

GRADUAAT IN HET
PROGRAMMEREN

Cursus C# Web – ASP.Net Core 6

Kristof Palmaers

auteur

Inhoud

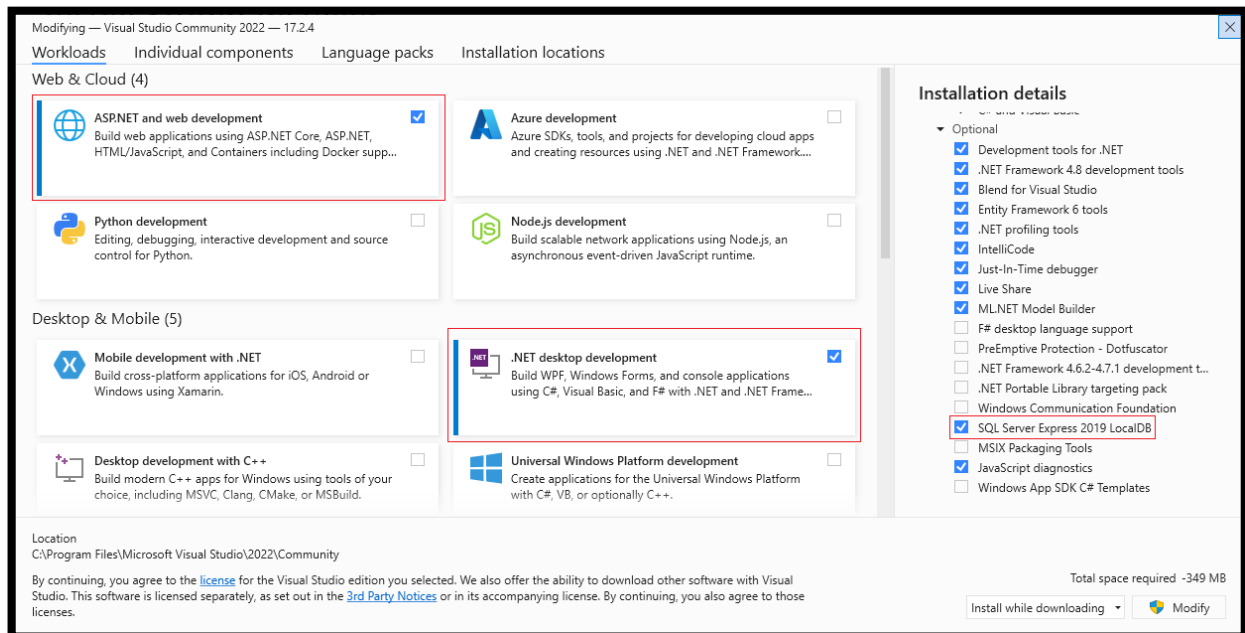
| | | |
|-------|--|----|
| 1. | Introductie: Visual Studio – C# Web | 4 |
| 1.1 | Visual Studio 2022 Installer | 4 |
| 1.2 | Visual Studio – New project..... | 4 |
| 1.3 | Program.cs | 7 |
| 1.4 | HTML5, CSS & Javascript | 7 |
| 1.4.1 | Program.cs | 8 |
| 1.4.2 | Html..... | 8 |
| 1.4.3 | Middleware..... | 8 |
| 1.5 | Nuget Package manager console | 9 |
| 2. | Dynamische HTML (Razor Pages)..... | 10 |
| 2.1 | Razor pages: inleiding..... | 10 |
| 2.2 | Razor page voorbeeld..... | 10 |
| 2.3 | Razor page project | 12 |
| 2.3.1 | Inhoud van de project folder | 13 |
| 2.3.2 | Scaffolding..... | 14 |
| 2.3.3 | Layout in ASP.Net Core..... | 15 |
| 2.3.4 | Cross-Site Request Forgery (XSRF/CSRF) attacks | 15 |
| 3. | Create a Razor Pages web app..... | 16 |
| 4. | Model View Controller (MVC) | 17 |
| 4.1 | Model | 17 |
| 4.1.1 | Model binding | 17 |
| 4.1.2 | Model validation | 17 |
| 4.2 | View..... | 18 |
| 4.2.1 | View Content..... | 18 |
| 4.2.1 | Partial view..... | 18 |
| 4.3 | Controller..... | 18 |
| 4.4 | Routing | 19 |
| 4.5 | Navigation View component | 19 |
| 4.6 | Tag Helper..... | 20 |
| 4.6.1 | Anchor tag helper..... | 20 |
| 4.6.2 | Form tag helper | 20 |
| 4.6.3 | Custom tag helper | 22 |
| 4.7 | MVC – Oefening Party Invites..... | 22 |

| | | |
|------|---|----|
| 4.8 | MVC – Oefening Sportstore | 22 |
| 5. | Entity Framework Core (EF Core)..... | 23 |
| 5.1 | Sql server | 23 |
| 5.2 | In memory | 23 |
| 5.3 | Reverse engineering..... | 24 |
| 5.4 | Data verwerking met Linq | 24 |
| 6. | Dependency Injection (DI)..... | 25 |
| 6.1 | Service lifetime | 25 |
| 6.2 | DbContext..... | 25 |
| 6.3 | Repository..... | 25 |
| 6.4 | ASP.Net Core Identity..... | 25 |
| 6.5 | Identity toevoegen aan een MVC project | 26 |
| 6.6 | Web Application Identity - Integrated authentication..... | 26 |
| 6.7 | Async programmeren..... | 30 |
| 7. | Authorisation | 30 |
| 7.1 | Role management | 30 |
| 8. | Middleware componenten..... | 31 |
| 8.1 | Startup.cs..... | 31 |
| 8.2 | Middleware voorbeelden | 31 |
| 9. | Built-in services..... | 32 |
| | • IServiceCollection | 32 |
| 10. | Oefeningen | 33 |
| 10.1 | Party invites (Apress – Pro ASP.Net Core 6) | 33 |
| 10.2 | Razor pages - Movie | 33 |
| 10.3 | SportsStore (Apress – Pro ASP.Net Core 6) | 33 |
| 10.4 | Todo Item | 33 |
| 10.5 | Voertuig Identity | 33 |
| 10.6 | Voertuig no Identity | 33 |
| 10.7 | Feestje | 33 |
| 10.8 | Secretariaat..... | 33 |

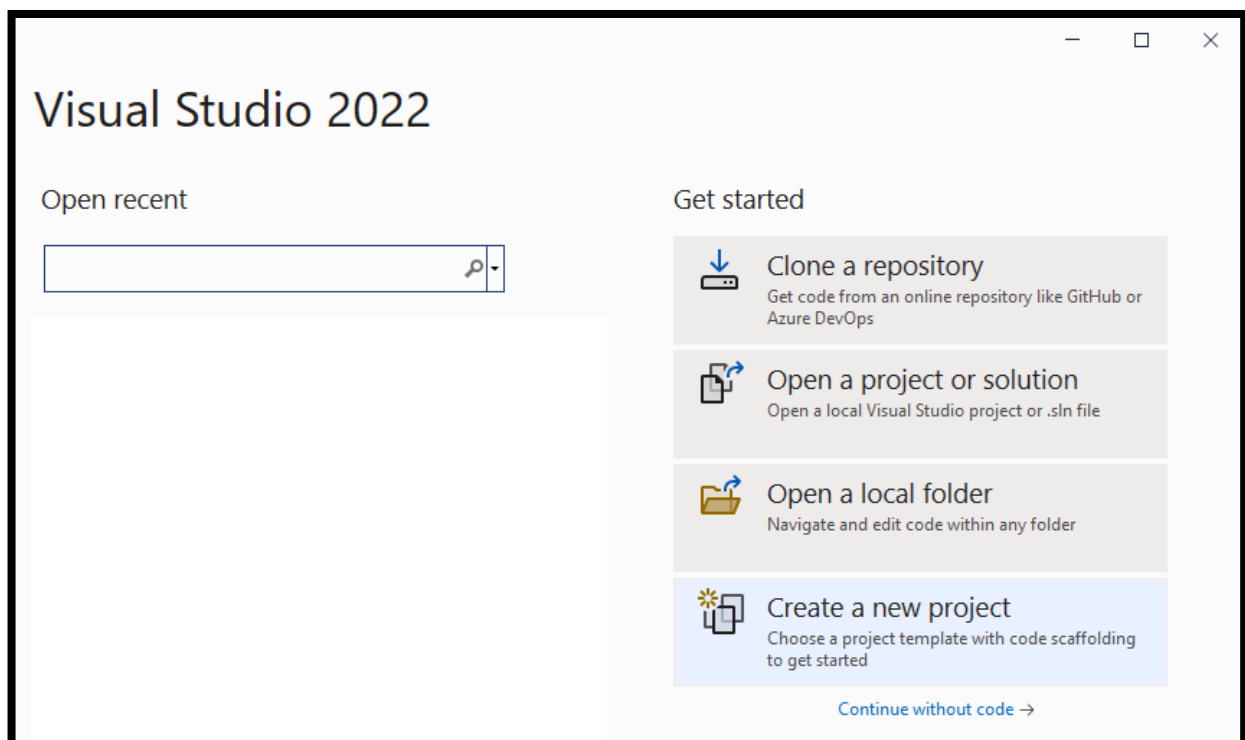
1. Introductie: Visual Studio – C# Web

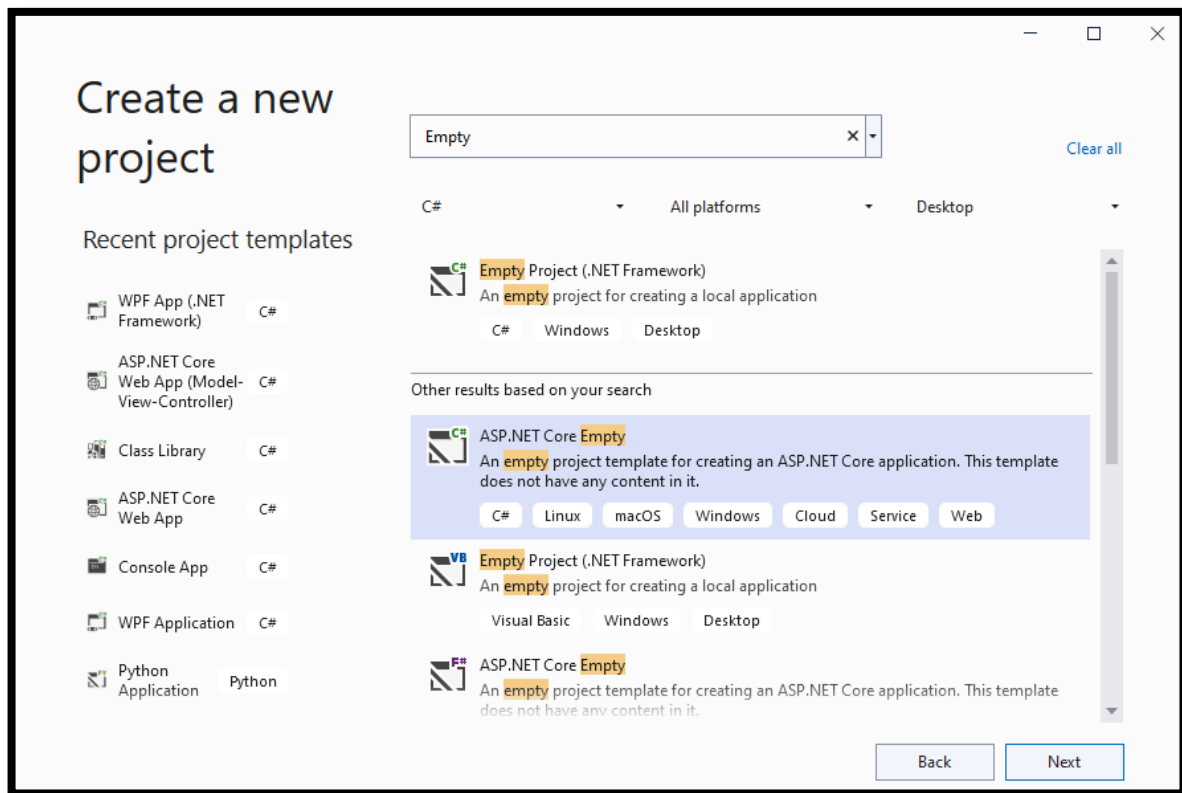
We gaan gebruik maken van Visual Studio 2022 en maken gebruik van .Net Core 6 in onze web applicaties.

1.1 Visual Studio 2022 Installer

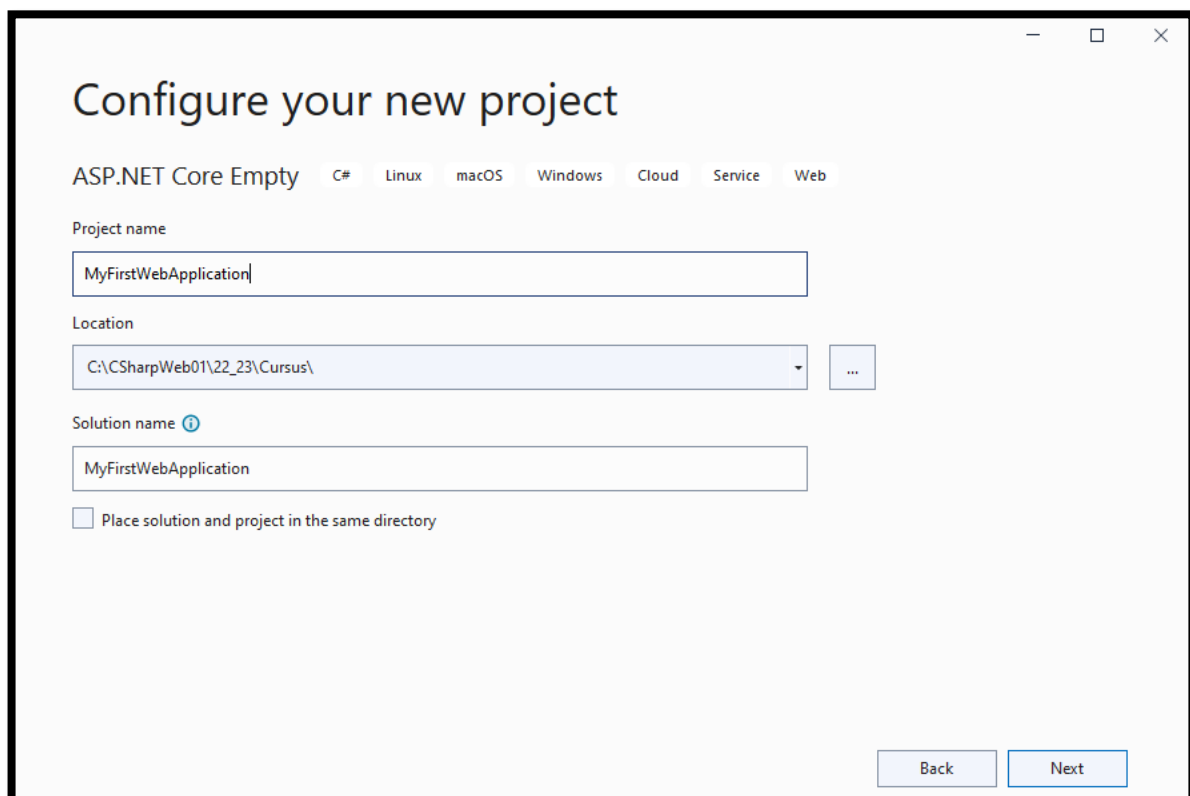


1.2 Visual Studio – New project

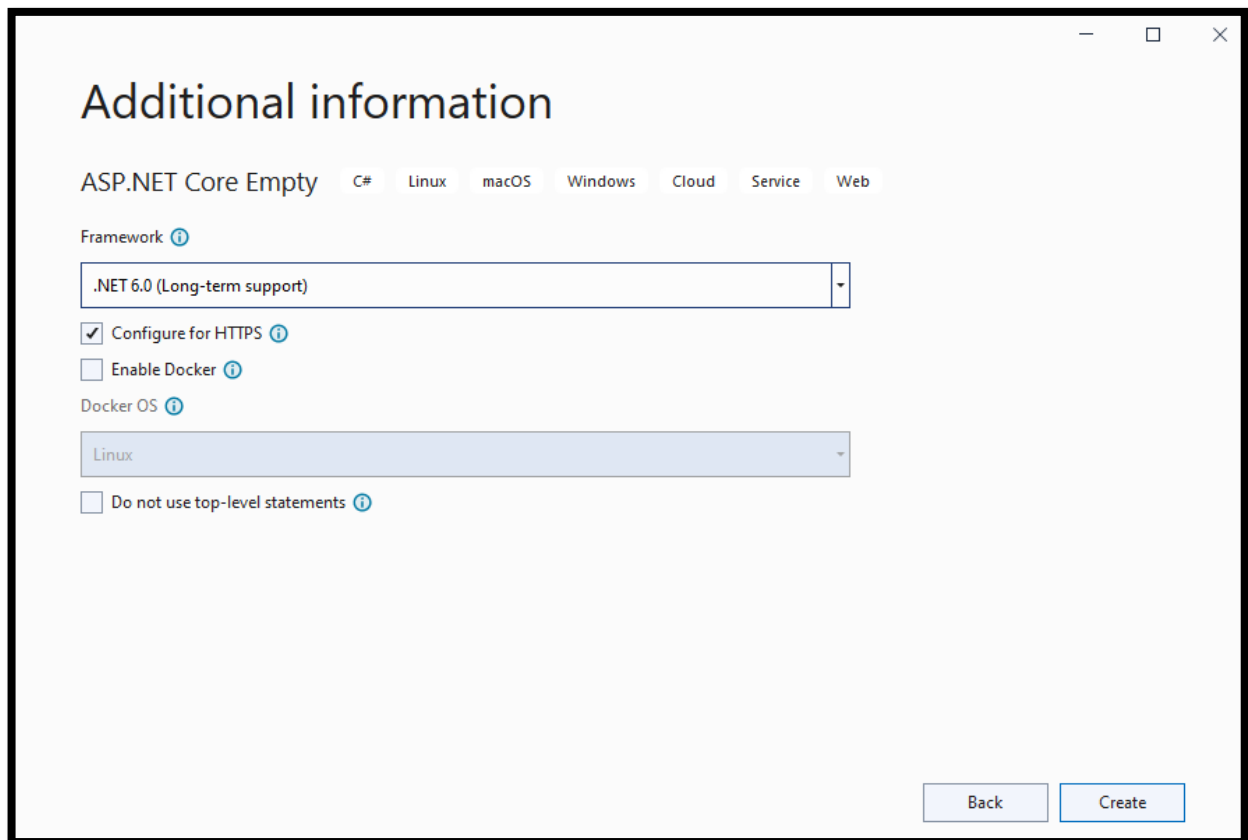




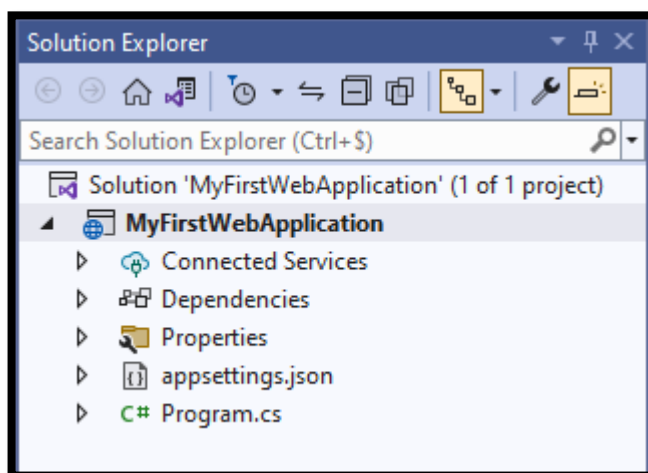
(Fig01)



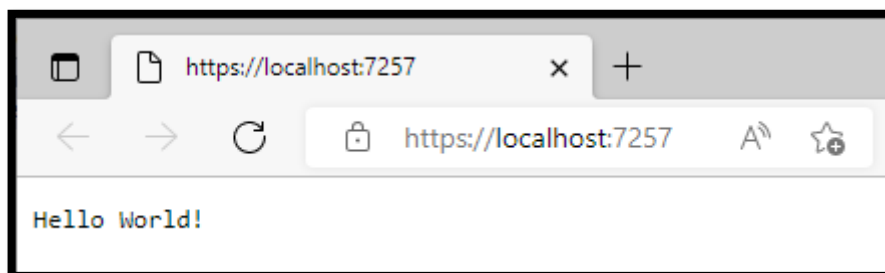
(Fig02)



(Fig03)

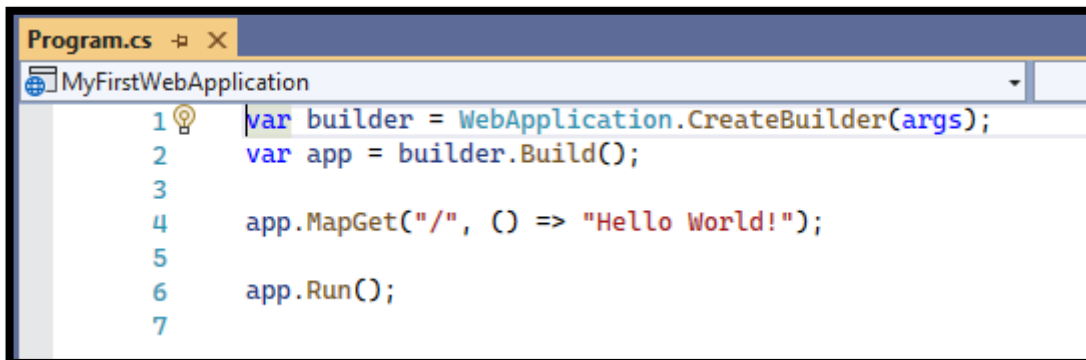


(Fig04)



(Fig05)

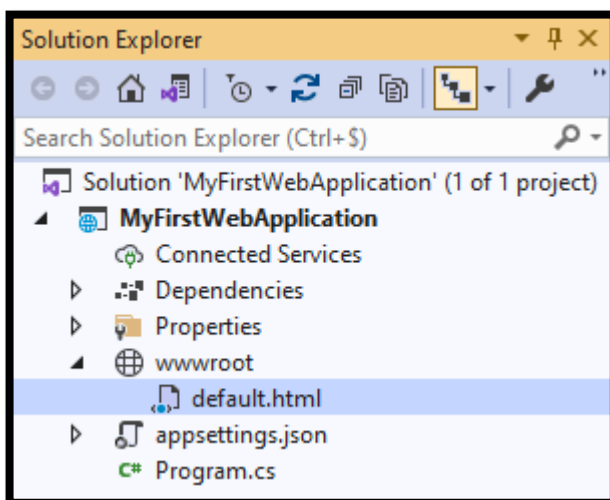
1.3 Program.cs



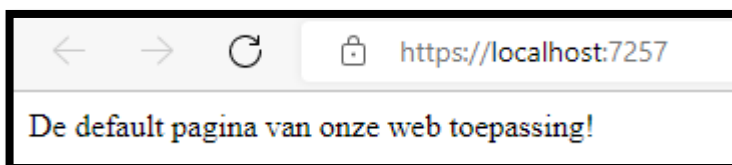
Wanneer onze web applicatie wordt gestart zal er Hello World! Getoond worden in de browser.

1.4 HTML5, CSS & Javascript

- Project – Add folder: **wwwroot**
- Folder wwwroot – Add file: **default.html**



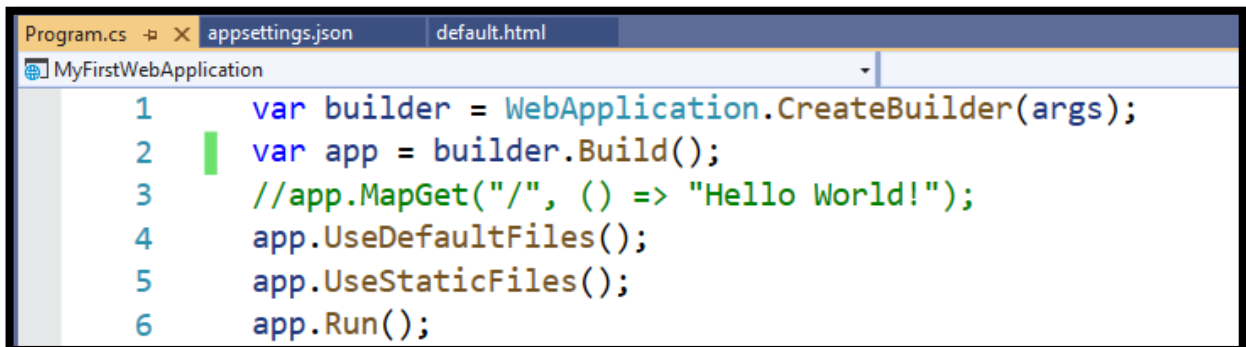
(fig06)



(fig07)

1.4.1 Program.cs

Voor deze nieuwe html pagina (fig07) te kunnen gebruiken moeten we onze configuratie aanpassen in onze Program klasse.



```
1 var builder = WebApplication.CreateBuilder(args);
2 var app = builder.Build();
3 //app.MapGet("/", () => "Hello World!");
4 app.UseDefaultFiles();
5 app.UseStaticFiles();
6 app.Run();
```

1.4.2 Html

We voegen in ons project een folder toe “wwwroot” en in deze folder voegen we een html pagina toe: “default.html”. In (fig06) kunnen we de nieuwe structuur van onze webapplicatie bekijken.

De inhoud van onze nieuwe html pagina. (default.html)

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title></title>
</head>
<body>
  De default pagina van onze web toepassing!
</body>
</html>
```

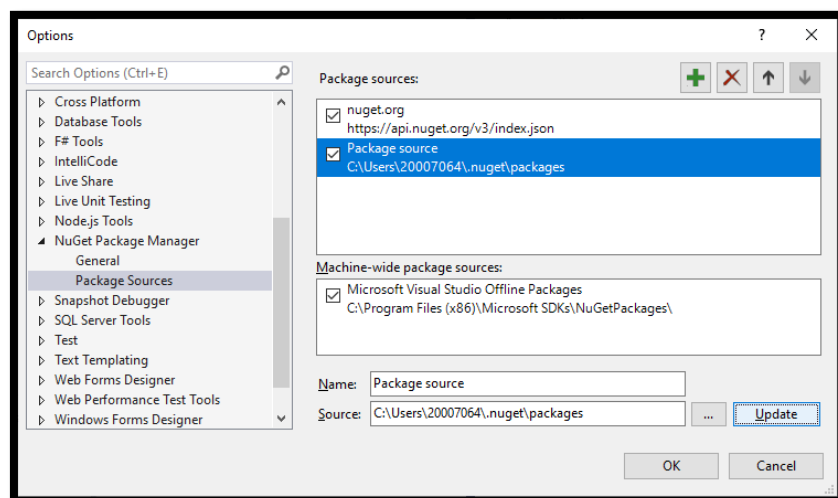
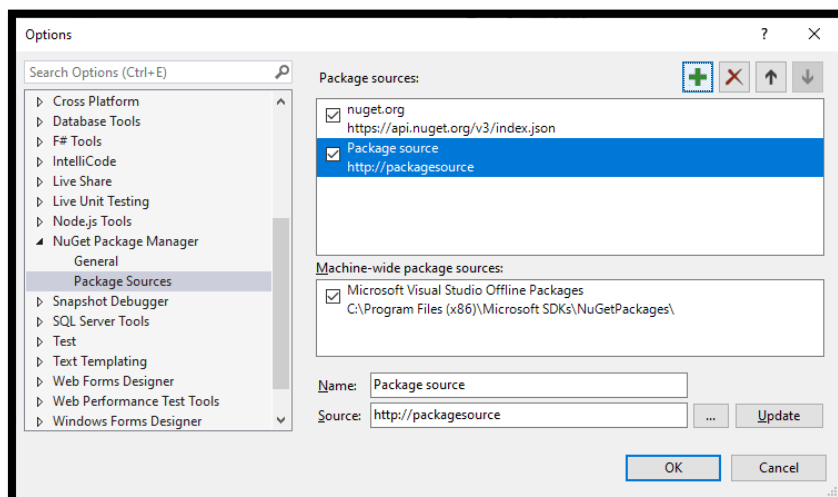
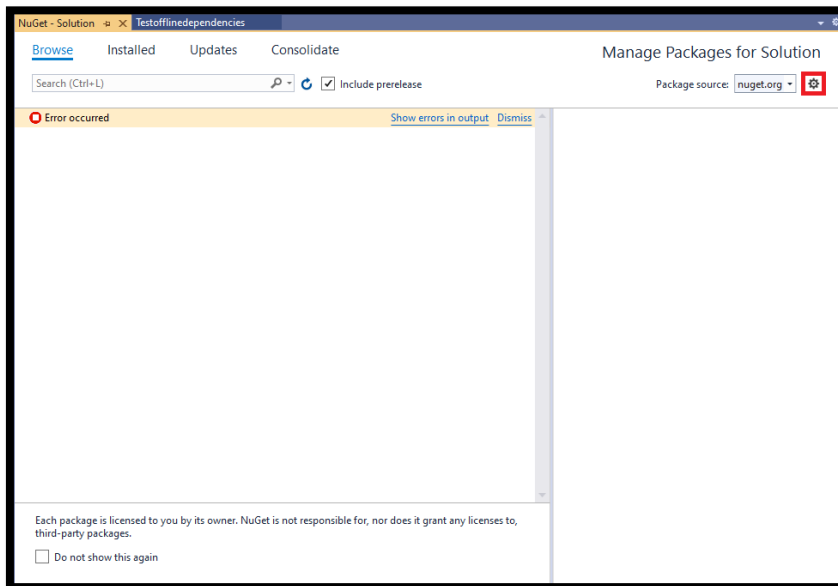
1.4.3 Middleware

- **UseDefaultFiles()**
Deze middleware gaat op zoek naar de standaard startpagina's in de wwwroot sectie.
 - Default.html()
 - Index.html()
- **UseStaticFiles();**
Deze middleware laat ons toe html files te gebruiken in onze toepassing, hij gaat steeds op zoek naar de wwwroot folder.

1.5 Nuget Package manager console

Via de Nuget Package Manager kunnen we packages toevoegen aan ons project. Ook zonder internet kunnen we de packages toevoegen aan ons project.

In Visual Studio kan je de NPM oproepen via het menu – Tools – Nuget Package Manager en dan krijgen we volgend scherm.



2. Dynamische HTML (Razor Pages)

2.1 Razor pages: inleiding

ASP.Net Core Razor pages is een server-side framework wat ons toelaat om dynamische html pagina's te genereren. Met ASP.Net core kunnen we razor pages installeren op Windows, Linux en Mac besturingssystemen. Het is dus platform onafhankelijk.

Het is een lightweight framework en is heel flexibel. De ontwikkelaar heeft de volledige controle over de gegenereerde html. Het Razor pages framework is gebouwd bovenop het MVC platform maar is ook afzonderlijk te gebruiken..

Men heeft geen kennis van MVC nodig om te kunnen werken met het Razor pages framework.

Razor pages maakt gebruik van de populaire programmeertaal C# voor het server-side programmeren.

Verder maakt men gebruik van de razor template syntax om C# rechtstreeks te implementeren in de HTML opmaak om zo dynamische content aan te maken.

Een razor page moet aan de volgende 3 karakteristieken voldoen:

- de bestandsnaam mag geen underscore bevatten
- de bestandsextensie is .cshtml
- de eerste lijn begint met @page

De @page verwijzing in de eerste lijn is verplicht. Als men dit niet doet zal de pagina niet als een razor pagina behandeld worden en zal ze ook niet gevonden worden bij het surfen naar deze pagina.

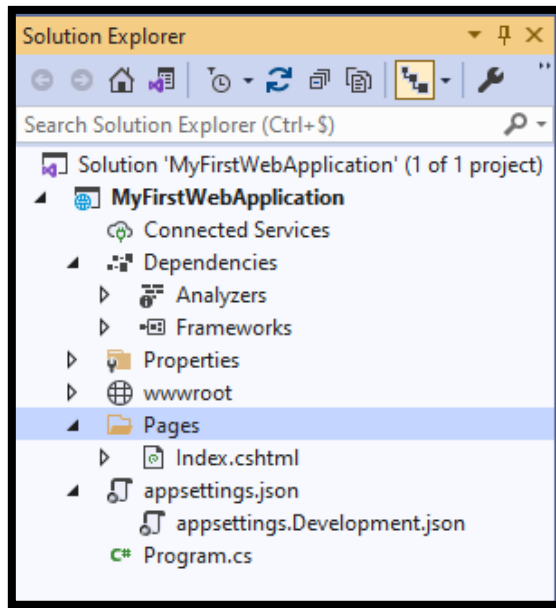
Razor content pagina's kunnen een layout pagina bevatten maar dit is niet verplicht. Optioneel kunnen ze ook code, HTML, javascript en inline razor code bevatten.

2.2 Razor page voorbeeld

Create folder Pages (fig08)

Add item – Razor page – Index.cshtml (fig08)

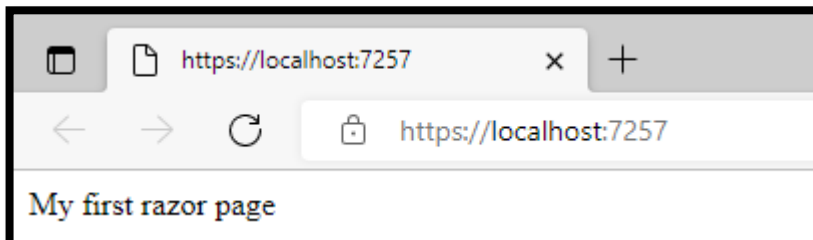
```
@page
@model MyFirstWebApplication.Pages.IndexModel
@{
}
<div>
    My first razor page
</div>
```



(fig08)

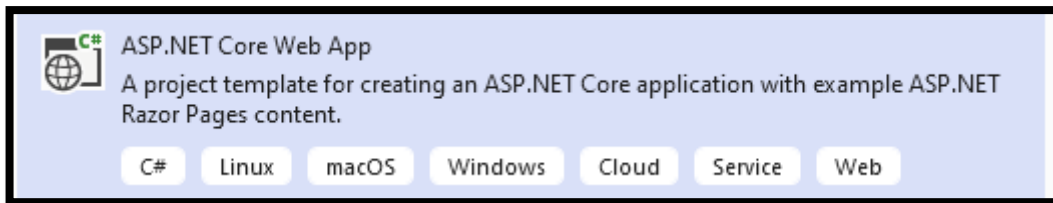
Modify Program.cs

```
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddRazorPages();
var app = builder.Build();
//app.MapGet("/", () => "Hello World!");
//app.UseDefaultFiles();
//app.UseStaticFiles();
app.MapRazorPages();
app.Run();
```

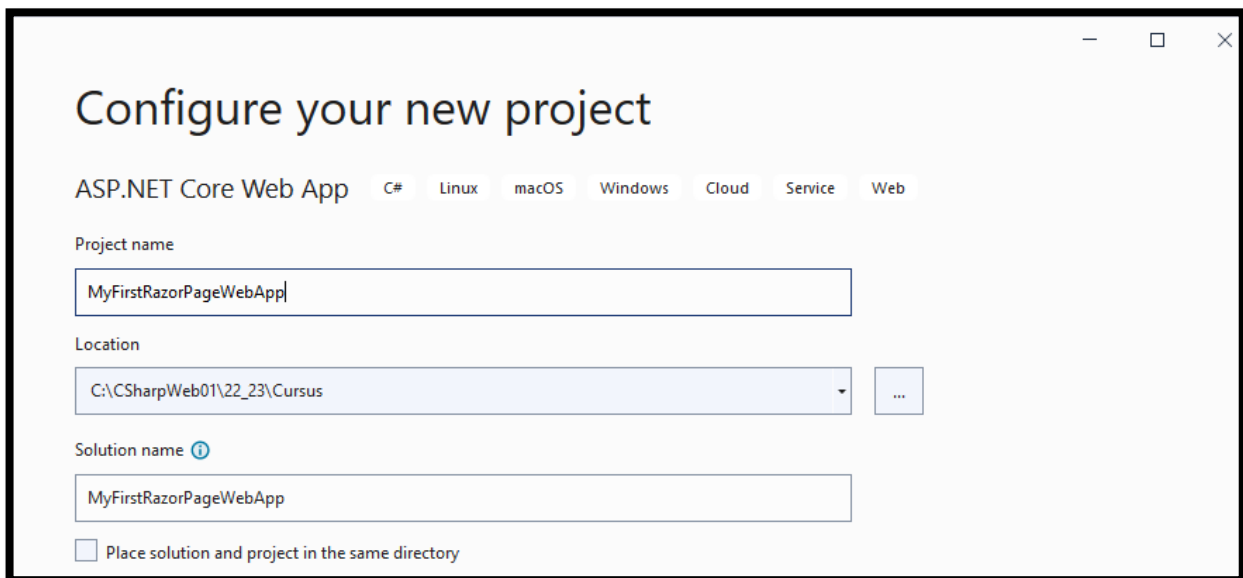


(fig09)

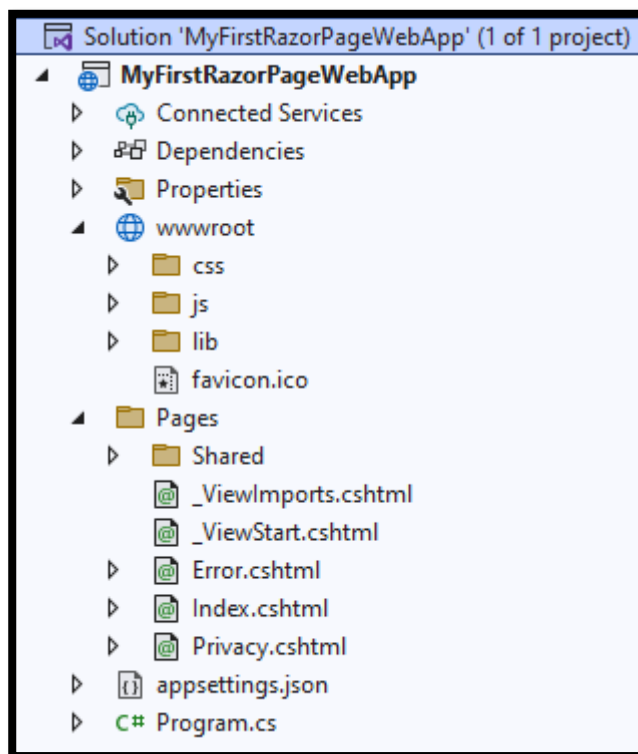
2.3 Razor page project



(fig10)



(fig11)



(fig12)

Automatisch zijn de folders `wwwroot` en `Pages` toegevoegd in het project. Er is ook een welkom pagina toegevoegd `Index.cshtml`, verder is er ook een `Error` en `privacy` pagina. In de "`Program.cs`" wordt de service ook al aangeroepen om te werken met `RazorPages` die we in het vorige project nog zelf moesten toevoegen.

2.3.1 Inhoud van de project folder

- **Pages folder**

Deze folder bevat de razor pagina's en de ondersteunende pagina's. Elke razor pagina bestaat uit 2 delen:

- `.cshtml` bestand met de HTML opmaak en de C# code volgens de razor syntax
- `.cshtml.cs` is het code-behind bestand die de C# code bevat om de pagina gebeurtenissen (events) te behandelen.

De ondersteunende bestanden beginnen met een underscore. Een voorbeeld hiervan in ons project is de `_Layout.cshtml`, deze pagina configureert al de UI elementen op de zelfde manier voor alle pagina's die er gebruik van maken.

- **wwwroot folder**

Deze folder bevat statische html bestanden, javascript en css bestanden.

- **appsettings.json**

Dit bestand bevat configuratiedata. In onze projecten gaan we het bijvoorbeeld gebruiken om onze connection string op te slaan.

```
(appsettings.json)
{
  "ConnectionStrings": {
    "TestDbContext": "Server=(localdb)\\mssqllocaldb; Database=TestDbContext;
    Trusted_Connection=True; MultipleActiveResultSets=true"
  }
}
```

- **Program.cs**

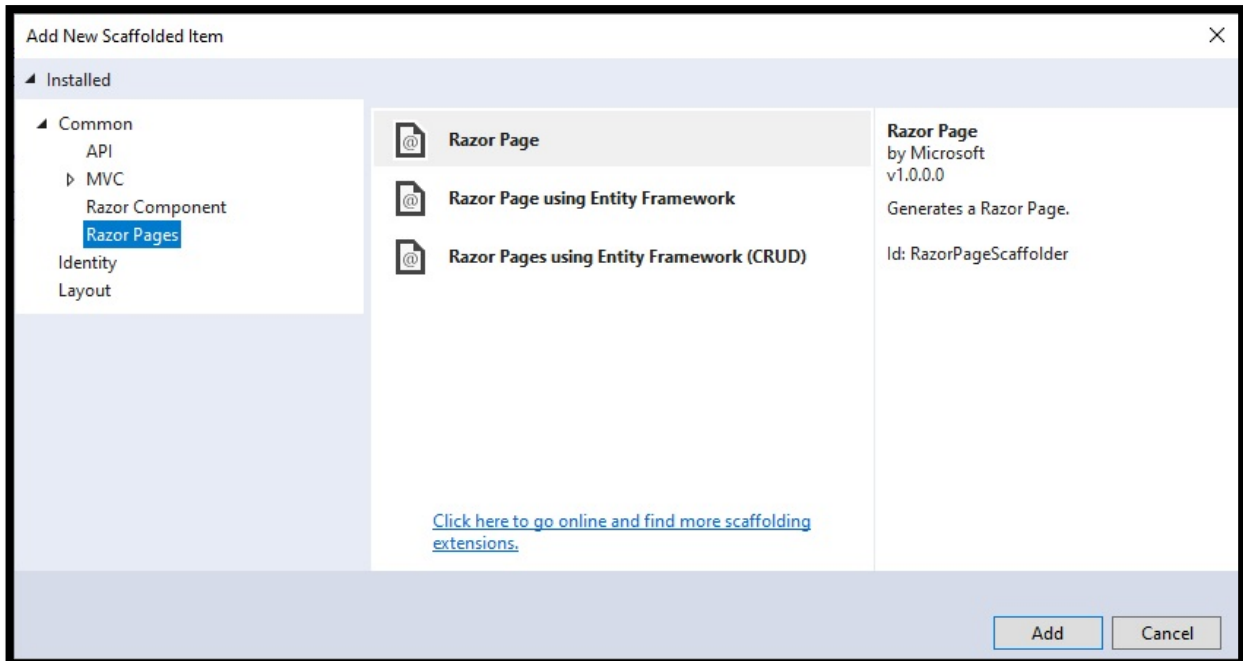
Dit is het entrypoint of startpunt van ons project.

Het `Program` bestand vervangt het startup bestand uit de vorige versies en bevat de configuratie van onze toepassing.

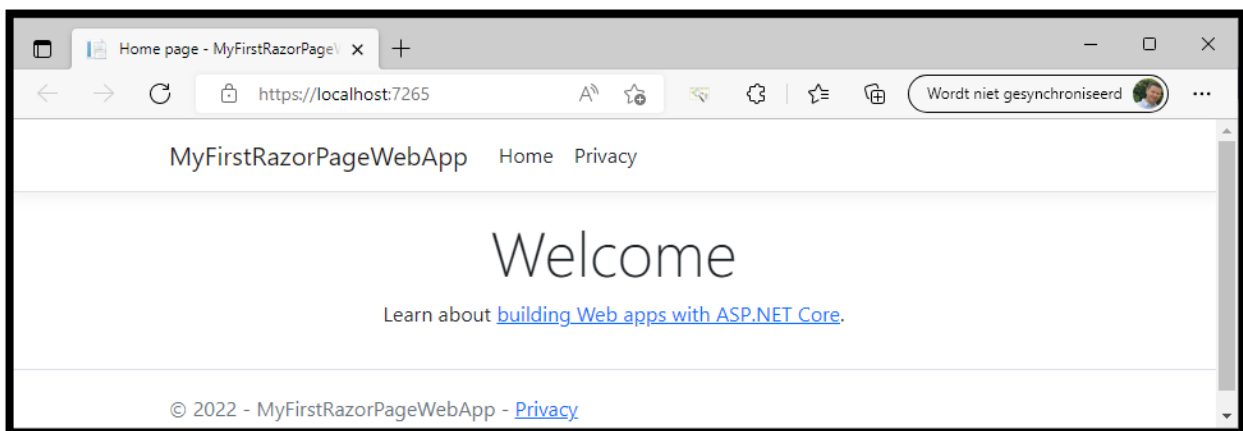
Het routing principe, gebruik van `razorPages`, static files, MVC, ...

2.3.2 Scaffolding

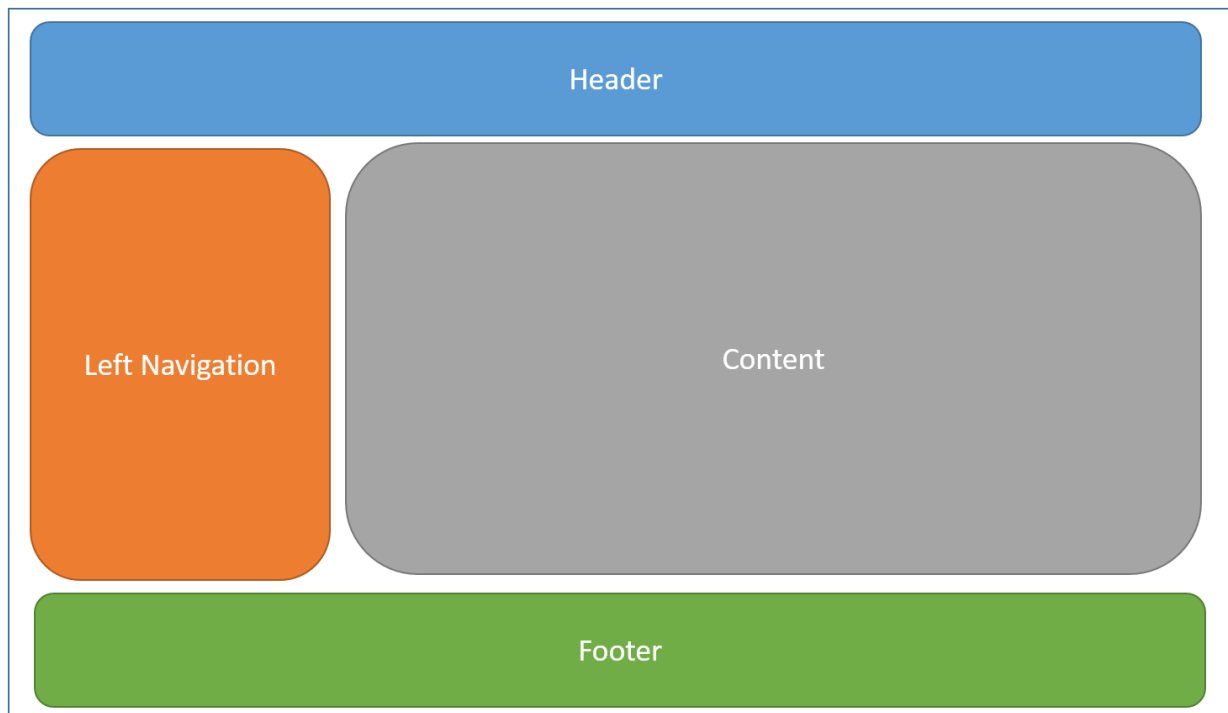
Project – Add – New scaffolded item (zie fig13)



(fig13)



2.3.3 Layout in ASP.Net Core



- Pages/Shared/_Layout.cshtml
 - Head
 - Css: bootstrap
 - Body
 - Header: Header - Navigation
 - Left navigation: /
 - Content: @RenderBody()
 - Footer: Footer
 - Script section

2.3.4 Cross-Site Request Forgery (XSRF/CSRF) attacks

```
<form asp-action="Index" asp-controller="Home" method="post">  
  @Html.AntiForgeryToken()  
  
  <!-- ... -->  
</form>
```

Door de “AntiForgeryToken” wordt er extra info toegevoegd aan de formulier elementen zodat post requests enkel door het juiste formulier worden behandeld.

[HttpGet] – [HTTPPost]

3. Create a Razor Pages web app

- Name: RazorPagesFilm
- Model: Film
 - FilmId
 - Titel
 - Genre
 - Datum
- Model: Acteur
 - ActeurId
 - Naam
 - Voornaam
- Hyperlinks
 - Home
 - Film
 - Acteur
- Razor Pages
 - Index.cshtml
 - Film.cshtml
 - Acteur.cshtml

4. Model View Controller (MVC)

In een model-view-controller architectuur wordt de toepassing onderverdeeld in drie groepen; de modellen, views en controllers.

We gaan ze hier afzonderlijk bespreken. Als we gebruik maken van de MVC architectuur in onze web applicatie worden onze aanvragen (requests) doorgestuurd naar de controller die op zijn beurt gebruik maakt van de Views met eventueel een model parameter (routing). De acties in de controller gaan acties of zoekopdrachten uitvoeren en via de views tonen op het scherm.

4.1 Model

Het model in een MVC toepassing bevat de business logica en de processen in de operaties. Deze logica slaan we op in een class en vertegenwoordigt meestal de verwijzing naar de databank entiteiten.

Via dit model kunnen we dan de juiste gegevens doorgeven via de views aan de gebruiker.

4.1.1 Model binding

ASP.NET Core MVC model binding gaat de aanvragen(form values, route data, query string parameters, HTTP headers) in onze toepassing omzetten naar objecten waar onze controllers verder mee kunnen werken, meestal een model klasse.

Hierdoor heeft de controller de juiste klasse of andere logica om deze als parameter door te geven aan een view of andere actie.

4.1.2 Model validation

In ons model kunnen we methodes gebruiken van de data annotations uit de C# namespace.

```
using System.ComponentModel.DataAnnotations;
public class LoginViewModel
{
    [Required]
    [EmailAddress]
    public string Email { get; set; }

    [Required]
    [DataType(DataType.Password)]
    public string Password { get; set; }

    [Display(Name = "Remember me?")]
    public bool RememberMe { get; set; }
}

public async Task<IActionResult> Login(LoginViewModel model,
string returnUrl = null)
{
    if (ModelState.IsValid)
    {
        // work with the model
    }
    // At this point, something failed, redisplay form
```

- ```
 return View(model);
 }

```
- [Required]
  - [EmailAddress]
  - [DataType(DataType.Password)]
  - [Display(Name = "Vul uw naam in?")]

## 4.2 View

De views zijn verantwoordelijk voor de presentatie van de inhoud via onze user interface.

De razor view engine wordt gebruikt om de C# code te implementeren in de HTML opmaak. Er moet zo weinig mogelijk logica geprogrammeerd worden in de views enkel wat relateert op de inhoud zelf.

Indien toch het geval zou zijn in een complex geval kan je misschien best overschakelen op een View component, view model of view template om de view toch overzichtelijk te houden.

### 4.2.1 View Content

Razor View : Test01.cshtml

In deze view gaan we dynamisch 5 producten genereren in de html pagina.

```

 @for (int i = 0; i < 5; i++) {
 Product @i
 }

```

Razor View: Test02.cshtml

In deze view gaan we de producten tonen die we via de parameter in de controller doorgeven aan de View.

```
@model IEnumerable<Product>

 @foreach (Product p in Model)
 {
 @p.Name
 }

```

### 4.2.1 Partial view

In onze oefening van de sportstore hebben we een partial view aangemaakt om een productfiche te tonen. Het voordeel van een partial view is het gemak om deze view te implementeren in verschillende pagina's.

## 4.3 Controller

Controllers zijn de componenten die de acties orchestreren tussen de modellen en de views om zo de inhoud op het scherm te tonen.

In een MVC toepassing gaat de view de inhoud tonen en de controller behandelt de invoer en de verwerking van de gebruiker (GET en POST).

De controller is het startpunt in een MVC toepassing en zal zelf selecteren welke view met welk model wordt aangemaakt.

## 4.4 Routing

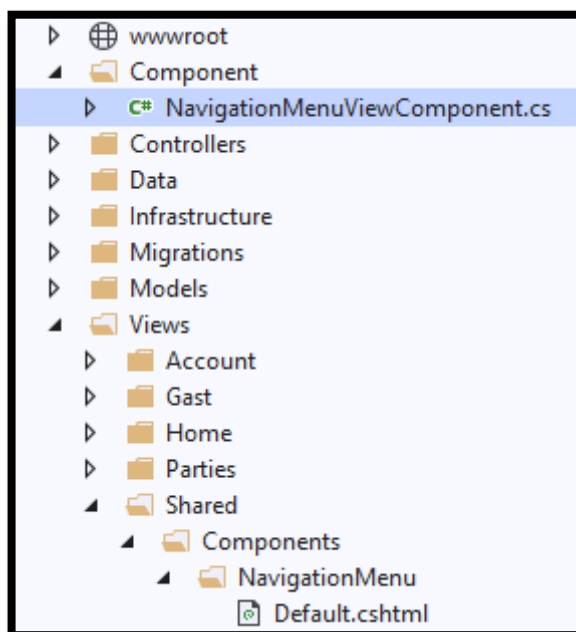
ASP.Net core routing maakt gebruik van het asp.net routing principe. Dit krachtige url routing mechanisme laat ons toe om heel gemakkelijk de juiste acties te koppelen in onze controllers via deze url's.

De routing wordt default ingesteld in de Configure methode in de Startup klasse.

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

## 4.5 Navigation View component

Onze componenten voegen we toe onder de Folder "Components" in de project root folder. Ook voegen we een verwijzingsfolder toe in de Shared folder onder de Views met dezelfde naam als onze nieuwe component. In dit voorbeeld hebben we een NavigationMenu component toegevoegd.



Onze nieuwe klasse overerft van de basisklasse ViewComponent en ziet er standaard als volgt uit:

```
public class NavigationMenuViewComponent : ViewComponent
{
 public IViewComponentResult Invoke()
 {
 return View();
 }
}
```

Door deze instelling wordt bij het oproepen van onze component de default view opgeroepen in de shared folder zoals hierboven weergegeven.

Het gebruik van deze componenten hebben we toegepast in de oefeningen van de sportstore en Feestje.

## 4.6 Tag Helper

### Enable Tag helper

Tag helpers zijn niet automatisch aanwezig en moeten toegevoegd worden aan het project door het toevoegen van een directive. Meestal wordt deze directive in het \_ViewImports.cshtml bestand geplaatst.

```
(_ViewImports.cshtml)
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

### 4.6.1 Anchor tag helper

- **asp-controller**      Naam van de controller.
- **asp-action**          Naam van de actie methode.
- **asp-area**             Naam van de area (Identity).
- **asp-page**             Naam van de Razor page.

### 4.6.2 Form tag helper

Met de form tag helper gaan we de HTML <FORM> genereren. Verder kunnen we attributen toevoegen voor een MVC controller actie op te roepen of een custom route.

We kunnen ook een "hidden Request Verification Token" toevoegen om zo "cross-site request forgery" te voorkomen.

Daar moeten we wel het attribuut "[ValidateAntiForgeryToken]" toevoegen in de HTTP Post action methode.

Via deze attack probeert een hacker via een geautoriseerde gebruiker toegang te krijgen tot het systeem meestal zonder dat de gebruiker dit weet.

Hij kan zo via de sleutels in cookies en instellingen gebruik maken van deze waarden om zelf toegang te krijgen tot de server.

### Voorbeeld html tag helper

```
<form asp-controller="Demo" asp-action="Register" method="post">
 <!-- Input and Submit elements -->
</form>
```

### De html die wordt aangemaakt in bovenstaande form tag helper

```
<form method="post" action="/Demo/Register">
 <!-- Input and Submit elements -->
 <input name="__RequestVerificationToken" type="hidden"
 value="<removed for brevity>">
</form>
```

### Voorbeeld van een custom route (Controller + View -> form)

#### Controller

```
public class HomeController : Controller
{
 [Route("/Home/Test", Name = "Custom")]
 public string Test()
 {
 return "This is the test page";
 }
}
```

#### View

```
<form method="post">
 <button asp-route="Custom">Click Me</button>
 <input type="image" src="..." alt="Or Click Me" asp-route="Custom">
</form>
```

### Veel gebruikte AnchorTagHelper attributen in onze form action.

| Attribuut                 | Beschrijving                                                    |
|---------------------------|-----------------------------------------------------------------|
| <b>asp-controller</b>     | Naam van de controller.                                         |
| <b>asp-action</b>         | Naam van de actie methode.                                      |
| <b>asp-area</b>           | Naam van de area (Identity).                                    |
| <b>asp-page</b>           | Naam van de Razor page.                                         |
| <b>asp-page-handler</b>   | Naam van de Razor page handler.                                 |
| <b>asp-route</b>          | Naam van de route.                                              |
| <b>asp-route-{value}</b>  | Een unieke URL route waarde. Bijvoorbeeld: asp-route-id="1234". |
| <b>asp-all-route-data</b> | Alle route waardes                                              |
| <b>asp-fragment</b>       | Het URL fragment.                                               |

Er is ook een alternatief bij de HTML helper: `Html.BeginForm` en `Html.BeginRouteForm`

### 4.6.3 Custom tag helper

Met een custom tag helper kunnen we een stuk code opslaan in een taghelper zodat we deze kunnen hergebruiken in een andere view.

We maken een folder TagHelpers aan binnen onze project folder.

#### Voorbeelden

- Pagination (zie oefening Sportstore)
- Custom TagHelper (zie oefening Navigation component)
- De custom taghelper om de rollen te tonen in de Voertuig MVC toepassing

## 4.7 MVC – Oefening Party Invites

- Nieuw MVC project
- Model toevoegen (Models/Gast)
- Actie toevoegen in onze startpagina (Views/Home/Index.cshtml)
  - Anchor tag helper
  - `<a asp-action="ReservatieForm">Reserveer nu</a>`
- Actie in HomeController toevoegen

```
[HttpGet]
public IActionResult ReservatieForm()
```

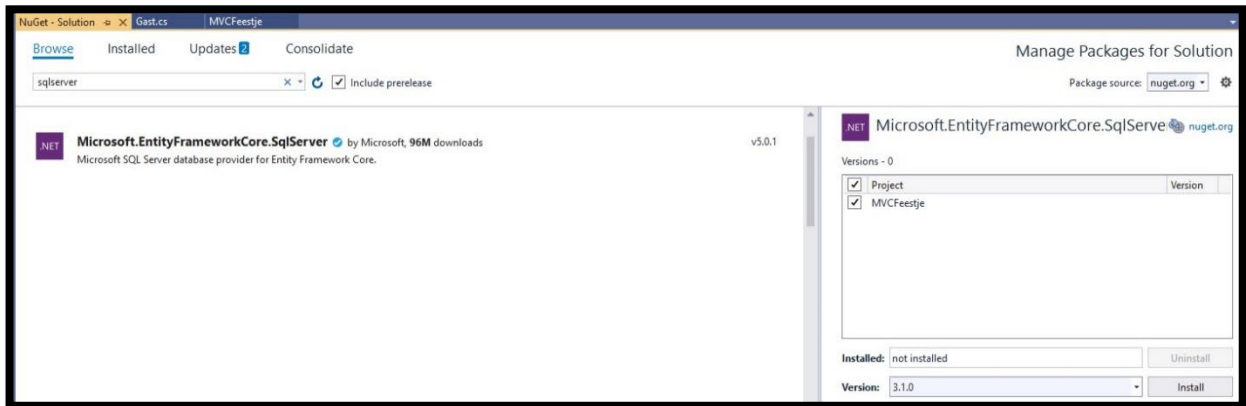
- Data folder en statische klasse toevoegen (Data/SeedData.cs)
- Volledige oefening: (C# Web MVC Party invites.pdf)

## 4.8 MVC – Oefening Sportstore

- Custom tag helper (pagination)

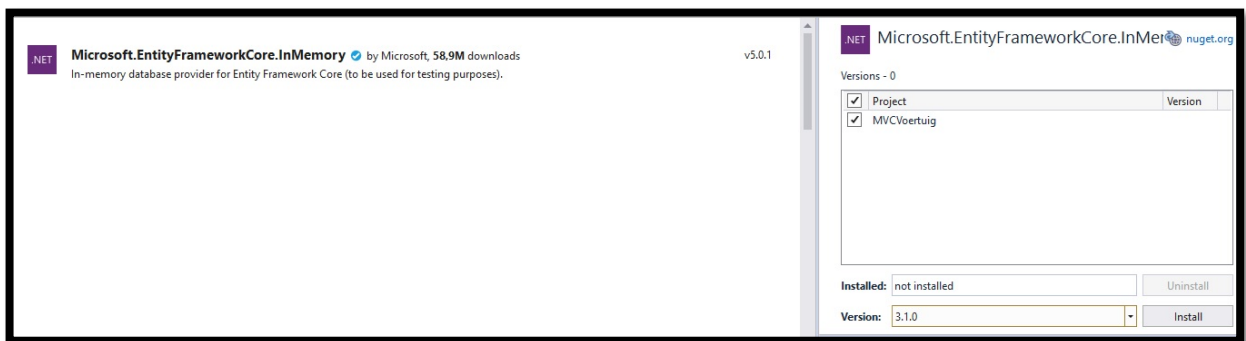
## 5. Entity Framework Core (EF Core)

### 5.1 Sql server



Met deze optie in het entity framework gaan we onze databank koppelen met onze MVC toepassing.

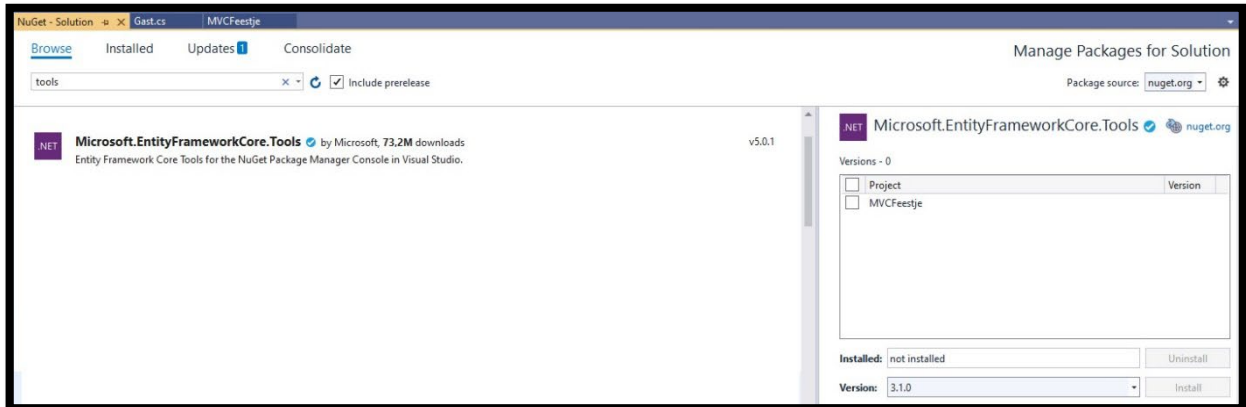
### 5.2 In memory



Het voordeel van de in memory functionaliteit in het entity framework is dat je al de databank commando's kan gebruiken maar dat er geen fysieke databank wordt aangemaakt. De databank wordt aangemaakt in het geheugen van de server van de toepassing.

## 5.3 Reverse engineering

We kunnen ook starten van een bestaand data model en zo ons model opbouwen via de databank in Visual Studio. Zorg er wel voor dat je de tools hebt toegevoegd via de package manager.



- Nuget package manager console:

Scaffold-DbContext

```
Server=(localdb)\MSSQLLocalDB;Database=TestDB;Trusted_Connection=True;
```

```
Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
```

- Startup.cs  
AddDbContext
- Oefening country – cityName

## 5.4 Data verwerking met Linq

Met Linq kunnen we de collecties van onze databank gemakkelijk filteren of samenvoegen om zo een nieuwe collectie aan te maken.

Veel gebruikte methodes van Linq:

- Where: filteren van gegevens met de where conditie
  - `IEnumerable<Object> newList = List(Object).Where(x => x.Property = value)`
  - Like '%'
    - `.Where(x => x.Property.Contains("searchvalue"))`
- OrderBy: collectie sorteren
  - `.OrderBy(x=>x.id)`
- Skip: Skip gaat het aantal elementen (parameter) overslaan in je collectie .
  - `.Skip(1)`
- Take: Take gaat het aantal elementen (parameter) selecteren in je collectie.
  - `.Take(1)`
- Join: collecties samenvoegen
  - `var newList = TableX.Join(TableY, x => x.id, y => y.id, (x,y) => new{ Xprop = x.Prop, Yprop = y.Prop});`
- Include

Met de Linq Join functie kunnen we dus heel gemakkelijk een nieuw object aanmaken met als basis de collecties uit onze toepassing.



## 6. Dependency Injection (DI)

### 6.1 Service lifetime

- Singleton
- Scoped
- Transcient

### 6.2 DbContext

De DbContext klasse is de klasse die verbinding legt met de fysieke databank in SQL server

### 6.3 Repository

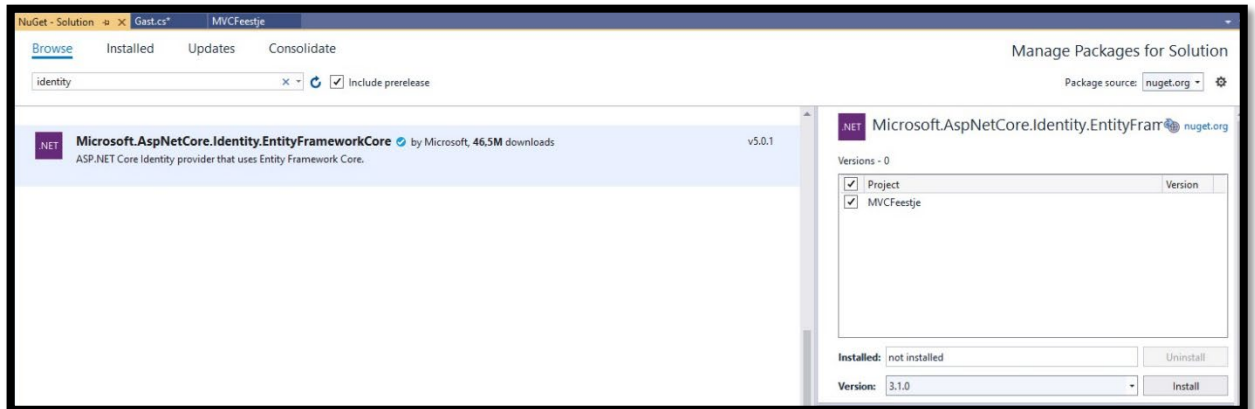
In verschillende oefeningen hebben we gebruik gemaakt van een Repository interface om de toepassing te koppelen aan onze databank context.

### 6.4 ASP.Net Core Identity

Met de Identity API kunnen we gebruikers beheren en onze web toepassingen beveiligen door middel van een uitbreiding aan onze bestaande user interface (UI). We kunnen gebruik maken van SQL Server, de API zal verschillende tabellen toevoegen gekoppeld aan het Identity framework.

## 6.5 Identity toevoegen aan een MVC project

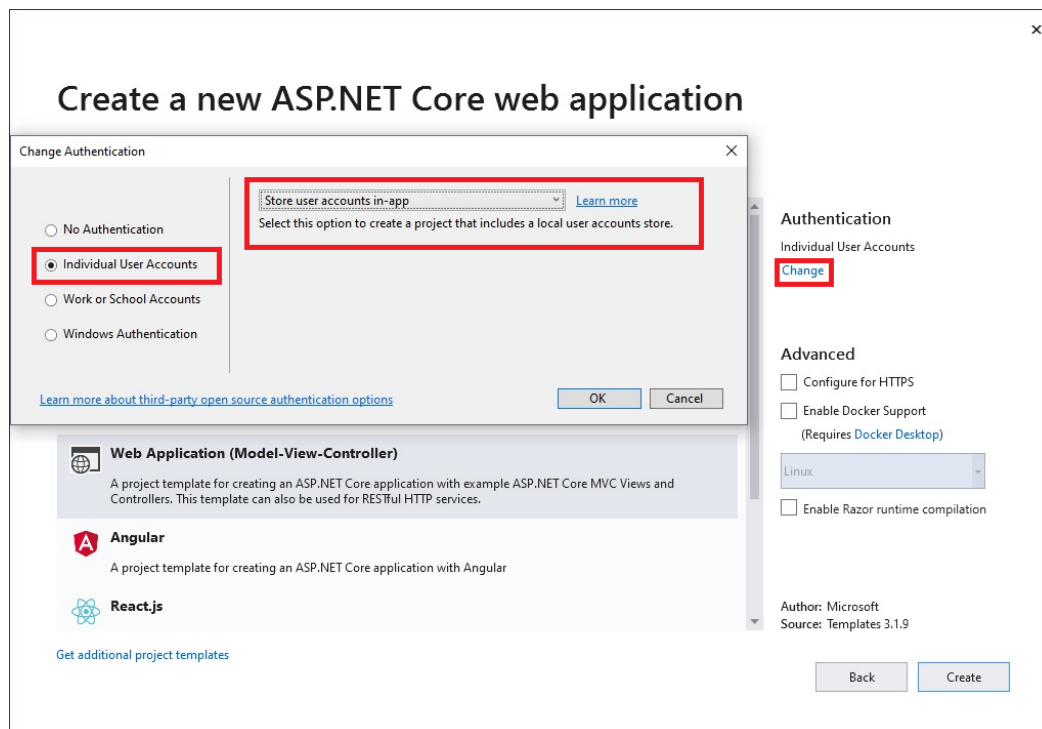
- Nuget packages

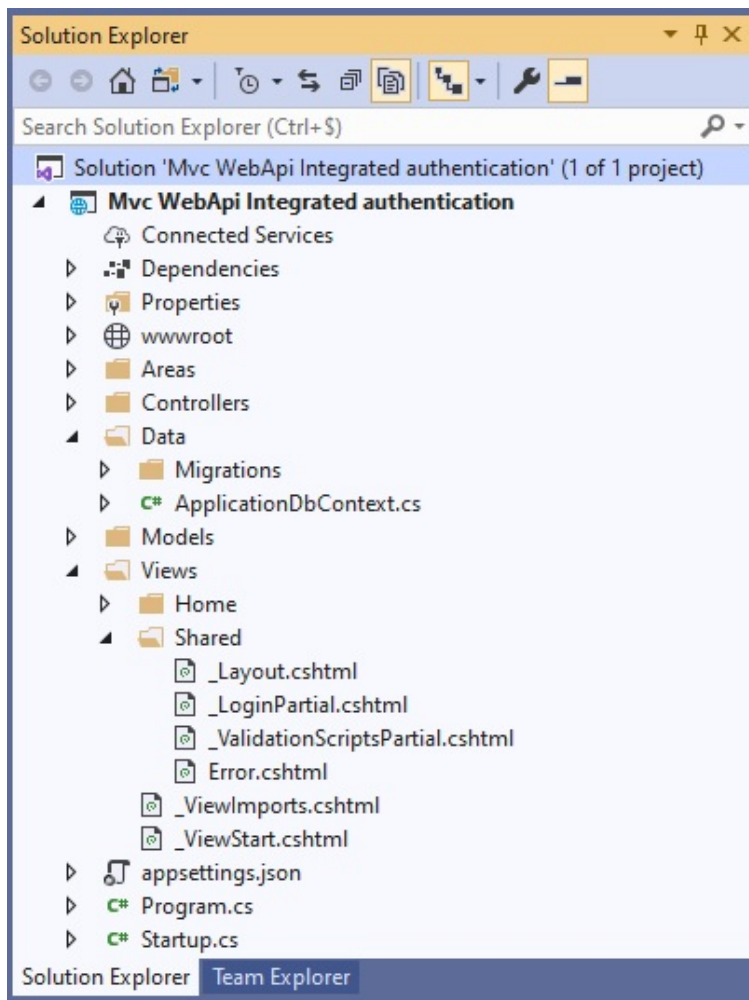


- AddDbContext  
`public class ApplicationDbContext : IdentityDbContext<IdentityUser>`

## 6.6 Web Application Identity - Integrated authentication

Bij een project met integrated authentication zijn automatisch de pagina's gelinkt aan het Identity framework geladen en kunnen we ze dus ook via onze browser oproepen. Om deze pagina's aan te passen aan specifieke project layout kan je deze pagina's scaffolden in jouw project.





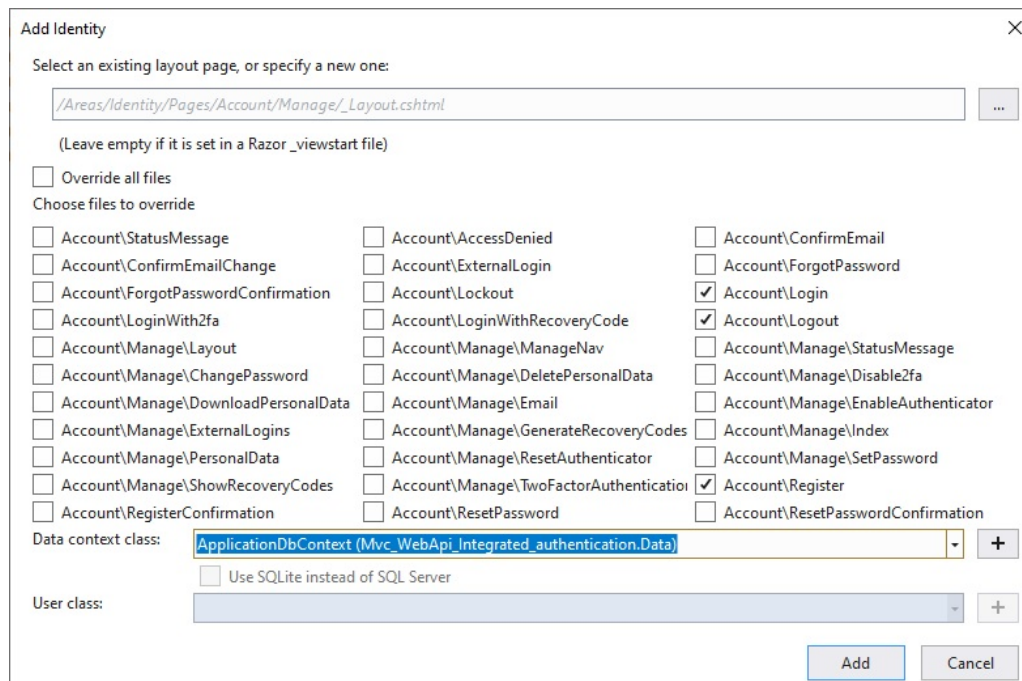
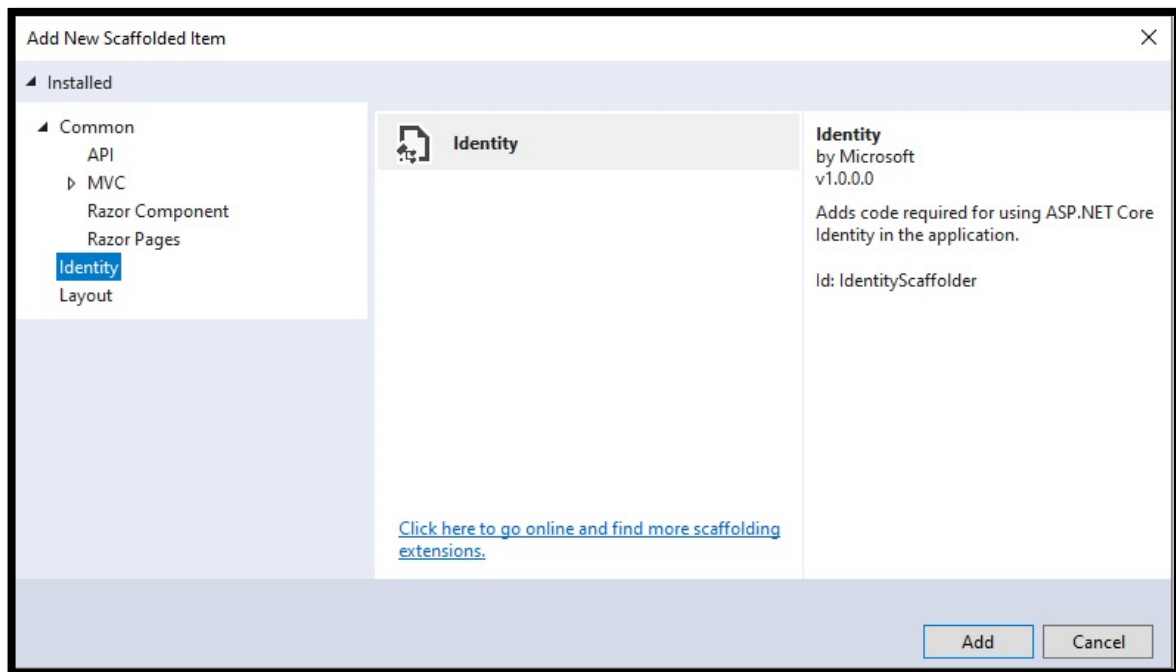
### Package manager console

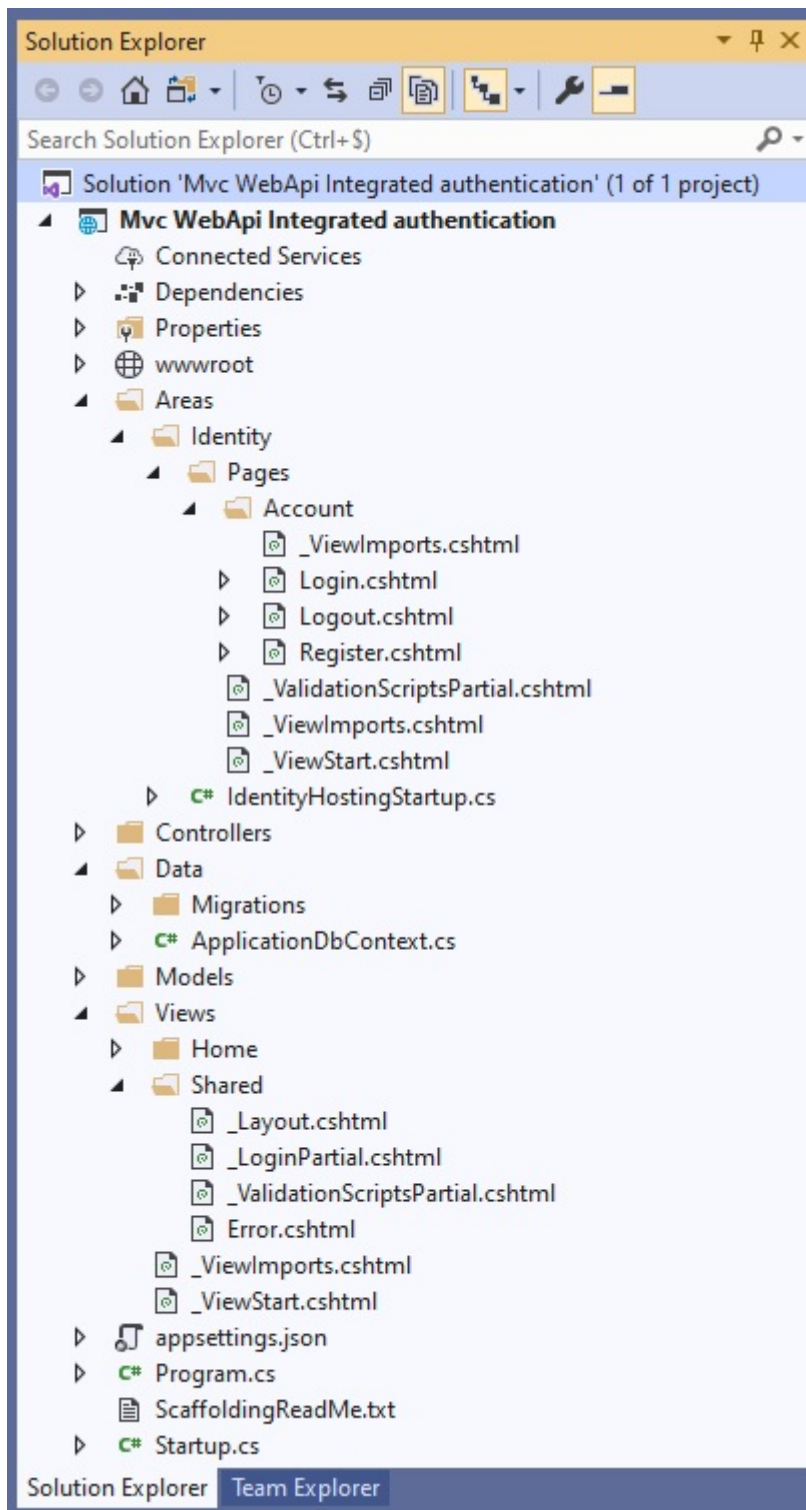
[Add-Migration](#) CreateIdentitySchema  
[Update-Database](#)

Het ASP.Net core identity project heeft een standaard Identity Area toegevoegd die enkel te bewerken valt na scaffolding.

De register en login pagina zijn dus nu al aan te spreken vanuit de web applicatie. Er is al een pagina toegevoegd in de \_layout.cshtml pagina namelijk de \_LoginPartial.cshtml.

De razor identity syntax is geïnstalleerd maar we moeten de pagina's scaffolden om ze te kunnen aanpassen.





Na het scaffold proces zijn de pagina's toegevoegd in de Areas/Identity/Pages/Account folder.

Nu kunnen we de pagina's aanpassen in ons project.

## 6.7 Async programmeren

In onze toepassingen hebben we vooral gebruikt gemaakt van de volgende klassen.

- UserManager
  - IdentityResult = UserManager.CreateAsync(IdentityUser, string pwd);
  - IdentityUser = UserManager.FindByEmailAsync(string email)
  - Bool = UserManager.IsInRoleAsync(IdentityUser, string role)
  - IdentityResult = UserManager.AddToRoleAsync(IdentityUser, string role)
  - List<string> = UserManager.GetRolesAsync(IdentityUser)
  - IdentityResult = UserManager.RemoveFromRoleAsync(IdentityUser, string role)
  - IdentityResult = UserManager.DeleteAsync(IdentityUser);
- SignInManager
  - SignInResult = SignInManager.PasswordSignInAsync(string username, string pwd, false, false)
- RoleManager
  - IdentityResult = RoleManager.CreateAsync(IdentityRole);
  - Bool = RoleManager.RoleExistsAsync(string role)
  - IdentityRole = RoleManager.FindByNameAsync(string role)

In deze klassen staan al heel wat methodes geprogrammeerd. De meest gebruikte methodes gaan we hier opsommen. Ze zijn allemaal async voorgeprogrammeerd en daarom moeten wij ook onze methodes async maken.

## 7. Authorisation

Met het keyword [Authorize] kunnen we een Controller of action beveiligen en controleren dat de user correct is ingelogd en bepaalde acties mag uitvoeren. Automatisch wordt een gebruiker doorgestuurd naar de Account/Login pagina wanneer de gebruiker niet is ingelogd.

### 7.1 Role management

Met het authorize keyword kunnen we aangeven dat er ingelogd moet worden om en bepaalde controller te kunnen gebruiken. We kunnen dit proces nog uitbreiden met het gebruik van rollen.

```
[Authorize(Roles = "Admin")]
```

Met bovenstaande instructie kunnen we voor een controller of een methode een bepaalde rol-authenticatie toepassen.

Er bestaan al methodes voor het Rollenbeheer te kunnen toepassen. Deze zijn al toegelicht bij de sectie async programmeren.

## 8. Middleware componenten

In onze projecten maken we veel gebruik van middleware componenten. Zonder deze middleware kunnen we geen gebruik maken van de onderliggende functionaliteiten.

### 8.1 Startup.cs

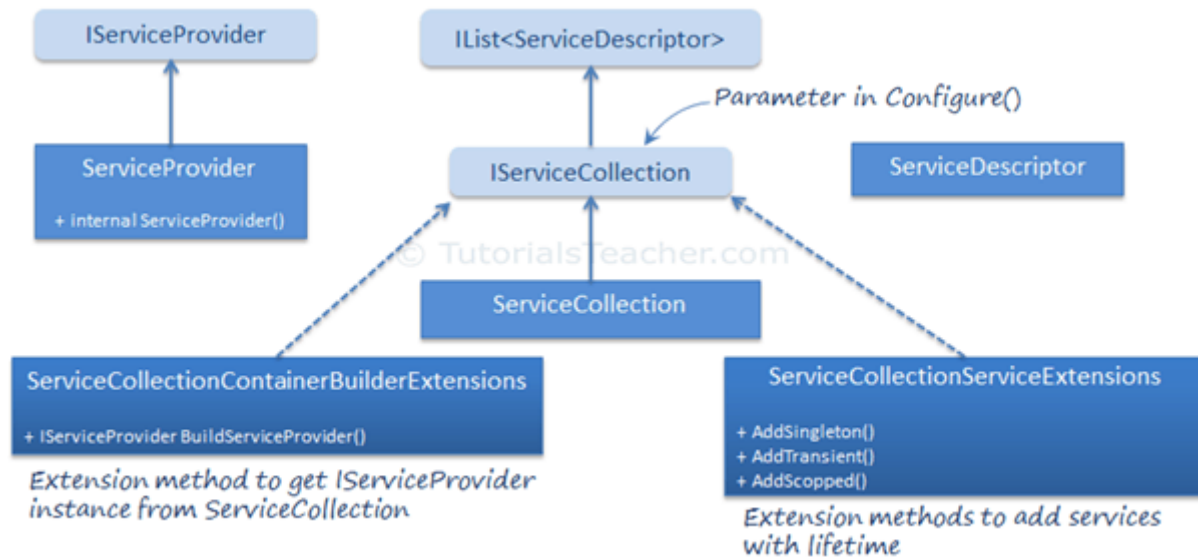
```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
```

In de Configure methode van de Startup klasse kunnen we Middleware componenten toevoegen die nuttig kunnen zijn voor onze webapplicatie.

### 8.2 Middleware voorbeelden

- `app.UseAuthentication()`
  - Voegt de authenticatie middleware toe aan je project
- `app.UseAuthorization()`
  - Voegt de autorisatie middleware toe aan je project
- `app.UseEndpoints()`
  - Voegt de endpoint middleware toe aan je project

## 9. Built-in services



- IApplicationBuilder

In de Startup.cs klasse kan je de volgende methode Configure vinden met de instellingen van de built-in services.

```

public void Configure(IServiceProvider pro, IApplicationBuilder app,
 IHostingEnvironment env)
{
 var services = app.ApplicationServices;
 var logger = services.GetService<ILog>() }

 //other code removed for clarity
}

```

Veel gebruikte services:

- services.AddControllersWithViews
- services.AddDbContext
- services.AddIdentity
- HttpContext
 

```

var services = HttpContext.RequestServices;
var log = (ILog)services.GetService(typeof(ILog));

```
- IServiceCollection
 

```

public void ConfigureServices(IServiceCollection services)
{
 var serviceProvider = services.BuildServiceProvider();
}

```



## 10. Oefeningen

10.1 Party invites (Apress – Pro ASP.Net Core 6)

10.2 Razor pages - Movie

10.3 SportsStore (Apress – Pro ASP.Net Core 6)

10.4 Todo Item

10.5 Voertuig Identity

10.6 Voertuig no Identity

10.7 Feestje

10.8 Secretariaat