

Labo Webtechnologieën

Reeks 4: ASP.NET MVC

20 en 27 november 2019

1 Inleiding

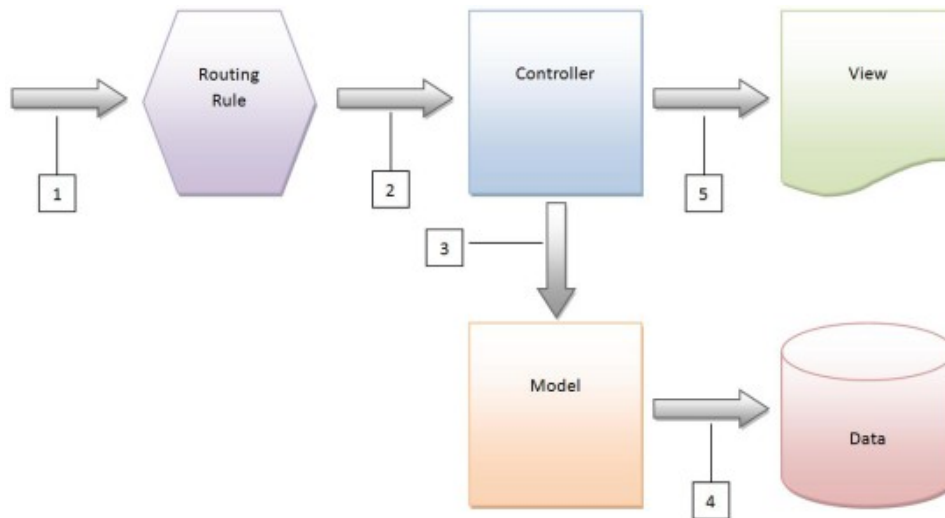
In deze reeks maken we kennis met een nieuw framework dat gebruikt kan worden voor het opbouwen van complexe webapplicaties: ASP.NET MVC. Zoals de naam al doet vermoeden is het een implementatie van het MVC (Model-View-Controller) design pattern voor .NET webapplicaties. We gebruiken hiervoor nu Visual Studio 2017.

In dit labo maken we een webapplicatie voor een theater academie. We maken hiervoor gebruik van een MVC-webapplicatie met ASP.NET Core 2.1 . (Dit stond voorheen ooit bekend als “ASP.NET 5 MVC 6”, maar onderging een naamsverandering. ASP.NET Core is immers sterk gewijzigd ten opzichte van het vorige “ASP.NET 4.6 MVC 5” . Zo is dit framework nu ook open-source beschikbaar en werkt het cross-platform.)

2 De ASP MVC structuur in het kort (herhaling/toe-passing theorie)

Uit de naam kan je al afleiden dat een ASP.NET Core MVC applicatie uit drie soorten componenten bestaat:

- Model: bevat de businesslogica van de applicatie, zorgt voor de opslag van gegevens.
- Controller: Zorgt voor het afhandelen van verzoeken, geeft aanpassingen door aan het model en geeft een view terug aan de gebruiker.
- View: Het visuele aspect van de applicatie.



Figuur 1: ASP.NET MVC model

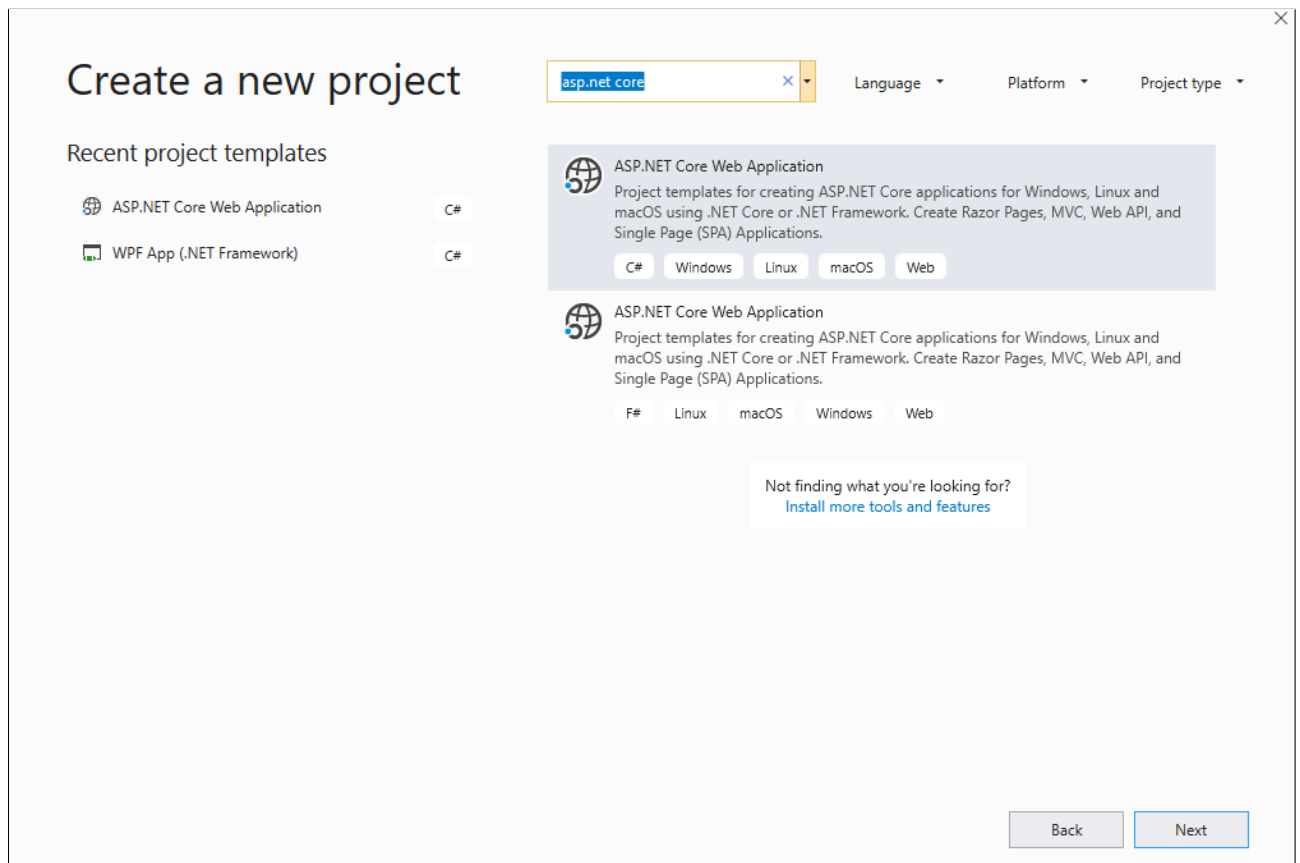
Op basis van de url wordt er nagegaan welke methode van welke controller het request moet afhandelen. Deze methode communiceert met het model en roept een view op. Eventueel kunnen er extra gegevens aan deze view doorgegeven worden.

De interactie in code tussen deze componenten gaat als volgt:

- De methodes (vb *Index*) van een Controller (vb aan *ExampleController*) zijn gekoppeld aan de url's van de webapplicatie. Optioneel hebben deze methodes een aantal parameters die overeenkomen met HTTP-parameters.
- Binnen deze methodes van de Controller staat de code die communiceert met het Model (de data).
- De Views hebben dezelfde naam als de methodenaam van de Controller. Een View is een .cshtml-bestand dat terecht komt in een submap van "Views". Deze submap heeft dezelfde naam als de Controller (vb *Views/Example/Index.cshtml*).
- Deze methodes in de Controller hebben als returntype vaak een "Task", "ActionResult" of "IActionResult". In de methode wordt als returnwaarde dan typisch een "View()" methode aangeroepen om de cshtml-code van de View naar effectieve HTML-code te vertalen en naar de gebruiker te sturen. Je kan variabelen meegeven aan de "View()" methode om zo de gewenste data in de View te steken.

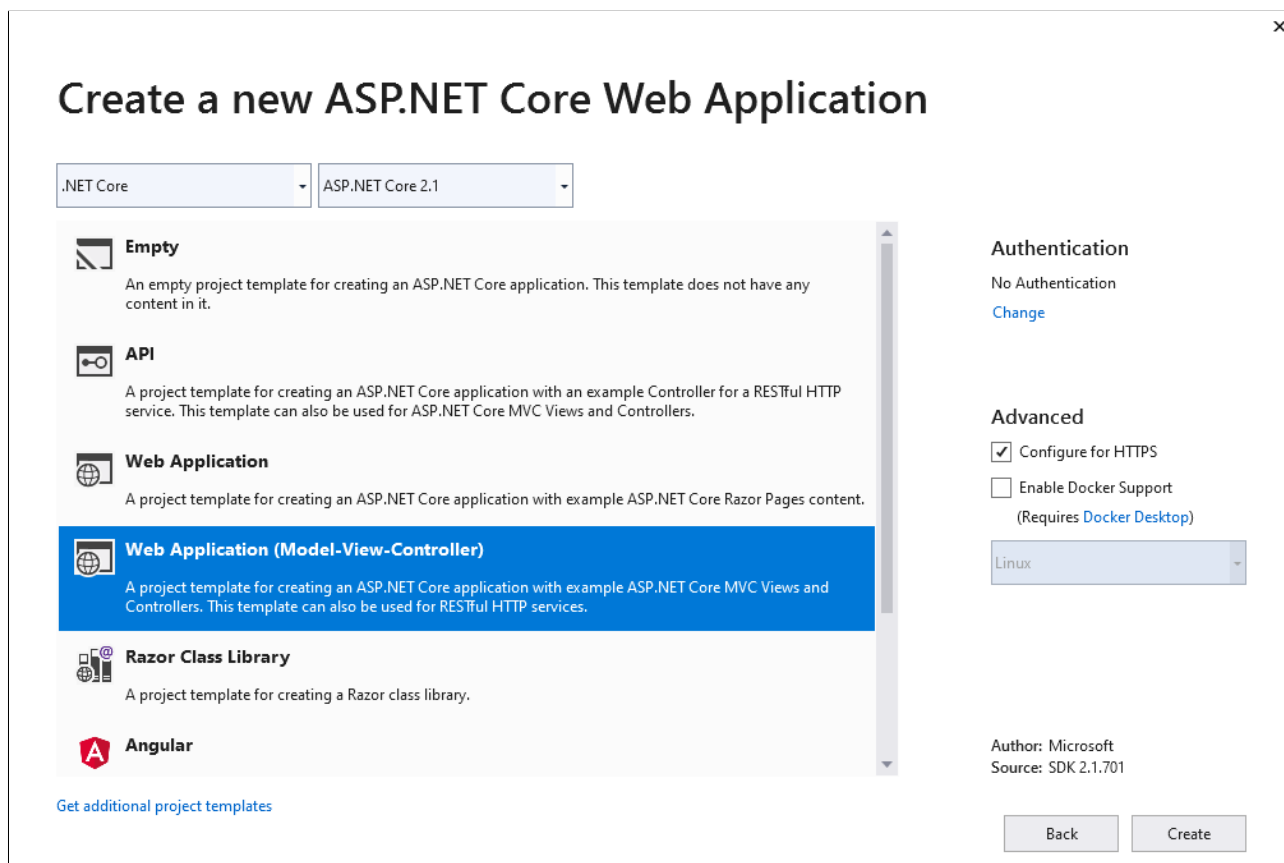
3 Nieuw project

- ◆ Maak in Visual Studio een nieuw "Visual C#" project van het type "ASP.NET Core Web Application", zie figuur 2. Het framework staat in dit venster nog op ".NET Framework 4.6.1", maar dit zal in het volgend venster wel juist op ".Net Core 2.0" komen te staan (zie volgende stap). Kies als locatie voor je project de lokale schijf i.p.v. de UGent-share. Dit zal vlotter zal werken en minder problemen opleveren.

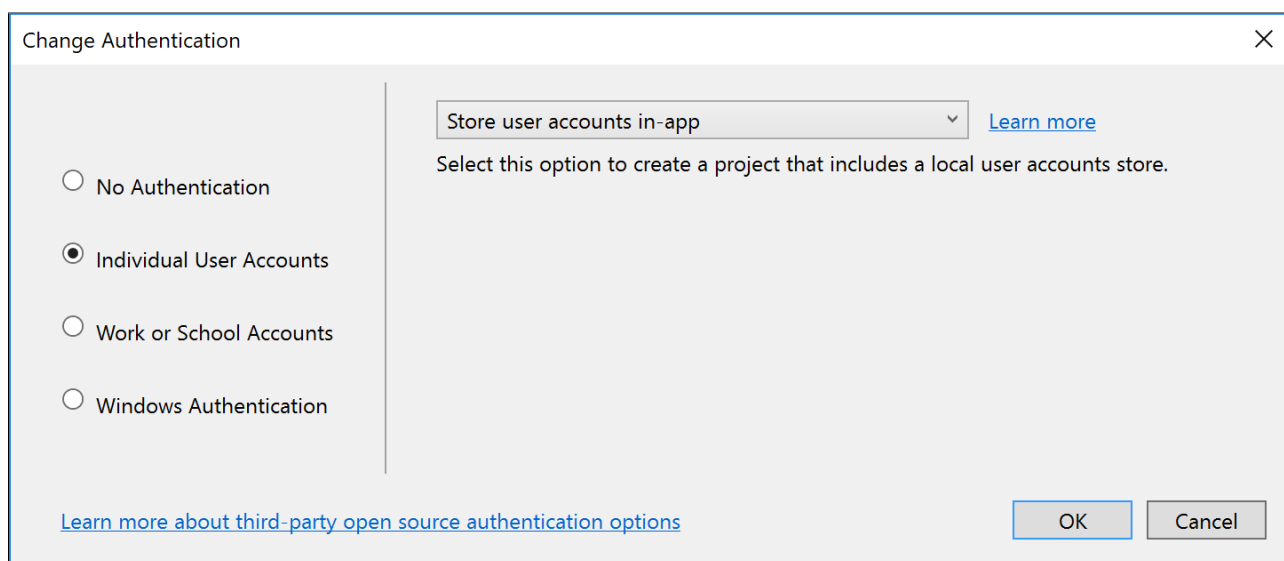


Figuur 2: Een nieuwe “ASP.NET Core Web Application (.NET Core)” .

- ◆ Kies als template **Web Application (Model-View-Controller)** (dus niet de Razor pages), zie figuur 3. Zorg dat Authentication op “Individual User Accounts” ingesteld staat, met in-app user accounts, zie figuur 4.

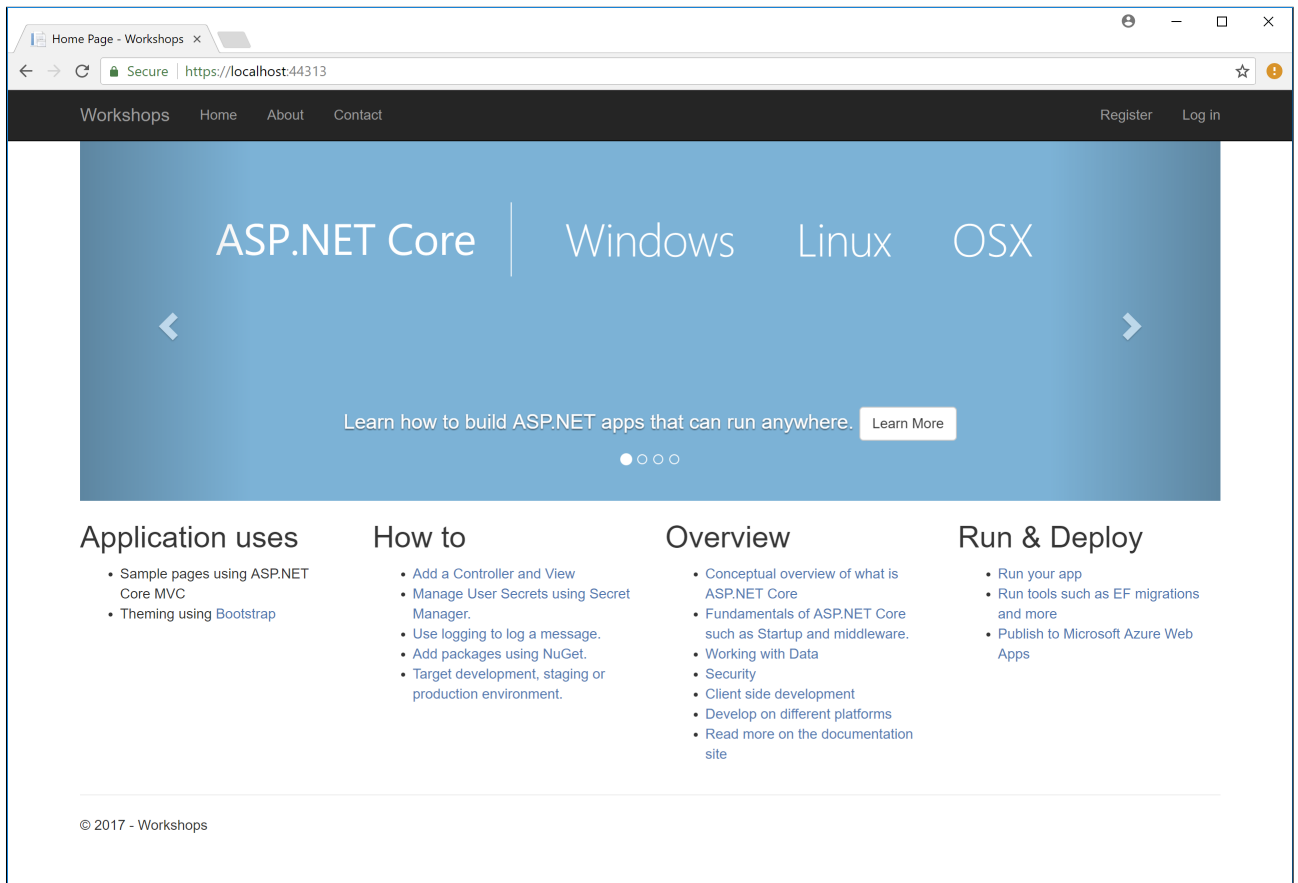


Figuur 3: De “Web Application (MVC)” template met authenticatie



Figuur 4: De authenticatie instelling aanpassen.

- Je kan de applicatie al eens uitproberen door deze te deployen (de “play”-knop met “IIS Express”). Je krijgt een waarschuwing ivm SSL-certificaten die je mag aanvaarden en het self-signed certificaat installeren. Je zal de website zoals figuur 5 bekomen. Bekijk de code van de Controller en Views eens goed, alsook de gegenereerde HTML-code aan client side in je browser. Let ook op de URLs van de verschillende pagina's.

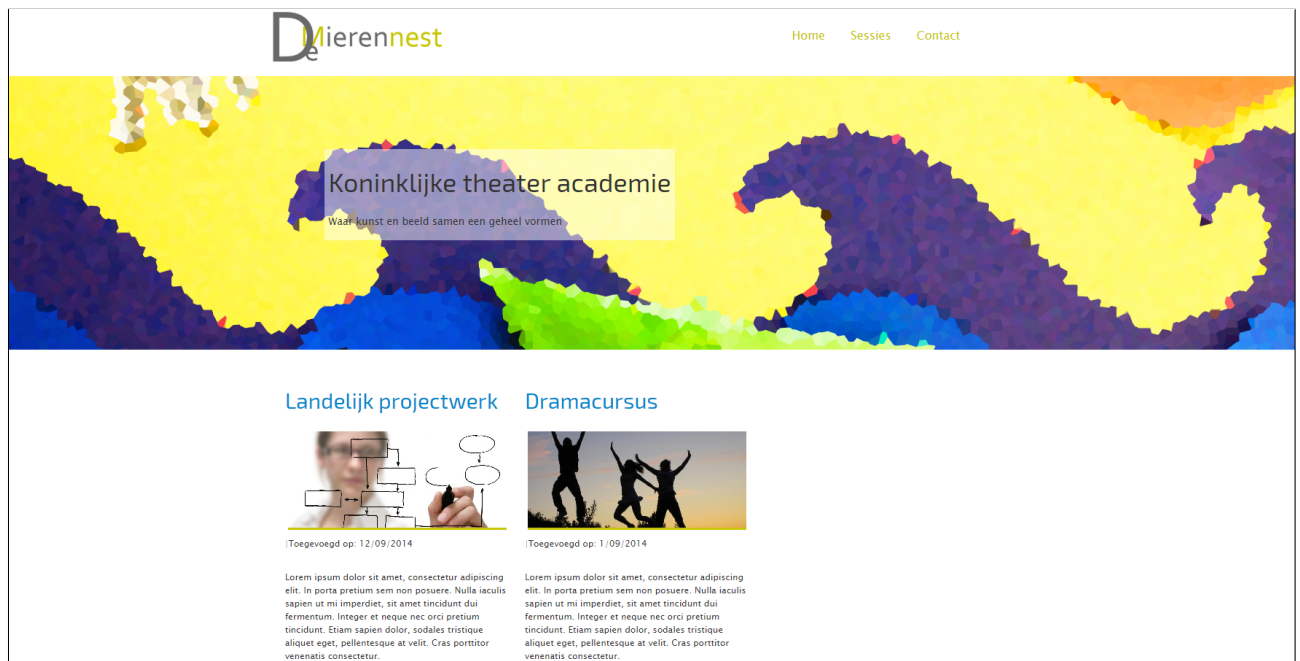


Figuur 5: De standaard indexpagina van een “ASP.NET Core Web Application (.NET Core)” met de “Web Application (MVC)” template

- ◆ Komt de layout je bekend voor ... ? Waar vind je de library hiervoor terug in je project?
- ◆ Bekijk de “About”-pagina. Waar wordt in het project de titel voor deze pagina gedefinieerd? En waar de ondertitel?
- ◆ Waar vind je de balk met de navigatielinks terug in de code? Vergelijk de cshtml-files met de gegenereerde HTML code in je browser. Kijk bijvoorbeeld eens naar de navbar. Zie je de verschillen?

4 1. Home

- ◆ Plaats de code uit de gegeven bestanden (uit de map startcode op Ufora) op een correcte plaats in je project. Zorg ervoor dat je Home pagina eruit ziet als op de onderstaande afbeelding.



Figuur 6: De uiteindelijke index pagina.

4.1 __layout

Via `_ViewStart.cshtml` wordt bij een basis project steeds doorverwezen naar `_Layout.cshtml`. Dit bestand kan gezien worden als een Masterpage die alle gemeenschappelijke code bevat voor de verschillende Views van je project.

- ◆ Gebruik de MVC controls om de navigatie correct in te stellen. Plaats de navigatie op de correcte plaats in het project. (Hint: Actionlinks)

4.2 Een nieuw model en een nieuwe scaffolded controller

Een webapplicatie die een front-end is voor een achterliggende dataset zal typisch volgende zaken bevatten: een overzichtslijst, een pagina voor aanmaak van nieuwe data, de mogelijkheid om data aan te passen of te verwijderen, etc. Dit zijn de typische CRUD (Create, Update, Delete) operaties. Dit betekent dat hiervoor verschillende `c#`-methodes in een controller aangemaakt moeten worden, alsook de bijhorende `cshtml` views.

Omdat dit vaak voorkomende code is, bestaan er in Visual Studio Code Generation tools om deze code automatisch te laten genereren (scaffolding). We hoeven die dan enkel nog aan te vullen met de juiste data.

- ◆ Voeg de meegegeven `NewsMessage.cs` class toe aan je project (op de correcte locatie).
- ◆ Maak een nieuwe `NewsMessagesController` met scaffolding, via rechtermuisknop in de solution overview: Add - New Scaffolded item. Selecteer de "MVC Controller with views, using Entity Framework". Als Model selecteer je de `NewsMessages.cs` class. Bij "Data context class" moet je een nieuwe "TheatersContext" context aanmaken (deze zal data persistentie verzorgen). De checkboxes voor de aanmaak van views mogen geselecteerd zijn (zie figuur 7).

Add MVC Controller with views, using Entity Framework

Model class:

Data context class:

Views:

☒ Generate views

☒ Reference script libraries

☒ Use a layout page:

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

Controller name:

Figuur 7: Aanmaak van een Controller met scaffolding en Entity Framework

De controller, de bijhorende views en de datacontext (in het mapje Data) zijn nu aangemaakt. Zie je waar de TheatersContext gekoppeld wordt aan de Controller (constructor-based Dependency-Injection)? De context zelf wordt eerst in Startup.cs gekoppeld aan een SQL-server. Deze koppeling tussen object en database is wat het Entity Framework (EF) doet: een object-relational mapper zonder expliciet code te moeten schrijven voor het benaderen van een database. Om deze koppeling compleet te maken moeten we wel nog een aantal extra stappen ondernemen.

In appsettings.json zien we dat de “localdb” dataservert gebruikt wordt.

- ◆ Je kan deze localdb SQL server bekijken via menu-component View - 'SQL Server Object Explorer'. Momenteel vindt je hier nog geen database voor ons project terug.

We moeten de database nog aanmaken met de EF Core Migrations feature van Visual Studio. Migrations maakt een database die matcht met het datamodel.

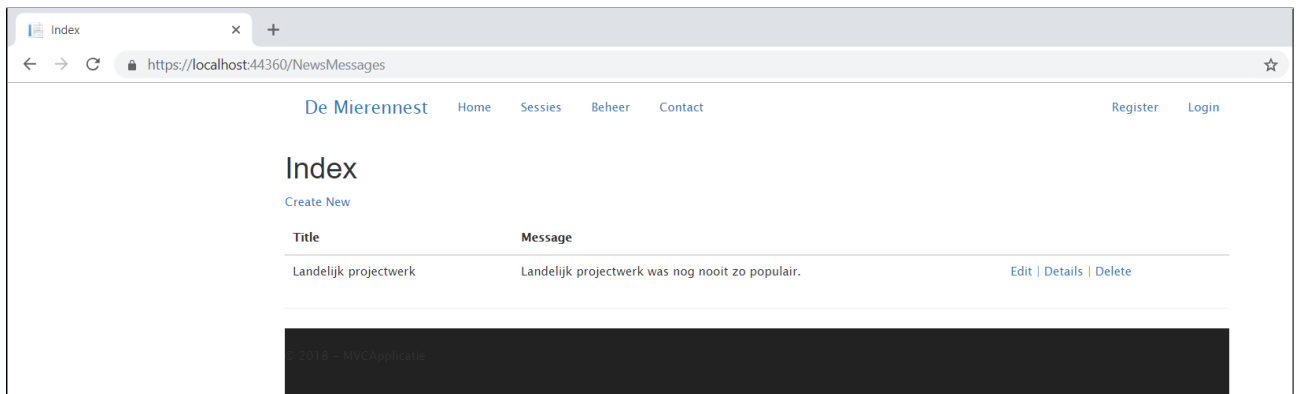
- ◆ In het Tools menu ga je naar “NuGet Package Manager” - “Package Manager Console”. Typ daar volgende commando's in:

```
Add-Migration Initial -Context TheatersContext  
Update-Database -Context TheatersContext
```

Na een refresh zou je nu in localdb een database moeten zien met dezelfde naam als de connectiestring in de appsettings.json.

- ◆ Bekijk de table “dbo.NewsMessage” in de database. Deze is de omzetting van je Model-klasse naar een relationele database. Via rechtermuisknop kan je ook de aanwezige data bekijken.
- ◆ Run je project en browse naar de URL die naar de start-view van je nieuwe controller verwijst. Welke URL is dat?

- Voeg via deze webinterface nu enkele nieuwsberichten toe, pas deze aan of verwijder ze weer en bekijk de opslag van deze data in de database (je moet deze manueel refreshen). Zie figuur 8 en 9.



Figuur 8: Web view van data

Id	Title	Message
1	Landelijk projectwerk	Landelijk projectwerk was nog nooit zo populair.
NULL	NULL	NULL

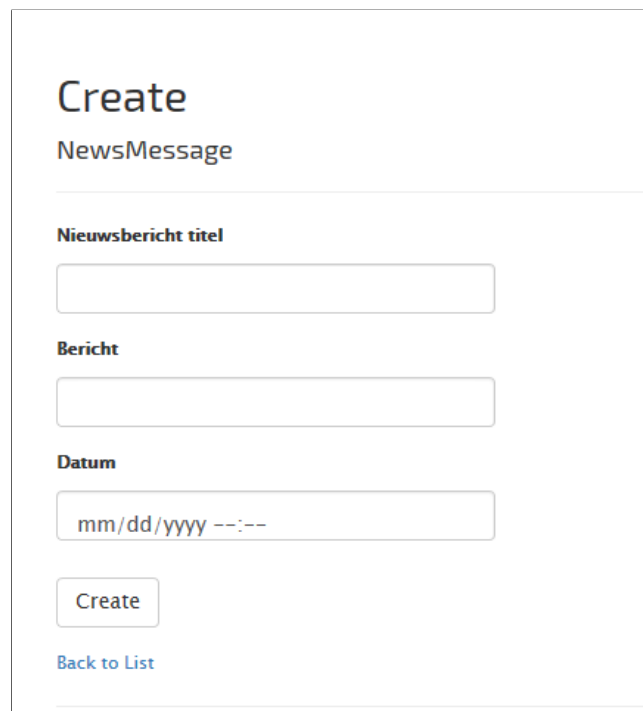
Figuur 9: Database view van data

5 Views aanpassen

Onze index pagina toont nu een overzicht van alle (hard-gecodeerde) nieuws berichten. Deze willen we nu aanpassen zodat de nieuws berichten uit de database gebruikt worden.

- Pas de index view (van NewsMessages) aan zodat deze dezelfde layout heeft als de huidige index view (van Home). Alle berichten die nu op het overzicht getoond worden zijn afkomstig uit de database die in de vorige stap is aangemaakt. Zorg er ook voor dat indien de ID van een nieuwsbericht te hoog is, de dummyimage getoond wordt in plaats van een image gebaseerd op de ID van het bericht.
- Pas de routing aan zodat default de Index pagina wordt getoond van de NewsMessagesController en niet die van de HomeController. Ook bij de navigatie zal de link moeten worden aangepast.
- Zorg voor een gepaste foutboodschap indien er geen nieuwsberichten zijn om weer te geven. Waar dien je dit te implementeren? In de Model, de View of de Controller? Test je implementatie uit door een lege lijst van nieuwsberichten aan de view mee te geven.
- Zorg ervoor dat je de mogelijkheid voorziet op de indexpagina om een nieuwsbericht toe te voegen.
- Voorzie bij de Detailpagina van een nieuwsbericht de mogelijkheid om een nieuwsbericht te verwijderen of om het aan te passen.
- Zorg dat je na aanmaak van een nieuwe nieuwsbericht naar de detailpagina van het toegevoegde bericht gaat, in plaats van naar de Index view. Waar dien je dit aan te passen?

- ◆ Gebruik annotaties in je model om er voor te zorgen dat “Title”, “Message” en “Date” niet leeg kunnen zijn. Voorzie een foutboodschap. Pas ook de labels bij de input velden aan zoals in Figuur 10.

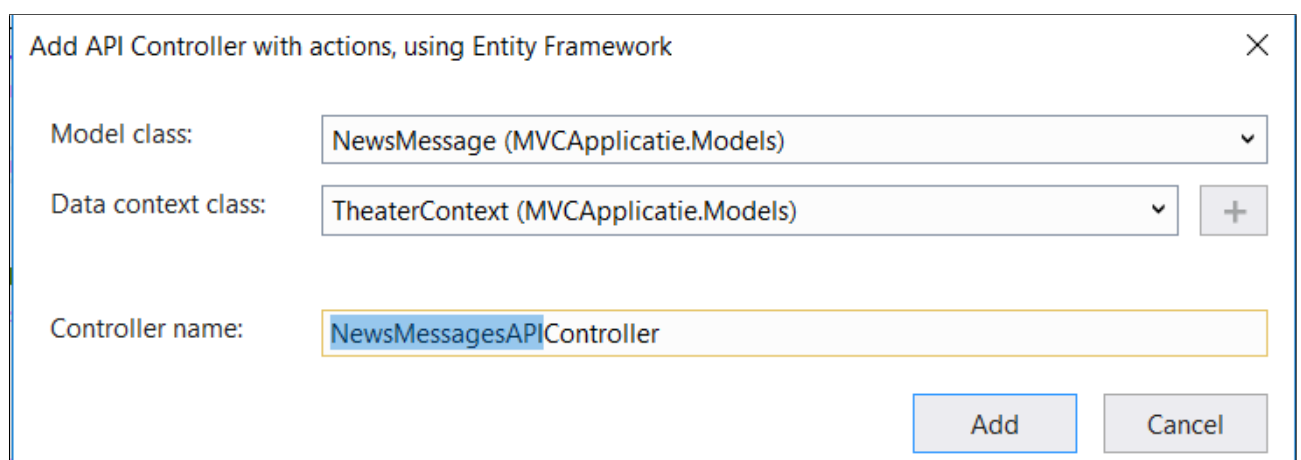


Figuur 10: Create pagina van een nieuws bericht.

6 Web API

We willen nu deze data ivm de NewsMessages van het theater ook openstellen voor 3rd party websites. Daarom willen we de data in een Web API toegankelijk maken (i.e. een RESTful service in .NET Core).

- ◆ Voeg een nieuwe scaffolded controller toe en kies voor “API with actions, using Entity Framework”. Kies als Model class voor de NewsMessages.cs klasse. Kies als Data context class voor de TheatersContext klasse die we eerder gegenereerd hebben. Kies ten slotte een naam voor de Web API controller. Deze zal ook deel uit maken van de URL van je API. Zie figuur 11.



Figuur 11: Delete van een workshop (Web API controller aanmaken)

- ◆ Kijk nu of het je lukt om naar de url van je Web API te surfen en je de data in json-formaat terugkrijgt.
- ◆ Probeer nu de Web API te benaderen met een simpel stukje client code. Gebruik hiervoor de files uit webapi-client-test.zip op Ufora. Pas de javascript-code aan met de juiste url. Je kan die html-pagina vervolgens gewoon lokaal openen. Bekijk met de developer tools in de console wat er misloopt?
- ◆ We dienen dit nu nog expliciet toe te staan in onze Web API. In ConfigureServices in Startup.cs, voeg je eerst volgende toe:

```
services.AddCors(options =>
{
options.AddPolicy("AllowAllOrigins",
builder =>
{
builder.AllowAnyOrigin();
});
});
```

- ◆ Ten slotte dien je deze CORS nog voor de hele Controller of voor bepaalde Actions toe te staan met
`[EnableCors("AllowAllOrigins")]`

7 Internationalisatie

Internationalisatie (I18N) beschrijft het proces om een applicatie meerdere talen en regio's te laten ondersteunen (globalisatie, G11N), en de applicatie hiervoor aan te passen (localisatie, L10N) ¹. Hiertoe zal de applicatie meerdere "cultures" (synoniem: "locales") ondersteunen. Deze kunnen neutraal (een specifieke taal zonder regio, vb "en" voor Engels, "es" voor Spaans, enz.) of specifiek (een specifieke taal met een bepaalde regio, vb "en-US" voor Amerikaans Engels, "en-GB" voor Brits Engels, "es-CL" voor Chileens Spaans, enz.) zijn.

We willen beginnen met het internationaliseren van de boodschap die wordt weergegeven wanneer geen nieuwsberichten beschikbaar zijn.

- ◆ Wanneer er geen nieuwsberichten beschikbaar zijn, diende je eerder een aangepaste boodschap weer te geven in je Index View. Zorg ervoor dat de tekst van deze boodschap in je Controller gedefinieerd wordt via **ViewData**.
- ◆ Voeg een **IStringLocalizer _localizer** toe als instantievariabele in de Controller die je wil internationaliseren. Via constructor based Dependency Injection, injecteer je deze Localizer in je controller (zie <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/localization?view=aspnetcore-3.0>).
- ◆ We moeten deze localisatieservice nu wel expliciet gaan definiëren in onze webapplicatie zodat de webapplicatie de juiste injectie in de controller kan uitvoeren. Voeg daartoe volgende statement toe in de **ConfigureServices** methode van Startup.cs:

¹<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/localization>

```

services.AddLocalization(
    options => options.ResourcesPath = "Resources");

services.AddMvc()
    .AddViewLocalization(
        LanguageViewLocationExpanderFormat.Suffix
    )
    .AddDataAnnotationsLocalization();

```

- ◆ Zorg er vervolgens voor dat je boodschap ingesteld wordt op een key van **_localizer**. Omdat we nog nergens een value voor deze key gedefinieerd hebben, zal de key zelf gebruikt worden.
- ◆ In de opties van onze lokalisatieservice hebben we verwezen naar “Resources” als path voor alle bestanden die de verschillende vertalingen zullen bevatten. Maak daarom een “Resources” map aan binnen je project. Dit kan door met de rechtermuisknop op je project te klikken en te kiezen voor “Add > New Folder”.
- ◆ Deze Resources folder zal subfolders voor de componenten die taalafhankelijk zijn. De vertalingen voor onze controller zullen dus in de subfolder “Controllers” binnen “Resources” moeten komen. Maak deze subfolder nu aan.
- ◆ Maak hierin een NewsMessagesController.en.resx resource file en een NewsMessagesController.fr.resx resource file aan (gebruik dezelfde naam als je controller). Je kan deze files aanmaken via rechtermuisknop “New Item > Resources file”. Zorg ervoor dat de “access modifier” op “public” staat, anders heeft een bezoeker van je webapplicatie geen toegang tot deze resource.
- ◆ Maak nu een Engelse en Franse vertaling van je boodschap in deze resources file.

Nu moeten we nog bepalen welke vertaling naar de client gestuurd moet worden.

- ◆ We moeten eerst in onze webapplicatie aangeven welke “cultures” we effectief willen ondersteunen. Daartoe moeten we via **UseRequestLocalization** de juiste opties meegeven aan onze ApplicationBuilder. Dit gebeurt in de **Configure** methode van Startup.cs . Gebruik hiervoor onderstaand stukje code. Zorg dat dit als eerste opgeroepen wordt binnen de Configure methode.

```

var supportedCultures = new []
{
    new CultureInfo("nl"),
    new CultureInfo("en-US"),
    new CultureInfo("en"),
    new CultureInfo("fr-FR"),
    new CultureInfo("fr"),
};

app.UseRequestLocalization(new RequestLocalizationOptions
{
    DefaultRequestCulture = new RequestCulture("nl"),
    // Formatting numbers, dates, etc.
    SupportedCultures = supportedCultures,
    // UI strings that we have localized.
    SupportedUICultures = supportedCultures
});

```

```
});
```

- ▶ Test je vertaling door de culture expliciet mee te geven in de URL als parameter. Bv surf naar <https://localhost:xxxxx/NewsMessages?culture=fr>
- ▶ Wanneer geen URL-parameter wordt meegegeven, zal er gekeken worden naar de **accept-language** header field van de HTTP-request. Immers, je browser zal een voorkeur voor een bepaalde culture automatisch naar elke web server sturen, op basis van de ingestelde taal van je browser en/of besturingssysteem. Bekijk met de developer tools van Chrome welke culture wordt aangegeven in je HTTP-request wanneer je surft naar je webapplicatie.

Nu heb je de basis van internationalisatie werkende gekregen voor inhoud die vanuit je Controller wordt aangemaakt. We zorgen nu in de volgende stappen dat de tekst in de Views zelf ook kan geïnternationaliseerd worden. Beschouw bijvoorbeeld in eerste instantie de “Voeg een nieuwsbericht toe” button in de Index.cshtml View file van onze NewsMessagesController.

- ▶ Voeg volgende toe aan je Index.cshtml:

```
@using Microsoft.AspNetCore.Mvc.Localization
@inject IViewLocalizer Localizer
```

- ▶ Gebruik vervolgens **@Localizer** op een gelijkaardige manier zoals je in je Controller **_localizer** gebruikte. Bekijk het voorbeeld in de theorieslides en/of <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/localization>.
- ▶ Maak nu ook een “views” folder aan binnen je resource folder met daarin een folder voor de view die je wil gebruiken. Hierin plaats je dan Resources files voor Engels en Frans voor de Index View. Zorg opnieuw ervoor dat de “access modifier” op “public” staat.
- ▶ Zorg er nu voor dat de tekst op de banner (“Koninklijke theateracademie”) ook vertaald wordt.

8 Authenticatie en autorisatie

We zouden er graag voor zorgen dat alleen een ingelogde gebruiker nieuwe berichten kan toevoegen. Hiervoor zullen we de autorisatie moeten instellen voor de gewenste acties. We hebben bij de start van dit project gekozen voor individual user accounts als authenticatie-methode. Hierdoor werd voor ons al wat configuratie toegevoegd aan het project.

- ▶ Voeg de loginfunctionaliteit toe via rechtermuisknop op het project > add new scaffolded item > Identity. Vink de login en logout pagina aan (Zie figuur 11.) Volg de instructies om de migration uit te voeren (vervang WebApplication3Context door de naam van jouw gekozen context:

```
Add-Migration CreateIdentitySchema
    -context WebApplication3Context
Update-Database
```

- ▶ Configureer “Startup.cs” om authenticatie te gebruiken door onderstaande lijn toe te voegen aan de “Configure” methode.

```
app.UseAuthentication();
```

- ◆ Aan de `_layout.cshtml` moeten we nog de `_loginPartial` toevoegen om zo onze login en registreer mogelijkheden beschikbaar te stellen aan de gebruiker. Hiervoor voegen we `<partial name="_LoginPartial"/>` toe aan de navigatie van het project. Dit komt na het sluiten van de `ul` tag.
- ◆ We moeten er nu eerst voor zorgen dat een gebruiker die niet is ingelogd geen nieuw bericht kan toevoegen aan de pagina. Hij mag ook geen berichten kunnen aanpassen en verwijderen. Voeg hiervoor de juiste annotaties toe op de correcte locaties in de controller van de `NewsMessages` en test uit of je de pagina's nog kan bezoeken. (zie <https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-3.0&tabs=visual-studio#test-identity>)
- ◆ Om een gebruiker te laten registreren zouden we graag enkele regels instellen voor het wachtwoord. Deze configuratie komt terecht in `Startup.cs` in de `ConfigureServices` (Zie <https://adrientorris.github.io/aspnet-core/identity/configure-password-policy.html>). Zorg ervoor dat het wachtwoord een lengte moet hebben van 6 karakters. Het gebruik van een cijfer is verplicht alsook het gebruik van een hoofdletter en kleine letter. Er moeten geen non-alfabetische karakters gebruikt worden bij het aanmaken van het wachtwoord. We willen er ook voor zorgen dat het aantal pogingen voor het ingeven van een fout wachtwoord beperkt wordt tot 3 keer. Voeg dit ook toe aan de configuratie.
- ◆ Stel ook in dat de email die gebruikt wordt om te registreren bij de applicatie uniek is.
- ◆ Zorg ervoor dat de koppen voor het toevoegen van een nieuw bericht op de index-pagina en het aanpassen en verwijderen van een bestaand bericht op de detail-pagina verborgen zijn voor niet ingelogde gebruikers.