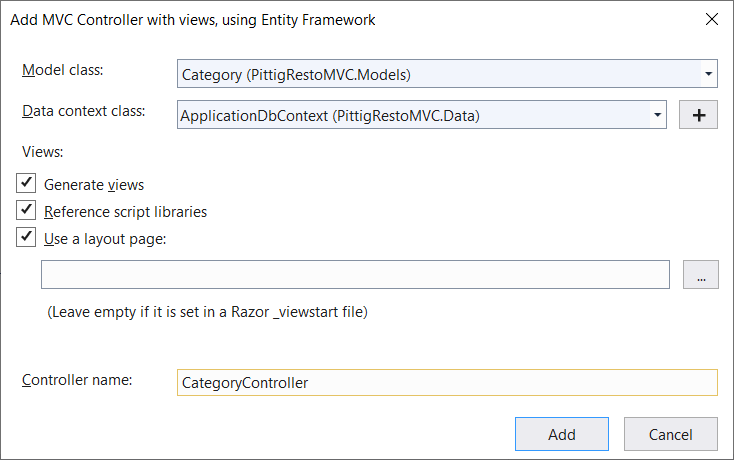
In het popup-venster, Kies **MVC Controller with views, using Entity Framework**



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.

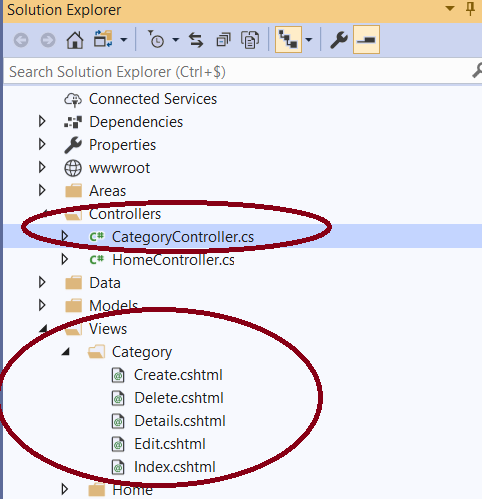
Geef als naam aan de Controller **CategoryController**



Klik op Add Button

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

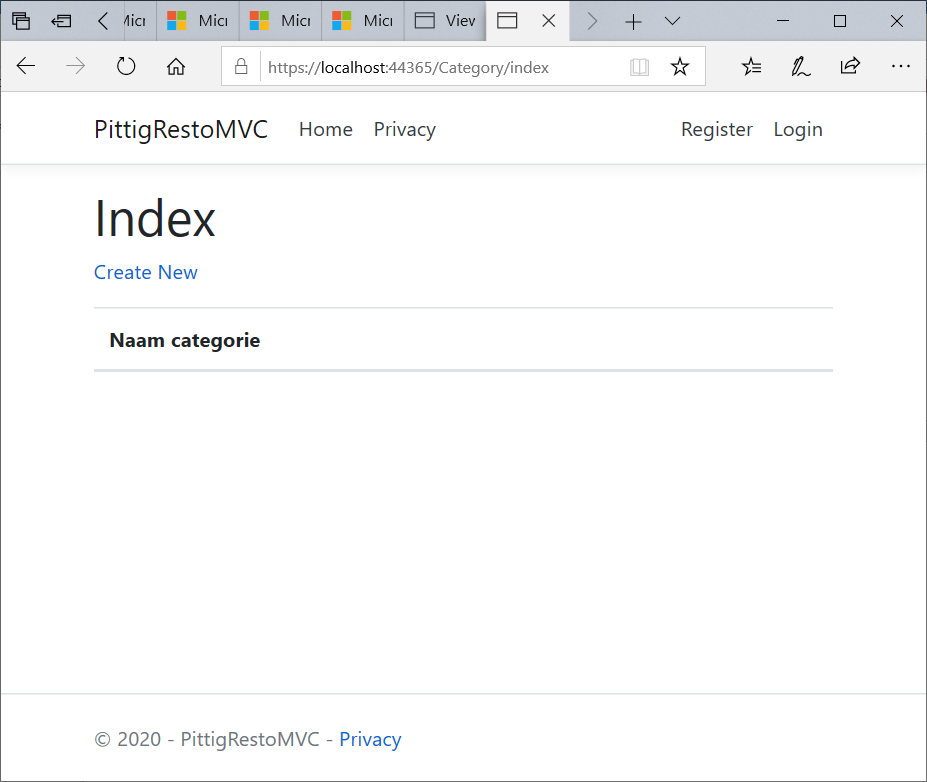
Onder de **folder Views is een nieuwe folder Category aangemaakt met verschillende Razor View files:**



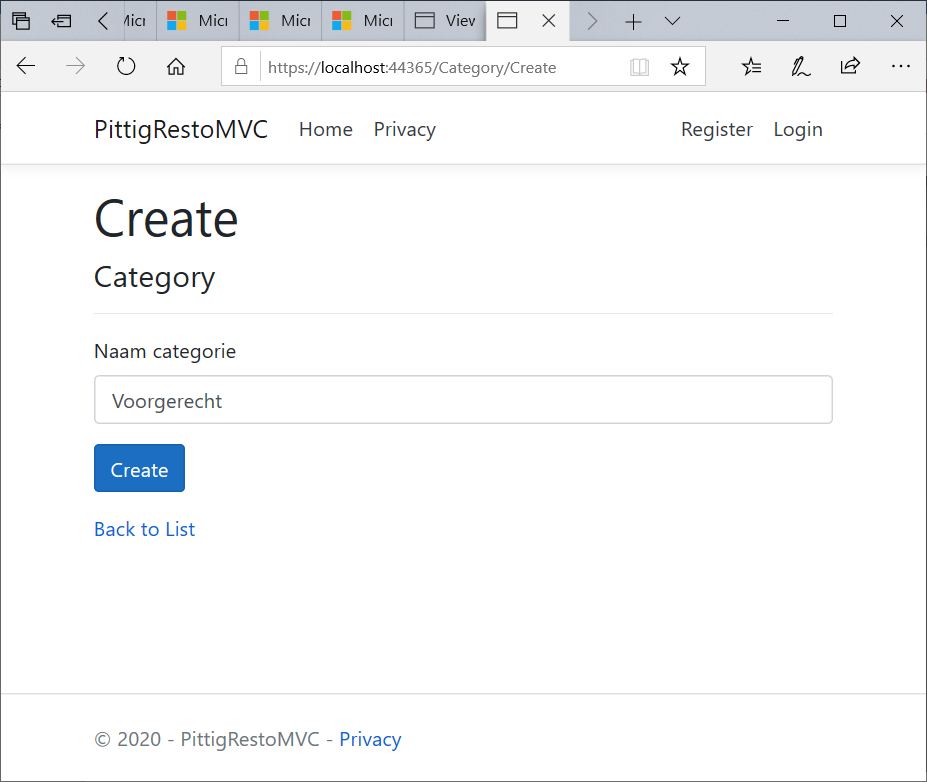
We testen nu de app. Run de app en typ als url in de browser:

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

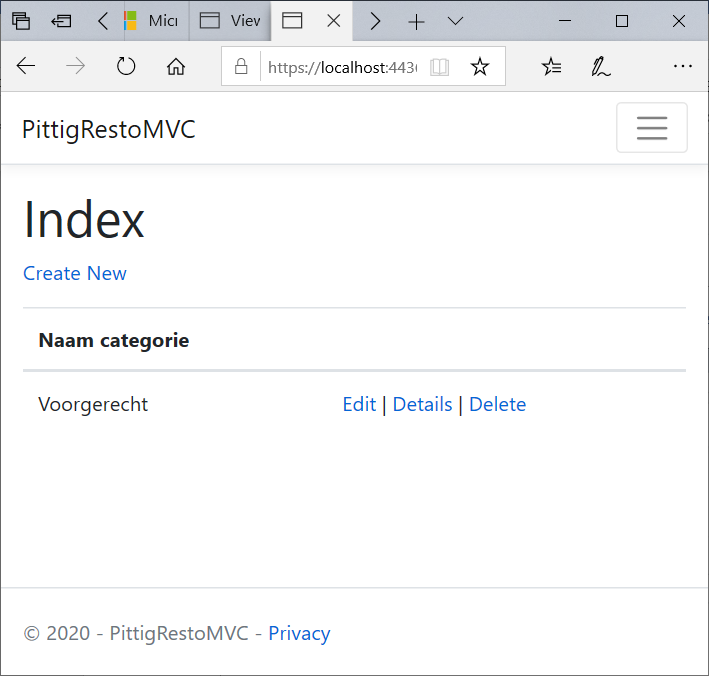
Je ziet de volgende pagina



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