# Backend Development

Tuur Vanhoutte

10 februari 2021

# Inhoudsopgave

# 1 .Net Core & Web API

## 1.1 .NET Historiek

### 1.1.1 .NET Framework

- Ontwikkeling .NET Framework & C# begonnen einde Jaren 90 (Opkomst van Java, Microsoft moest reageren ⇒ C#)
- Versie 1.0 .NET Framework op 13 February 2002
- Windows Only
  - Beperkt open source via Home | Mono (`mono-project.com`)
- Was vooral framework voor desktop applicaties (Winforms, WPF)
- Web applicaties via ASP.NET Forms
- Laatste versie .NET Framework is 4.8 (april 2019)
- Geen verdere ontwikkeling meer van .NET Framework

### 1.1.2 Rond 2012 enkel strategische veranderingen

- Dominantie Windows was minder groot geworden
  - Opkomst mobile devices (iPhone, Android)
  - Cloud was belangrijker aan het worden (veel Linux)
  - Open-source werd belangrijker
- Microsoft zag een shift van .NET Framework naar andere technologie (Node, Python, etc. . . )

### 1.1.3 Ontwikkeling .NET Core

- Cross platform (Windows/Linux/macOS)
- Volledig open-source
  - Samsung TV
  - ARM Raspberry Pi
  - . . .
- Clean-up van bestaande .NET Framework code
- Zoveel mogelijk compatibel
- Built for the Cloud

Sinds November 2020 spreken we niet meer over .NET Framework of .NET Core maar over .NET 5, .NET 6, .NET 7, . . .

### 1.1.4 Tijdslijn

- .NET 5 ⇒ November 2020
- .NET 6 ⇒ November 2021 (LTS)
- .NET 7 ⇒ November 2022

- .NET 8 ⇒ November 2023 (LTS)
- LTS ⇒ 3 jaar support
- Zonder LTS ⇒ 1 jaar support

Alles volledig opensource via `https://github.com/dotnet`

## 1.2 Soorten .NET applicaties

### 1.2.1 Type applicaties in .NET 5:

- ASP.NET Web Applicaties
- Console Applicaties
- Xamarin
- Winforms & WPF Desktop applicaties

### 1.2.2 Ondersteuning voor verschillende talen

- C# (wij gebruiken dit)
- Visual Basic
- F# (functional programming)
- C++ (desktop development)

### 1.2.3 Toekomst desktop development

- Zeker nog belangrijk
- Niet alles kan in de browser (vb zware 3D apps, interfacing met machines etc)
- Zit niet meer in MCT opleiding (wel Xamarin natuurlijk)
- Project MAUI
  - 1 framework voor .NET/Xamarin/Windows Desktop
  - Introducing .NET Multi-platform App UI | .NET Blog : `https://devblogs.microsoft.com/dotnet/introducing-net-multi-platform-app-ui/`

## 1.3 ASP.NET Core

= Framework voor het bouwen van Web Applicaties

### 1.3.1 Verschillende soorten applicaties zijn mogelijk:

- Klassieke server-side framework (Razor) (zoals PHP,...)
- Realtime Framework (web sockets etc...) met als naam Signal R
- Frontend Framework Blazor (C# in de browser)
  - Web Assembly support
- **Framework om API's te bouwen , webapi (wij gebruiken dit)**

Cross platform zowel om uit te voeren als te onwikkelen (Visual Studio & Visual Studio Code zijn cross-platform)

### 1.3.2 ASP.NET Web API

= Framework voor het bouwen van API's voor toepassingen

Deze toepassingen zijn:

- Front-End Web App (Vue, Angular, Blazor,...)
- Desktop Applications
- Alles wat HTTP calls kan uitvoeren en JSON begrijpt

Zelfde als HTTP Triggers in Azure Functions (cfr. Module IoT Cloud Semester 3)

## 1.4 HTTP (Herhaling)

= Hyper Tekst Transfer Protocol

- Onderliggende protocol waarop Internet werkt
- Opvragen van tekst, bestanden vanaf servers
- Request *meestal* afkomstig van een webbrowser maar ook smartphone, IoT device
- HTTP zal bepalen hoe een request en response er moeten uitzien
- HTTP bevat een aantal commando's (HTTP Verbs)
- HTTP is stateless, het zal dus geen rekening houden met voorgaande requests
- HTTP is niet sessionless, we kunnen cookies (client-side) gebruiken om data bij te houden
- HTTP is relatief eenvoudig

## 1.5 API Design

- In deze module gaan we API's programmeren
- De bedoeling is dat frontend applications deze gebruiken
- Onze API's zullen enkel JSON data ontvangen en terugsturen
- Oudere API's systemen keren soms ook XML terug (deze zien wij niet)
- Een API noemen we ook soms een endpoint
- De vorm en naamgeving is hier belangrijk

### 1.5.1 Richtlijnen bij het opstellen van API URL's

- Gebruik Engelse woorden
- Start met het woord 'api':
  - http://www.mct.be/api
- In de URL plaatsen we de naam van objecten die we wensen op te halen
  - http://www.mct.be/api/courses

- Willen we specieke cursus ophalen op basis van zijn code steken we dit in de URL
  - http://www.mct/be/api/courses/MCT2IOTCLOUD
- Willen we de weken ophalen, plaatsen we dit ook in de URL
  - http://www.mct/be/api/courses/MCT2IOTCLOUD/weeks
- Zoeken in de weken kunnen we als volgt gaan doen:
  - http://www.mct/be/api/courses/MCT2IOTCLOUD/weeks?q=iothub
- Wil je een bepaalde week ophalen:
  - http://www.mct/be/api/courses/MCT2IOTCLOUD/weeks/1
- Wil je de docent ophalen kan dit op deze manier in de URL:
  - http://www.mct/be/api/courses/MCT2IOTCLOUD/weeks/1/teacher

## 1.6 Anatomie van een ASP.NET Web API Project

### 1.6.1 csproj file

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>net5.0</TargetFramework>
    <RootNamespace>backend_labo01_wijn</RootNamespace>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.Authentication.JwtBearer" Version="5.0.1" NoWarn="NU1605" />
    <PackageReference Include="Microsoft.AspNetCore.Authentication.OpenIdConnect" Version="5.0.1" NoWarn="NU1605" />
    <PackageReference Include="Swashbuckle.AspNetCore" Version="5.6.3" />
  </ItemGroup>

</Project>
```

Figuur 1: csproj voorbeeld

- Beschrijft project
- Zal ingelezen worden door Visual Studio
- Bevat alle referenties naar gebruikte nuget packages
- dotnet build etc zal deze file gebruiken
- We mogen deze wijzigen (maar weet wat je doet)

### 1.6.2 Program.cs

- Main entry point van ASP.NET API applicatie
- Hier begint alles
- Soms moeten we hier iets instellen
- Voorlopig niet van belang

### 1.6.3 Startup.cs

- Naam spreekt voor zich, bevat alle startup code voor onze API
- Hier bepalen we welke **services** we wensen te gebruiken van ASP.NET

```
public void ConfigureServices(IServiceCollection services)
```

Figuur 2: ConfigureServices

```
services.AddControllers();
```

Figuur 3: Controllers

```
services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo { Title = "backend_labo01_wijn", Version = "v1" });
});
```

Figuur 4: Swagger

**Definitie 1.1 (Dependency Injection)** *Dependency Injection is het injecteren van services in de ASP.NET Container*

*(komen we later nog op terug)*

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

Figuur 5: Configureren van de HTTP Pipeline

```
if (env.IsDevelopment())
{
    app.UseDeveloperExceptionPage();
    app.UseSwagger();
    app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "backend_labo01_wijn v1"));
}
```

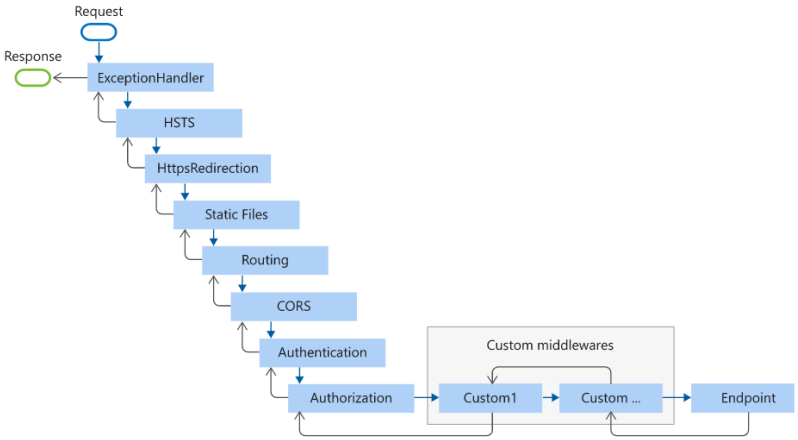
Figuur 6: Configureren van services

```
app.UseRouting();

app.UseAuthorization();
```

Figuur 7: Configureren van Routing, Security etc. . .

**HTTP Pipeline**

- We spreken van middleware componenten in de pipeline
- Deze componenten zijn verbonden en geven de data door aan elkaar
- Ieder component zal zijn bijdrage leveren
- Endpoint is onze code in de controller
- Request en Response gaan door deze pipeline
- ASP.NET Core Middleware: `https://docs.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-5.0`




Figuur 8: De HTTP Pipeline met middleware componenten

### 1.6.4 Appsetting.json

= Configuratie info nodig in de applicatie

- Niet inchecken in GitHub (.gitignore gebruiken)
- Bv.: Mailserver settings, connectionstrings etc...
- We zien later (in een labo) hoe we deze kunnen uitlezen
- We kunnen verschillende files gebruiken:
  - Appsettings.Development.json
  - Appsettings.Production.json
  - ...
- Configuration in ASP.NET Core: `https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/?view=aspnetcore-5.0`

## 1.7 Controllers

- Belangrijkste concept in ASP.NET
- Zorgt voor de verwerking van de requests
- Zal een response terugkeren naar de aanvrager (bv. onze React JS applicatie)
- Is klasse in map **Controllers** in onze applicatie
- Naam moet eindigen op **Controller**
- Klasse moet erven van **ControllerBase**
- Attribute [ApiController] moet er staan om aan te duiden dat het een API is als we [Route] gebruiken op Controller niveau
- Via [Route] kunnen we de URL bepalen

```
[ApiController]
[Route("[controller]")]
public class WineController : ControllerBase
{

    public WineController()
    {
    }
}
```

Figuur 9: Voorbeeld Controller

### 1.7.1 Endpoints

Controller bevat Endpoints = functies die we uitvoeren

- Endpoint bevat HTTP Verb
  - GET
  - POST
  - PUT
- Endpoints voeren de code uit
  - Data ophalen of toevoegen
  - Berekening doen
  - ...

```
[HttpGet]
0 references
public IEnumerable<Wine> GetWines()
{
    return _wines;
}
```

```
[HttpPost]
0 references
public Wine Post(Wine wine)
{
    wine.WineId = _wines.Count + 1;
    _wines.Add(wine);
    return wine;
}
```

Figuur 10: HTTP GET & POST

**Route**

- Een route is de URL naar het Endpoint
- Er zijn verschillende routes mogelijk per Endpoint
- Dezelfde routes zijn mogelijk zolang het HTTP Verb verschillend is
- We kunnen parameters megeven in de route via {}
- Indien geen routes: Default is GET met route = naam van Controller

```
[HttpGet]
[Route("wines")]
[Route("wijnen")]
0 references
public IEnumerable<Wine> GetWines()
{
    return _wines;
}
```

```
[HttpDelete]
[Route("wines/{wineId}")]
0 references
public ActionResult Delete(int wineId)
{
    return Ok();
}
```

Figuur 11: Meerdere routes per endpoint mogelijk (links). Delete request met parameters (rechts)

```
[ApiController]
[Route("[controller]")]
public class WineController : ControllerBase {

    [HttpGet]
    [Route("wines")]
    [Route("wijnen")]
    public IEnumerable<Wine> GetWines()
    {
        return _wines;
    }
```

Figuur 12: https://localhost:5001/wine/wines en https://localhost:5001/wine/wijnen

```
[ApiController]
[Route("[controller]")]
public class WineController : ControllerBase {

    [HttpGet]
    public IEnumerable<Wine> GetWines()
    {
        return _wines;
    }
```

Figuur 13: /wine haalt hij uit de naam WineController

```
[ApiController]
[Route("api")]
public class WineController : ControllerBase {

    [HttpGet]
    [Route("wines")]
    public IEnumerable<Wine> GetWines()
    {
        return _wines;
    }
```

Figuur 14: Best practice: met api in de naam

```
[ApiController]
[Route("api")]
public class WineController : ControllerBase {

    [HttpGet]
    [Route("wines/{year}")]
    public IEnumerable<Wine> GetWines(int year)
    {
        return _wines;
    }
```

Figuur 15: Route met parameter. https://localhost:5001/api/wines/2005

- Endpoints keren altijd iets terug:
  - Status codes
  - Data die we opvragen
- Niet altijd makkelijk wat je moet kiezen
- Status codes uit HTTP standaard 5 grote verdelingen

- **1xx: Informational** – Communicates transfer protocol-level information.
- **2xx: Success** – Indicates that the client's request was accepted successfully.
- **3xx: Redirection** – Indicates that the client must take some additional action in order to complete their request.
- **4xx: Client Error** – This category of error status codes points the finger at clients.
- **5xx: Server Error** – The server takes responsibility for these error status codes.

Figuur 16: Status codes

Overzicht statuscodes: `https://restfulapi.net/http-status-codes/`

Meest gebruikt:

- 200 OK
- 201 Created
- 400 Bad Request, 401 Unauthorized, 403 Forbidden, 415 Unsupported Media Type
- 500 Internal Server Error

**Returnen van data en statuscodes: veel ingebouwde klassen**

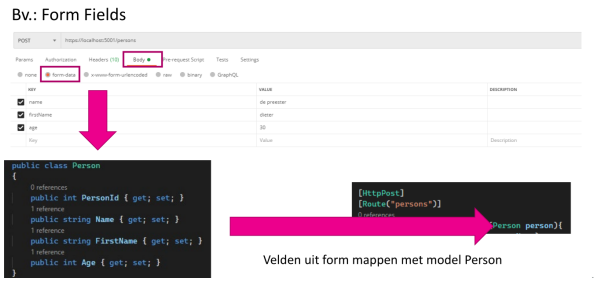
- OkObjectResult(data)
- Ok()
- NotFoundObjectResult(data)
- NotFound()
- StatusCodeResult(int statuscode)

## 1.8 Model binding

- Default enkel bij HTTP Post ⇒

- ASP.NET Core volgorde voor ophalen van key/values:
  - Form fields
  - Request body (enkel bij gebruik [ApiController] attribuut)
  - Route
  - Querystring

### 1.8.1 Voorbeelden




Figuur 17: Voorbeeld: Form Fields


Figuur 18: Voorbeeld: Request Body (enkel bij gebruik van [ApiController] attribuut)

Via Route attribuut kunnen we parameters doorgeven:

```
[HttpPost]
[Route("persons/{name}/{firstName}/{age}")]
```

Figuur 19: We bepalen de parameter namen tussen de {parameter naam}

```
[HttpPost]
[Route("persons/{name}/{firstName}/{age}")]
0 references
public ActionResult AddPersonRoute(string name,string firstName,int age){
```

Figuur 20: In de C# functie voorzien we dezelfde naam als in de route

```
[HttpPost]
[Route("persons/{name}/{firstName}/{age}")]
0 references
public ActionResult AddPersonRoute([FromRoute]Person person){
```

Figuur 21: We kunnen ook de waarden uit de route direct opslaan in het object: via **[FromRoute]** attribuut


Figuur 22: Kan ook via Querystring

## 1.9 Configuration

- Bepaalde info willen we niet hardcoden
  - Mailserver, connectionstring, password, . . .
- We moeten dit buiten de applicatie kunnen opslaan (appsettings.json, Azure Keyvault)

### 1.9.1 appsettings.json

- JSON file die bij de applicatie zit
- Verschillende versies mogelijk
  - Development/Testing/Production..
- Eerst geladen
- Bevat dingen die zowel voor development als production geldig zijn
- appsettings.Development.json (naargelang de omgeving inladen)
  - Overriden wat reeds in appsettings.json szit
- JSON file

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "MailServerSettings" :{
    "ServerName" : "server",
    "UserName" : "test-mailer",
    "Password" : "test-passwrd"
  }
}
```

Figuur 23: Probeer met Secties te werken (zie MailServerSettings)

- We gebruiken Option pattern voor inladen data uit configuratie files
- Eerst klasse maken die overeenkomt met de waarden in de JSON file

```
public class MailServerSettings
{
    0 references
    public string ServerName { get; set; }
    0 references
    public string UserName { get; set; }
    0 references
    public string Password { get; set; }
}
```

Figuur 24: Klasse die overeenkomt met het MailServerSettings object in appsettings.json

- Registreren van de sectie in de startup en koppelen aan de klasse
- Hierdoor krijgen we een strongly typed settings file

```
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<MailServerSettings>(Configuration.GetSection("MailServerSettings"));
```

Figuur 25: Configuration.GetSection("SectionNaam")

Gebruik in de controller:

- IOptions<MailServerSettings> via CONSTRUCTOR (CTOR) binnenbrengen
- Value property zal deze uitlezen en opslaan in property _mailserverSettings
- Hierdoor kunnen we deze vlot uitlezen in de controller en gebruiken

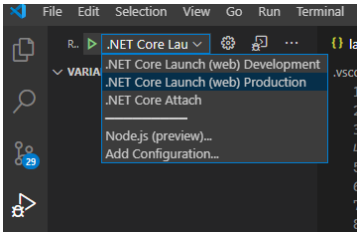
```
private readonly MailServerSettings _mailserverSettings;

0 references
public DemoController(IOptions<MailServerSettings> options, ILogger<DemoController> logger)
{
    _mailserverSettings = options.Value;
    _logger = logger;
}
```

Figuur 26

In VSCode:

- launch.json openen
- Hier kan je de profielen vinden
- Copy/paste profile
- Per profiel kan je instellingen maken zoals URL of ASPNETCORE_ENVIRONMENT


Figuur 27: launch.json




Figuur 28: In de dropdown kies je wat je wil starten

**Visual Studio 2019**

- We bepalen de profielen in de map Properties
- In launchSettings.json

### 1.9.2 Azure Keyvault

- Kluis in Azure waar alle info zit
- Beste oplossing voor productie projecten

## 1.10 Wat moet je kennen?

- Zorg dat je de HTTP Request/Response kan uitleggen
- Wat zijn statuscodes en wanneer gebruik ik welke
- Een duidelijke API URL kunnen opstellen
- Manieren van modelbinding
- Hoe configureer je uw omgeving met profielen