

# Labo frameworks voor serverapplicaties

## REST webservices in C#

27 november 2024

### 1 Inleiding

In deze reeks maken we kennis met een nieuw framework dat gebruikt kan worden voor het opbouwen van complexe webapplicaties. **ASP.NET Core** is een cross-platform, hoog performant open-source framework voor het bouwen van moderne cloud-enabled applicaties. Bij het opzoeken van informatie op internet is het belangrijk om telkens de volledige term ‘ASP.NET Core’ te vermelden aangezien er meerdere .NET frameworks bestaan. Voor het programmeren maken we gebruik van Visual Studio 2022 als IDE. In dit labo ontwikkelen we een REST-API voor de backend van een nieuwsdienst.

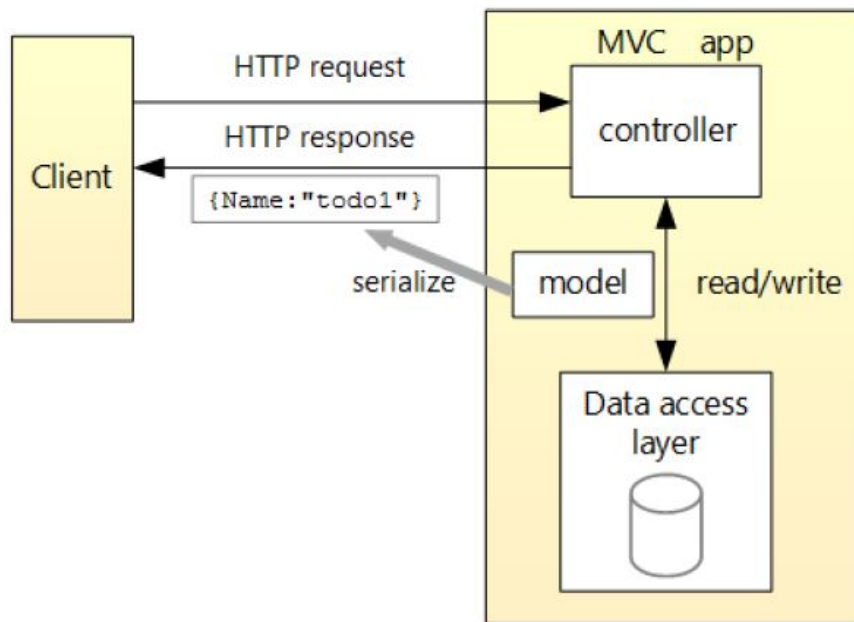
### 2 Restful Webservice ASP.NET Core (herhaling theorie)

De ASP.NET Core API bestaat uit twee componenten:

- Model: bevat de businesslogica van de applicatie, zorgt voor de opslag van gegevens (in een database).
- Controller: Zorgt voor het afhandelen van verzoeken, geeft aanpassingen door aan het 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 geeft de correcte data terug aan de client. De methodes in de Controller hebben als returntype vaak een “Task”, “ActionResult” of “IActionResult”: <https://learn.microsoft.com/en-us/aspnet/core/web-api/action-return-types>.

---

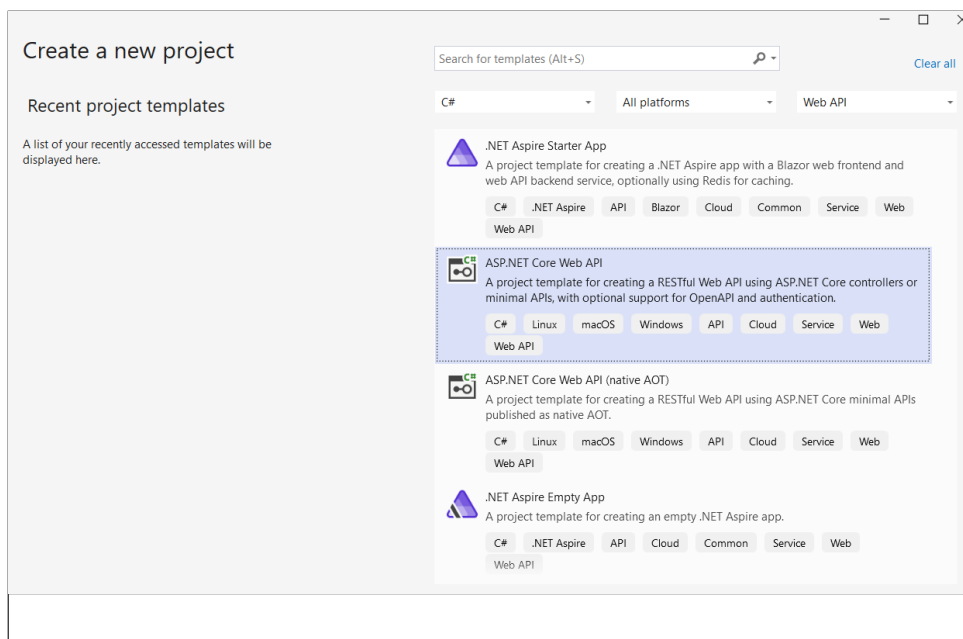


<https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api>

**Figuur 1:** ASP.NET Core API model

### 3 Nieuw project

- Maak in Visual Studio 2022 een nieuw “Visual C#” project van het type “ASP.NET Core Web API”, zie figuur 2. Vul een naam in voor je project en kies een locatie op je lokale schijf om het op te slaan.



**Figuur 2:** Een nieuwe “ASP.NET Core Web API” .

- Zorg dat OpenAPI Support aan staat, zie figuur 3.

**Additional information**

ASP.NET Core Web API C# Linux macOS Windows API Cloud Service Web Web API

Framework [?](#)  
.NET 8.0 (Long Term Support)

Authentication type [?](#)  
None

☒ Configure for HTTPS [?](#)  
☐ Enable container support [?](#)

Container OS [?](#)  
Linux

Container build type [?](#)  
Dockerfile

☒ Enable OpenAPI support [?](#)  
☐ Do not use top-level statements [?](#)  
☒ Use controllers [?](#)  
☐ Enlist in .NET Aspire orchestration [?](#)

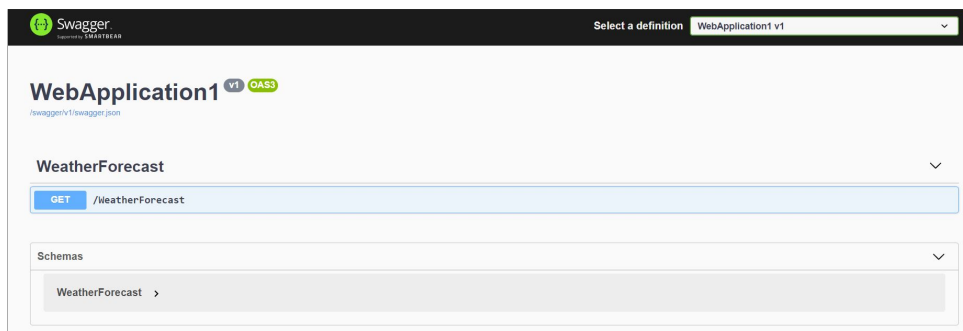
Aspire version [?](#)  
9.0

Back Create

**Figuur 3:** De “Web API” template met OpenApi support

► Je kan de applicatie al eens uitproberen door deze te deployen (de “play”-knop). Indien je een waarschuwing krijgt i.v.m. SSL-certificaten die je mag aanvaarden en je kan het self-signed certificaat installeren. Je zal de website zoals figuur 4 bekomen. Bekijk de gegenereerde code in de Controller en configuratie eens goed.

- Waar vind je “https” configuratie terug?
- Zijn er meerdere profielen beschikbaar?
- Waar vind je de code terug die verantwoordelijk is voor de generatie van het swagger-bestand?



**Figuur 4:** De standaard pagina van een “ASP.NET Core Web Application” met de “Web API” template

► Test de default endpoint eens uit vanuit zowel de browser als “Postman”. Als je een foutmelding krijgt bij het uittesten van het endpoint in Postman. Schakel dan “SSL certificate verification” uit in de instellingen. Aangezien het certificaat dat gebruikt wordt bij HTTPS niet geregistreerd is bij een externe instantie, kan het niet gecontroleerd worden.

## 4 REST API

### 4.1 Model

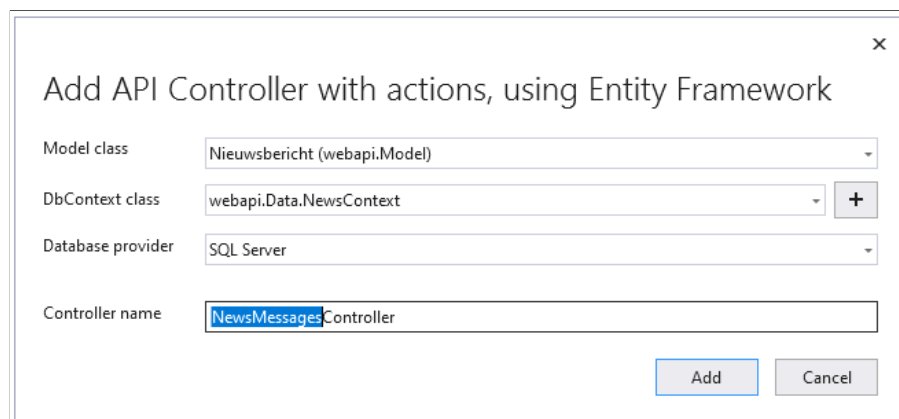
- ◆ Maak een nieuwe folder “Model” aan met daarin een klasse die een nieuwsbericht voorstelt. Een nieuwsbericht bevat een nullable id, titel, bericht en datum. Het id is nullable omdat bij creatie van een nieuwe instantie het id door de database moet bepaald worden en dus initieel leeg is.

### 4.2 In-Memory Controller

- ◆ Maak een nieuwe folder “Data” aan met daarin een “NewsMessagesRepository”. Deze klasse bevat een Dictionary die alle berichten zal bijhouden. Registreer deze klasse als een singleton, zo wordt er maar 1 instantie van deze klasse aangemaakt <https://learn.microsoft.com/en-us/aspnet/core/mvc/controllers/dependency-injection>. Merk op dat in Visual Studio 2022 er geen apart “Startup.cs” bestand meer is. Alle configuratie is opgenomen in “Program.cs” (<https://learn.microsoft.com/en-us/aspnet/core/fundamentals/startup>)
- ◆ Maak een nieuwe “API Controller with read/write actions” NewsMessagesMemoryController aan. Injecteer de net aangemaakte repository in deze controller. En implementeer alle CRUD endpoints.
- ◆ Start de applicatie en test de endpoints uit aan de hand van de browser en Postman. Via welke URL kan je de controller bereiken?
- ◆ Zorg ervoor dat je API de juiste informatie terug geeft: 404 bij een onbestaand item, 400 bij een verkeerde vraag, een link bij een nieuw aangemaakt item, 204 indien geen inhoud, ...

### 4.3 Database Controller

- ◆ Maak een nieuwe NewsMessagesController met scaffolding, via rechtermuisknop op de folder Controllers: Add - New Scaffolded item. Selecteer de “API Controller with actions, using Entity Framework”. Als Model selecteer je de NewsMessages.cs class. Bij “Data context class” moet je een nieuwe “NewsContext” context aanmaken (deze zal data persistentie verzorgen). Zie figuur 5.



**Figuur 5:** Aanmaak van een Controller met scaffolding en Entity Framework

De controller en de bijhorende datacontext (in het mapje Data) zijn nu aangemaakt, bekijk weer aandachtig de gegenereerde code. Zie je waar de NewsContext geïnjecteerd wordt in de Con-

troller (constructor-based Dependency-Injection)? De context zelf wordt eerst in Program.cs gekoppeld aan een SQL-server. Deze koppeling tussen object en database is wat het Entity Framework (EF) doet: een object-relational mapper (ORM) 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 vind 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:

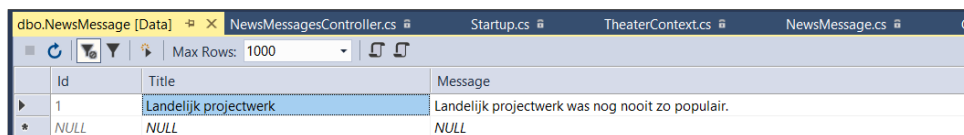
**Add-Migration Initial -Context NewsContext**

**Update-Database -Context NewsContext**

De eerste opdracht maakt een script om de databank te genereren aan op basis van de klasse “NewsContext”. De tweede opdracht voert dit script uit. Als de tweede opdracht een fout geeft, zoek op de foutmelding en pas de juiste configuratie aan. 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.
- ◆ Voeg nu via de webinterface van de API of via postman 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 6.

Let er op dat je het ID laat genereren door de database.



The screenshot shows the SQL Server Object Explorer in Visual Studio. The 'dbo.NewsMessage' table is selected, and its data is displayed in a grid. The grid has three columns: 'Id', 'Title', and 'Message'. The first row contains the values '1', 'Landelijk projectwerk', and 'Landelijk projectwerk was nog nooit zo populair.'.

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

**Figuur 6:** Database view van data

- ◆ Probeer nu de Web API te benaderen met een simpel stukje client code. Gebruik hiervoor de files uit startcode.zip op Ufora. Pas de javascript-code aan met de juiste url. Je kan vervolgens de html-pagina lokaal openen. Inspecteer met behulp van de developer tools, wat loopt er mis?
- ◆ We dienen dit nu nog expliciet toe te staan in onze Web API. [Enable CORS](#).

## 5 Swagger/OpenApi

Doordat we OpenApi support enabled hebben toen we het project aanmaakten is de meeste configuratie al gebeurd. Er wordt automatisch een swagger file gegenereerd en de SwaggerUI is bereikbaar. Enkel de extra documentatie bij de verschillende methodes ontbreekt nog.

- ◆ Laat je project een documentatie file genereren en link deze file met swagger. Voeg bij elke methode een korte uitleg toe. Op deze links vind je extra info: [Getting started with swashbuckle](#) en [Document your code with XML comments](#)
  - ◆ Doordat de Id van een NewsMessage in de database aangemaakt wordt, mag je deze niet meesturen in de body van een POST. Pas het sample request in de SwaggerUI aan zodat het id niet meer deel uit maakt van de body. Het is voldoende om dit aan te passen in jouw commentaar. Wil je ook het gegenereerde voorbeeld aanpassen kijk dan op [Swashbuckle Pro Tips for ASP.NET Web API – Example\(s\) Using AutoFixture](#)
  - ◆ Gebruik annotaties in je model om er voor te zorgen dat “Titel”, “Bericht” en “Datum” niet leeg kunnen zijn. Wat zie je veranderen in de SwaggerUI?
-