

CompanyEmployees Manager

Asp.Net Core MVC

1 Inleiding MVC – design patroon

Model-view-controller (of MVC) is een ontwerppatroon ("design pattern") dat het ontwerp van complexe toepassingen opdeelt in drie eenheden met verschillende verantwoordelijkheden: datamodel (model), datapresentatie (view) en applicatielogica (controller).

Het scheiden van deze verantwoordelijkheden bevordert de leesbaarheid en herbruikbaarheid van code. Het maakt ook dat bijvoorbeeld veranderingen in de gebruikersinterface niet direct invloed hebben op het datamodel en vice versa.

De programmeer-code wordt opgesplitst in **3 componenten** die **los aan elkaar gekoppeld** worden en elk een **aparte verantwoordelijkheid** hebben (ontwerp-principes **Loose Coupling** en **Separation of Concerns (Soc)**)

MVC stands for Model, View, and Controller. MVC separates an application into three components - Model, View, and Controller.

Model: een Model vertegenwoordigt de vorm van bepaalde gegevens. Een klasse in C # wordt gebruikt om een model te beschrijven.

Modelobjecten slaan gegevens op die uit de database zijn opgehaald.

Model is de voorstelling en vorm van de data.

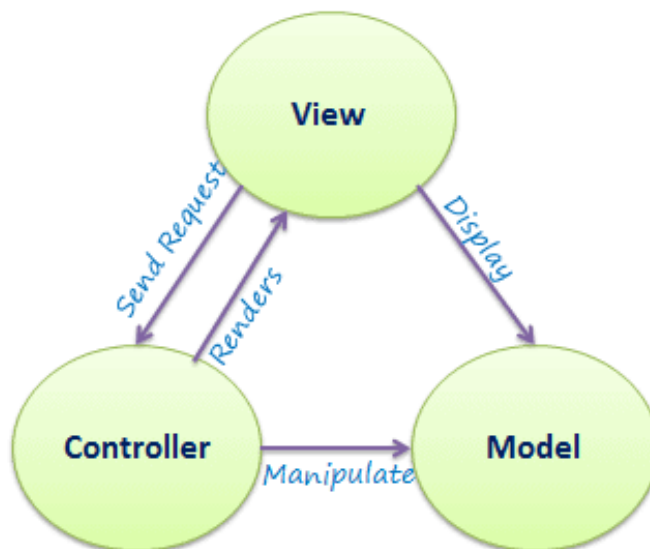
View: een View in MVC is een gebruikersinterface. Deze geeft aan de eindgebruiker de mogelijkheid om modelgegevens te bekijken en de wijzigen. In ASP.NET MVC worden views getoond in HTML, CSS en een speciale syntaxis (**Razor-syntaxis**) die het gemakkelijk maakt om te communiceren met het model en de controller.

View is de User Interface.

Controller: De controller behandelt de HTTP Requests. De controller verwerkt het HTTP Request en stuurt als HTTP Response de juiste weergave terug.

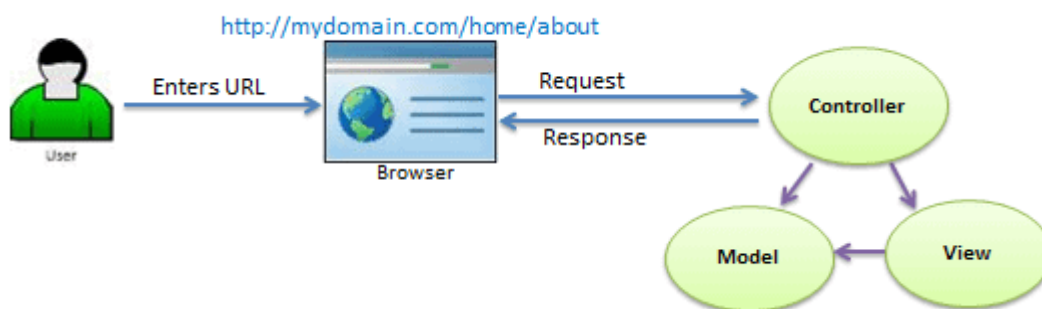
Controller is de request handler.

In de volgende figuur toont de interactie tussen de componenten Model, View, and Controller:



MVC Architectuur (<https://www.tutorialsteacher.com/mvc/mvc-architecture>)

De volgende figuur toont de HTTP user's request en Http Response via ASP.NET Core MVC:



MVC Architectuur Request/Response
(bron: <https://www.tutorialsteacher.com/mvc/mvc-architecture>)

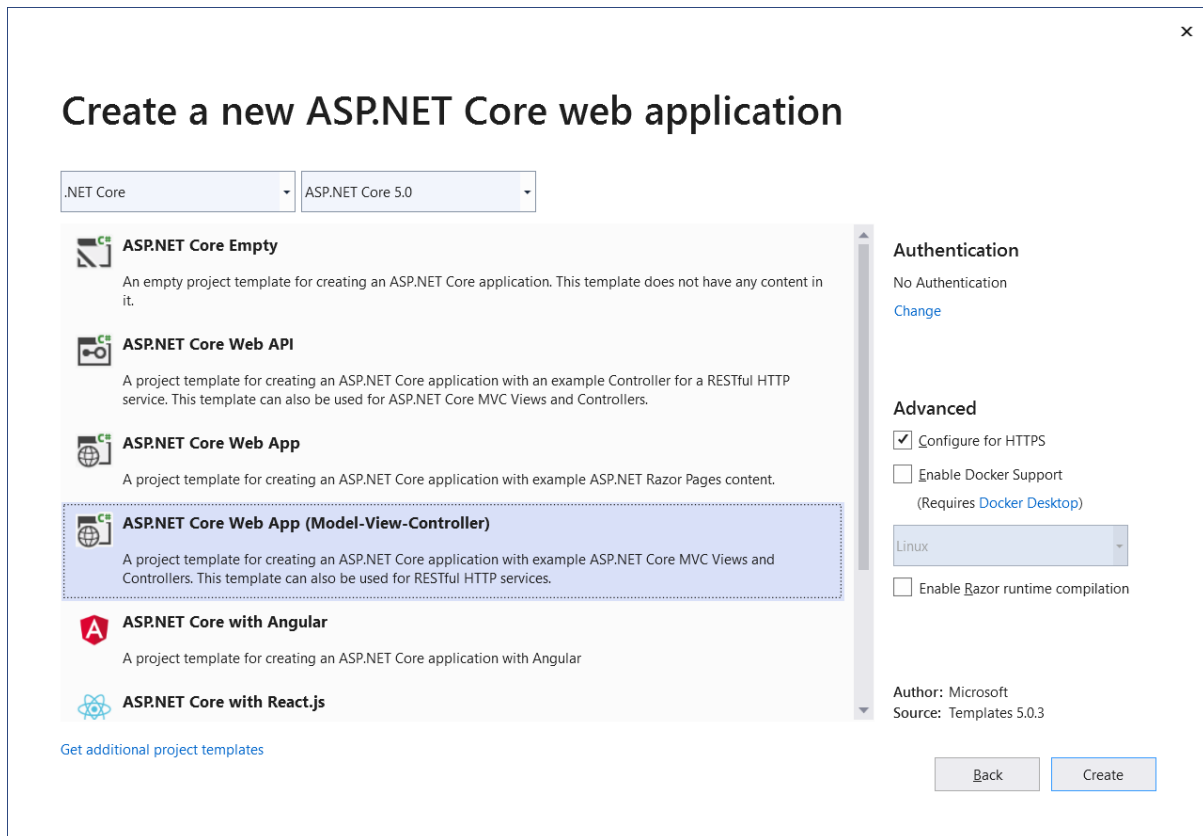
Wanneer een eindgebruiker(User) navigeert naar een URL in the browser, wordt deze doorgestuurd (routed) van de webserver and naar een controller. Een controller voert de nodige acties uit, haal de nodige data op via de models en stuurt de juiste view terug als HTTP response. Deze wordt dan teruggestuurd naar de browser van de eindgebruiker.

2 Creatie van een ASP.NET Core MVC Web Applicatie

We beginnen met het aanmaken van een nieuwe ASP.NET Core MVC Web applicatie in de bestaande solution CompaniesEmployees (<https://github.com/CSharpSyntraWest/CompaniesEmployees5>).

De naam van de applicatie is **ComanpanyEmployees.MVC**

We vertrekken vanaf een **Model-View-Controller project template**.



3 Project referenties leggen naar Entities, Contracts en Repository projecten

De applicatie zal CRUD operaties op een database verrichten d.m.v. Entity Framework Core

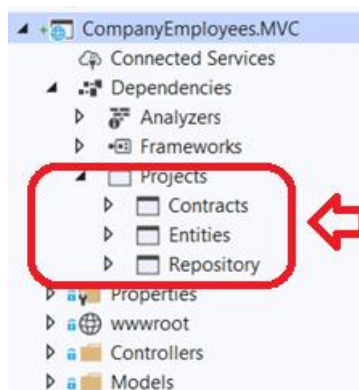
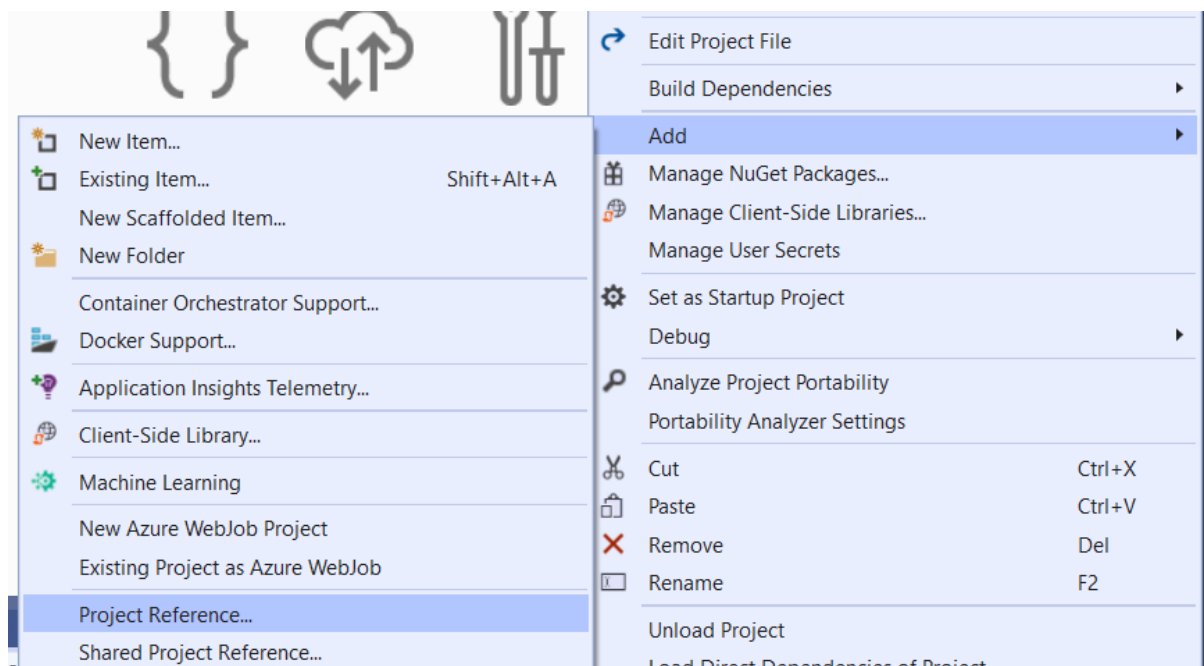
Om te kunnen werken met EF Core, moeten we eerst een EF Core model aanmaken. Dit doen we in dit onderdeel.

Een EF Core model is een verzameling van entity classes en een custom DbContext class (afgeleide class van DbContext). De Entity Classes zijn een representatie van de business objecten van de applicatie. De DbContext class representeert een sessie met de onderliggende database. Het voorziet

verschillende functionaliteiten, zoals bv connectie-management, change tracking, mapping, ondersteuning voor database operaties,...

DbContext houdt één of meer DbSet objecten bij. Een DbSet is een collectie van entiteiten.

We hebben reeds Models en een afgeleide class van DbContext (in project Entities) voor onze Web API. Eveneens hebben we Repository en Contracts projecten gemaakt. We gaan deze hergebruiken en leggen project references vanuit de CompanyEmployees.MVC applicatie naar deze 3 class library-projecten:



4 ConnectionString toevoegen aan Appsettings.json

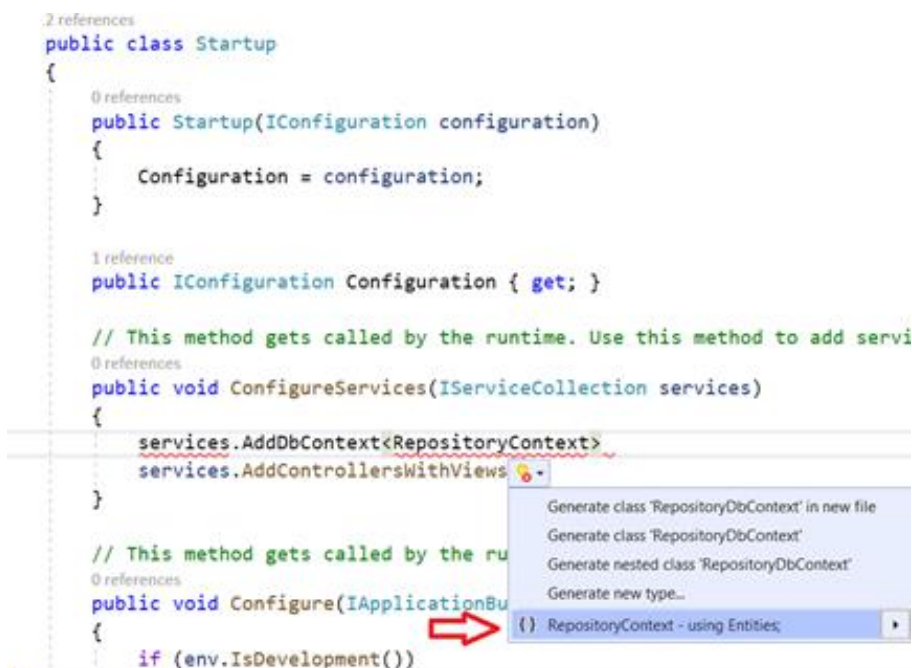
Open appsettings.json en voeg een **connectionstring** toe (dit mag dezelfde zijn als deze uit de Web API project, of een nieuwe)



5 Configuraties toevoegen aan startup.cs

Open **Startup.cs** en registreer de **RepositoryContext** class als **DbContext** service in de **ConfigureServices** methode (hier worden alle services geregistreerd met de IoC built-in container van ASP.NET Core)

Voeg de nodige using directive toe (using Entities;)



Specifieer bij de opties van de service dat een SqlServer database wordt gebruikt en haal via Configuration de connectionstring uit appSettings.json

```

public class Startup
{
    0 references
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    2 references
    public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this method to add services to the container.
    0 references
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddDbContext<RepositoryContext>(opt => opt.UseSqlServer(Configuration.GetConnectionString("sqlConnection")));
        services.AddControllersWithViews();
    }

    // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
    0 references
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }
        else
        {
            app.UseExceptionHandler("/Home/Error");
            // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.
            app.UseHsts(30);
        }

        app.UseStaticFiles();
        app.UseRouting();

        app.UseAuthorization();

        app.Map("/");
        app.MapControllerRoute(
            name: "default",
            pattern: "{controller}/{action}/{id?}");
    }
}

```

using Microsoft.EntityFrameworkCore;

CS1061 'DbContextOptionsBuilder' does not contain a definition for 'UseSqlServer' and no accessible extension method 'UseSqlServer' accepting a first argument of type 'DbContextOptionsBuilder' could be...

Lines 3 to 4

using Microsoft.AspNetCore.HttpsPolicy;

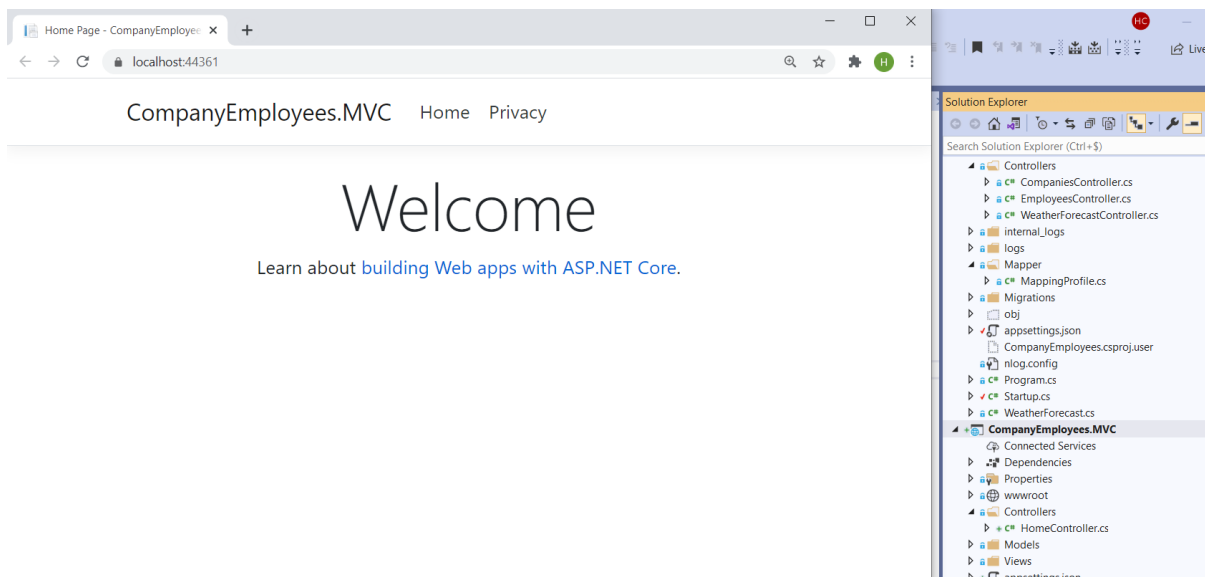
using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

Voeg Microsoft.EntityFrameworkCore als using directive toe (verwijzing naar de namespace)

6 Zet CompanyEmployee.MVC als startup project en run

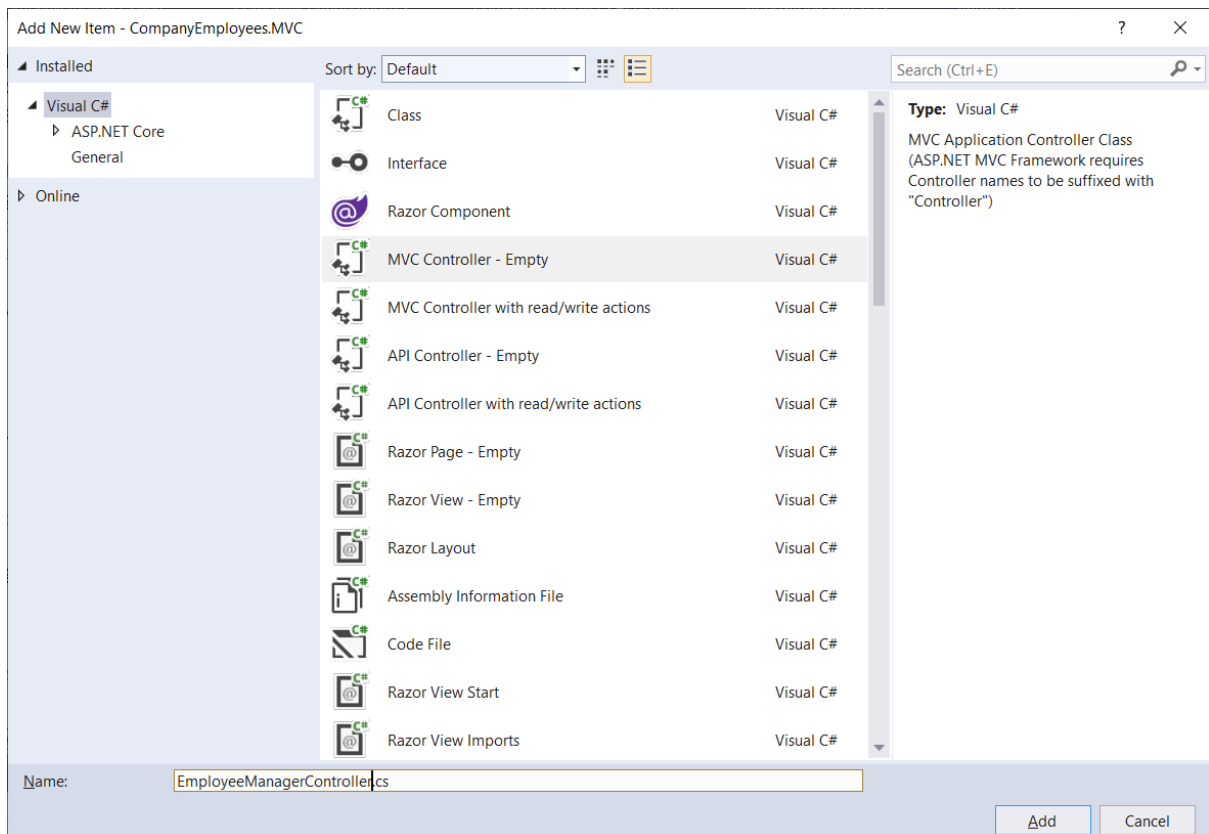
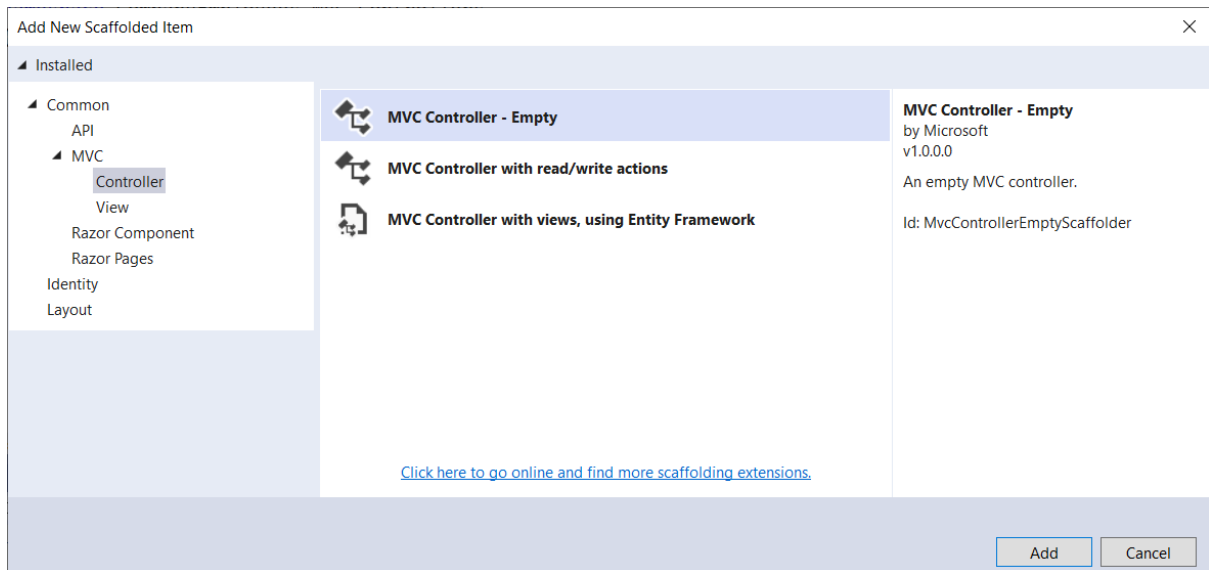
Zet CompanyEmployee.MVC als startup project **en run de web app**, je krijgt de volgende pagina in de browser:



7 Creatie van een EmployeeManager Controller

De CRUD functionaliteit van de applicatie zal voorzien worden in een Controller class: **EmployeeManagerController**.

Voeg een nieuwe Empty MVC Controller toe onder de folder Controllers met naam **EmployeeManagerController**.



EmployeeManagerController is **een afgeleide class van de Controller** class die in de namespace Microsoft.AspNetCore.Mvc is gedefinieerd.

De EmployeeManagerController zal CRUD operaties verrichten op de Employees tabel.

We gebruiken de Repository classen via de **RepositoryManager** die deze operaties reeds heeft geïmplementeerd.

In plaats van rechtstreeks een instantie aan te maken van RepositoryManager, gaan we dit doen via DI (Dependency Injection), voorzien in .NET Core.

Standaard voorziet ASP.NET Core construction injection voor alle services die geregistreerd zijn met de built-in IoC container (via methode ConfigureServices in startup.cs).

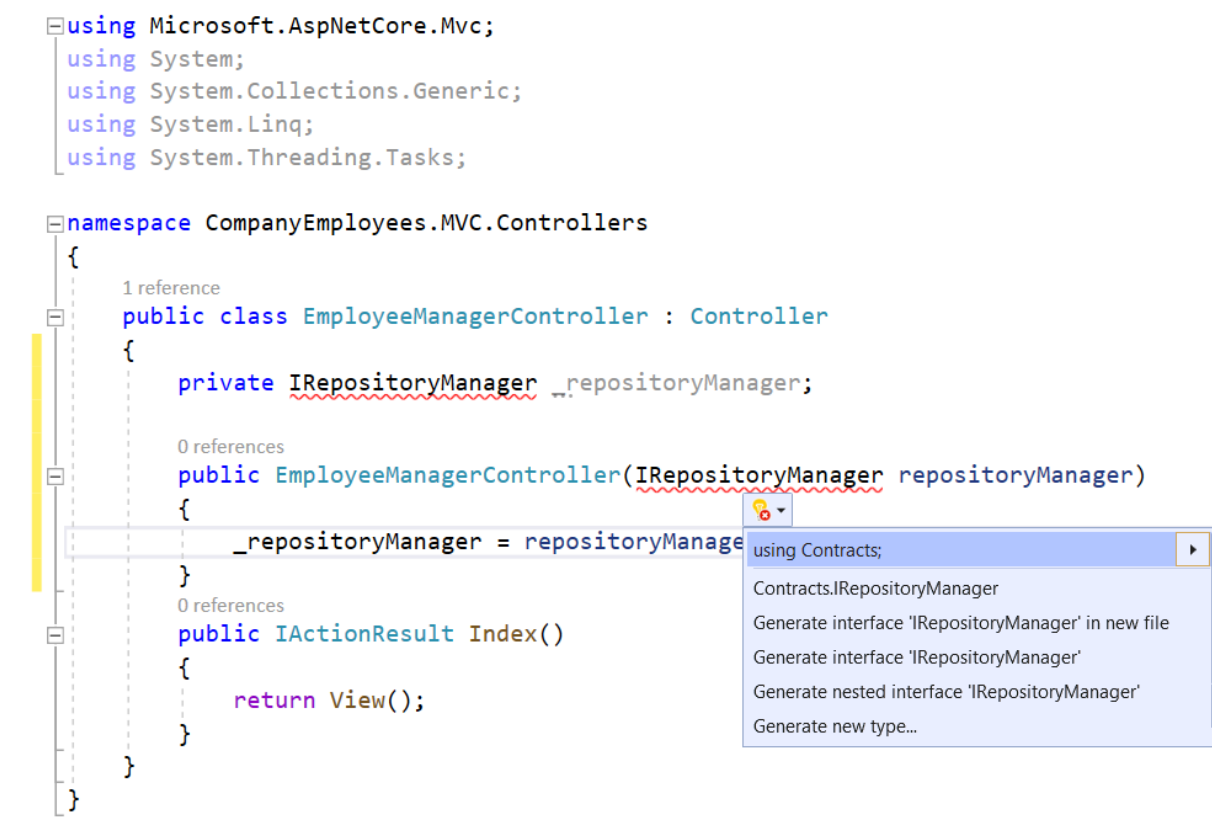
Bij dependency Injection via constructor injection speciëren we de interface IRepositoryManager als parameter in de constructor van onze Controller class:

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

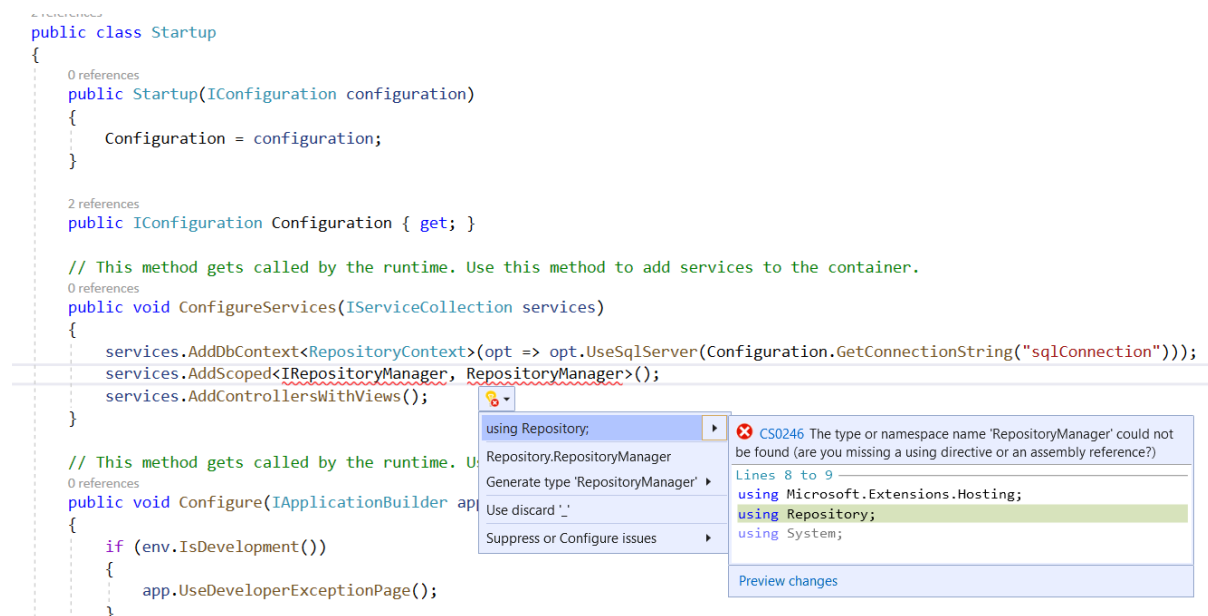
namespace CompanyEmployees.MVC.Controllers
{
    1 reference
    public class EmployeeManagerController : Controller
    {
        private IRepositoryManager _repositoryManager;

        0 references
        public EmployeeManagerController(IRepositoryManager repositoryManager)
        {
            _repositoryManager = repositoryManager;
        }

        0 references
        public IActionResult Index()
        {
            return View();
        }
    }
}
```

A screenshot of a code editor showing the implementation of the EmployeeManagerController class. The code includes using statements for Microsoft.AspNetCore.Mvc and various System namespaces. The class is defined within the CompanyEmployees.MVC.Controllers namespace. It inherits from Controller and has a private field _repositoryManager of type IRepositoryManager. The constructor takes an IRepositoryManager parameter and assigns it to _repositoryManager. The Index method returns a View. A dropdown menu is open next to the IRepositoryManager parameter in the constructor, showing options like 'using Contracts;', 'Contracts.IRepositoryManager', and 'Generate interface 'IRepositoryManager' in new file'.

We hebben de RepositoryManager nog niet geregistreerd als service met de IoC container van ASP.NET Core, dit moeten we nog doen in de methode **ConfigureServices** van startup.cs:



Voeg de nodige using directives toe (Repository en contracts)

AddScoped definieert de lifetime van de service. Dit wil zeggen dat er voor elke scope (de scope is hier HTTP request) een nieuwe instantie zal worden gecreëerd van de RepositoryManager class en deze instantie zal binnen deze scope worden gebruikt)

8 Een lijst tonen van Employees

Op de eerste pagina van onze web applicatie willen we een lijst van Employees tonen:

[←](#) [→](#) [↻](#) [🔒 localhost:44361/EmployeeManager](#)

CompanyEmployees.MVC [Home](#) [Privacy](#) [Employees](#)

List of Employees

[Insert](#)

Name	Position	Age	Actions	
Jana McLeaf	Software developer	30	Update	Delete
Kane Miller	Administrator	35	Update	Delete
Jos De Klos	Developer	45	Update	Delete

© 2021 - CompanyEmployees.MVC - [Privacy](#)

Om dit te kunnen tonen, hebben we een action method nodig in onze EmployeeManagerController class:

Voeg hiervoor een public Index() methode toe aan de EmployeeManagerController:

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

namespace CompanyEmployees.MVC.Controllers
{
    public class EmployeeManagerController : Controller
    {
        private IRepositoryManager _repositoryManager;

        public EmployeeManagerController(IRepositoryManager repositoryManager)
        {
            _repositoryManager = repositoryManager;
        }

        public IActionResult Index()
        {
            var employees = _repositoryManager.Employee.GetAllEmployees(false);
            return View(employees);
        }
    }
}
```

De Index() methode wordt een “action method” genoemd als het in een Controller class voorkomt. Deze geeft een object van type IActionResult terug. De code haalt een lijst van Employees op in de database door gebruik te maken van de RepositoryManager en EmployeeRepository classes.

De List wordt doorgegeven aan de view via de View() methode. De View() methode is gedefinieerd in de Controller base class en aanvaardt een model object dat kan doorgegeven worden naar een view.

De **View()** methode geeft een **ViewResult** object terug (dit is een class die de **IActionResult** interface implementeert).

We gaan nu een **Razor view** met naam **Index.cshtml** aanpassen die de lijst van employees kan tonen in een html tabel:

Alle views die bij een bepaalde controller horen worden standaard in een subfolder bijeengezet, die **dezelfde naam** heeft als de controller:

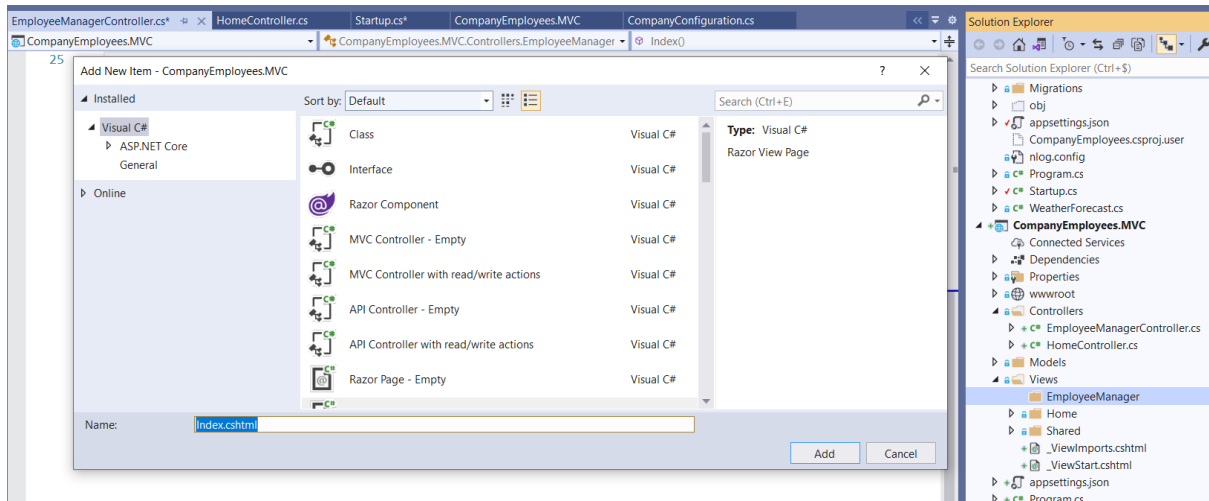
Maak dus een folder EmployeeManager aan onder de Views folder

In de folder Views/EmployeeManager, maak een nieuwe Razor View aan met

naam: **Index.cshtml** (dezelfde naam als de action methode in controller **EmployeeManager**)

Rechtsklik op de folder **Views/EmployeeManager** en kies **Add New Item** en dan **Razor View---Empty**.

Geef als naam **Index.cshtml**



Open de Razor View Index.cshtml en zet de volgende code (dit is code met Razor syntax):

```
@model IEnumerable<Employee>
<h2>List of Employees</h2>
<h2 class="message">@TempData["Message"]</h2>

<a asp-controller="employeeManager"
  asp-action="Insert"
  class="linkbutton">Insert</a>

<br />
<br />

<table border="1">
  <tr>
    <th>Name</th>
    <th>Position</th>
    <th>Age</th>
    <th colspan="2">Actions</th>
  </tr>
  @foreach (var item in Model)
  {
    <tr>
      <td>@item.Name</td>
      <td>@item.Position</td>
      <td>@item.Age</td>
      <td>
        <a asp-controller="employeeManager"
          asp-action="Update"
          asp-route-id="@item.Id" class="linkbutton">Update</a>
      </td>
    </tr>
  }
}
```

```

        </td>
        <td>
            <a asp-controller="employeeManager"
               asp-action="Delete"
               asp-route-id="@item.Id" class="linkbutton">Delete</a>
        </td>
    </tr>
}
</table>

```

Uitleg code:

De List.cshtml code begint met `@model` directive. Deze specificeert het type van het model dat wordt gebruikt in deze view.

De List view heeft een List van Employee objecten nodig om de lijst te kunnen weergeven in een html tabel.

In de lijn

```
<h2 class="message">@TempData["Message"]</h2>
```

Wordt de code `TempData["Message"]` naar de response stream geschreven. `TempData` is een **dictionary** object en bewaart data totdat het wordt uitgelezen. We gaan deze `TempData` later gebruiken om boodschappen te tonen wanneer employee's worden verwijderd en gaan dit later verder bespreken wanneer we de Delete functionaliteit toevoegen.

Verder wordt er een insert link getoond via de Anchor Tag helper.

Tag Helpers zorgen dat server-side code HTML elementen kunnen tonen in razor files.

Er zijn veel ingebouwde Tag Helpers, zoals Form, Select, en Input Tag Helpers. Een overzicht van alle Razor Tag Helpers kan je hier vinden:

<https://www.learnrazorpages.com/razor-pages/tag-helpers>

Merk op dat de Tag helper `asp-controller="EmployeeManager"`
en `asp-action="Insert"`

Deze Insert actie gaan we later implementeren in de `EmployeeManagerController` class

Een `<table>` tag afficheert de list van alle employees. De tabel zal enkel 3 properties tonen in deze tabel: Name, Age en Position

Om een lijst te kunnen genereren wordt een foreach loop gebruikt. Merk op dat we de public property Model gebruiken. Deze is het model object dat is doorgegeven aan de View door de controller. (dus IEnumerable <Employee>)

De 2 laatste kolommen op elke rij bevatten links die worden gegenereerd via de Anchor Tag Helper

De attributen van deze Tag Helper **asp-controller**, **asp-action**, **asp-route-id**

Genereren de volgende hyperlinks: /employeeManager/Update/86dba8c0-d178-41e7-938c-ed49778fb52a bv. Of /employeeManager/Delete/86dba8c0-d178-41e7-938c-ed49778fb52a

Wanneer de gebruiker klikt op deze links, zal er een Update of Delete action methode in de EmployeeManagerController worden aangeroepen (Deze functionaliteit gaan we later toevoegen)

Opmerking: de id parameter die zal worden gebruikt in de Update en Delete hyperlinks is een onderdeel van de Routing configuratie van ASP.NET Core MVC.

9 _ViewImports.cshtml bestand

De _ViewImports file is een speciaal bestand: het bevat enkel imports van namespaces en directives om Tag Helpers te kunnen gebruiken in razor views en pages

Om de Models en Tag Helpers in onze Razor View te kunnen gebruiken moet de @addTagHelper directive toegevoegd zijn. Om de model classes te kunnen refereren in de Razor views, moeten we ook @using Entities.Models toevoegen

Dit gebeurt standaard in een Razor View Import file.

Open _ViewImports.cshtml en zet als inhoud in de _ViewImports.cshtml:

```
@using CompanyEmployees.MVC
@using CompanyEmployees.MVC.Models
@using Entities.Models
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

De laatste lijn zorgt dat de TagHelpers kunnen worden gebruikt in de Razor Views.

* duidt aan dat alle Tag Helpers van de TagHelpers namespace beschikbaar zullen zijn

10 javascript, css , shared_Layout en ViewStart files

Om onze applicatie te verfraaien en meer interactiviteit te voorzien in de web pagina's, gaan we **javascript** en **css** files toevoegen. We voegen deze files toe onder de standaard webroot **wwwroot** folder;

Onder wwwroot staat een folder scripts en een folder styles

Onder deze vind je .css en javascript (.js) files en standaard jquery library files

We moeten nog specificeren dat we de javascript en css gaan gebruiken in onze razor views, dit doen we in een Shared **_Layout.cshtml** file.

Open de _Layout.cshtml file onder Views/Shared folder en voeg de volgende geel gemarkeerde tekst toe.

```
<li class="nav-item">
    <a class="nav-link text-dark" asp-area="" asp-
controller="EmployeeManager" asp-action="Index">Employees</a>
</li>
```

Deze tekst toont een extra list-item (li) in de header navigation: Volledige inhoud van _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"] - CompanyEmployees.MVC</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">CompanyEmployees.MVC</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="Home" asp-action="Privacy">Privacy</a>
    </li>
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-
controller="EmployeeManager" asp-action="Index">Employees</a>
    </li>
</ul>
</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; 2021 - CompanyEmployees.MVC - <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>

```

Via de link en script tags refereren we naar de css en js files. De tilde (~/) geeft aan dat de applicatie moet vertrekken van de root (wwwroot) folder om deze bestanden te localiseren in de subfolders styles en scripts

Als laatste bekijken we onder de folder Views de Razor View Start file _ViewStart.cshtml

Via deze _ViewStart.cshtml bestand koppelen we de Shared _Layout.cshtml file aan alle views van de applicatie.

De inhoud van deze file is:

```

@{
    Layout = "_Layout";
}

```

Nu kan je de applicatie opstarten en wanneer je op de Employees link in de navigatie klikt (of de url EmployeeManager/index), krijg je de Index view te zien:

[←](#) [→](#) [↻](#) [🔒 localhost:44361/EmployeeManager](#)

CompanyEmployees.MVC [Home](#) [Privacy](#) [Employees](#)

List of Employees

[Insert](#)

Name	Position	Age	Actions	
Jana McLeaf	Software developer	30	Update	Delete
Kane Miller	Administrator	35	Update	Delete
Jos De Klos	Developer	45	Update	Delete

© 2021 - CompanyEmployees.MVC - [Privacy](#)

11 Referenties

<https://www.tutorialsteacher.com/mvc/mvc-architecture>

<https://www.yogihosting.com/aspnet-core-views/>

<https://dotnettutorials.net/lesson/introduction-asp-net-core-mvc/>

https://www.w3schools.com/asp/razor_intro.asp

<https://www.learnrazorpages.com/razor-pages/tag-helpers>

<https://asp.mvc-tutorial.com/sr/445/razor/basic-razor-syntax/>

<https://asp.mvc-tutorial.com/sr/474/tag-helpers/introduction/>

https://www.w3schools.com/html/exercise.asp?filename=exercise_html_attributes2