

# Labo 01

---

## Doelstellingen

---

In dit labo is het de bedoeling de nodige tools te installeren en een eerst project aan te maken en te verkennen. Daarnaast leren we ook hoe we moeten debuggen. Een tweede doelstelling is het herhalen van het gebruik van Git, Github.

## Installatie

---

Het is de bedoeling dat je deze module zowel op **PC** (Windows 10/Linux) als Mac (OSX) kan volgen. Hiervoor maken we zoveel mogelijk gebruik van **Visual Studio Code**. Indien u dit wenst mag ik ook Visual Studio Community gebruiken (zowel Mac als Windows). We gaan ook enkel andere tools gebruiken zoals **Windows Terminal** en **Azure Data Studio**. Ik zal proberen zoveel mogelijk de demo's en voorbeelden in Visual Studio Code te tonen.

## Downloads:

Installeer onderstaande (indien u dit nog niet hebt staan) in deze volgorde:

[Download .NET Core SDK for Visual Studio Code \(microsoft.com\)](#)

[Visual Studio Code - Code Editing. Redefined](#)

<https://www.postman.com/>

[Download and install Azure Data Studio - Azure Data Studio | Microsoft Docs](#)

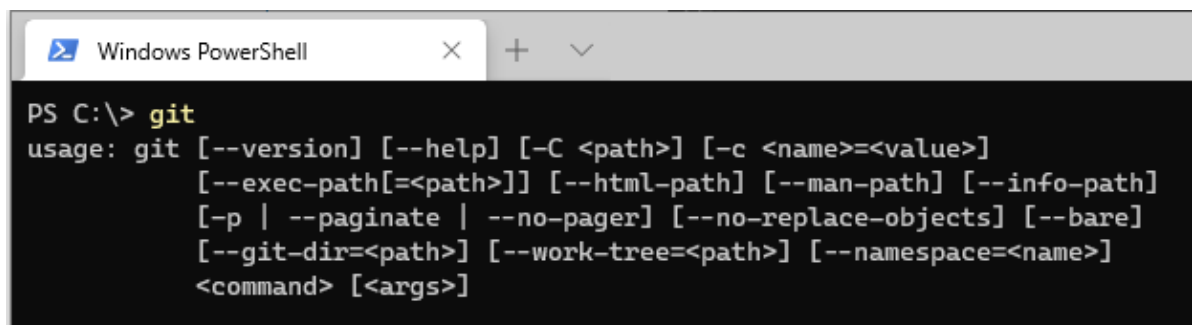
[Get Windows Terminal - Microsoft Store](#)

[iTerm2 - macOS Terminal Replacement](#)

[GitHub Desktop | Simple collaboration from your desktop](#)

Controleer ook of je Git op je systeem hebt staan. Open een command prompt en tik

`git`



```
PS C:\> git
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
      [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
      [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
      [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
      <command> [<args>]
```

bovenstaande zou moeten verschijnen. Indien niet dan moet je nog **git** apart installeren via

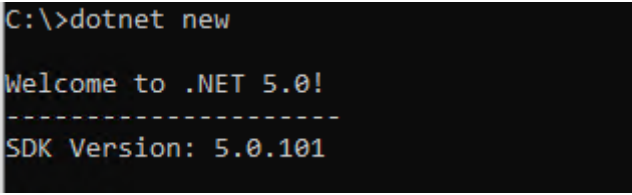
[Git \(git-scm.com\)](https://git-scm.com/)

## Eerste .NET Web API

---

Open een terminal venster of een command prompt en tik het volgende commando:

```
dotnet new
```

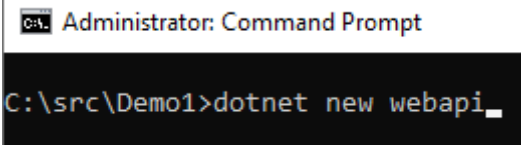


```
C:\>dotnet new

Welcome to .NET 5.0!
-----
SDK Version: 5.0.101
```

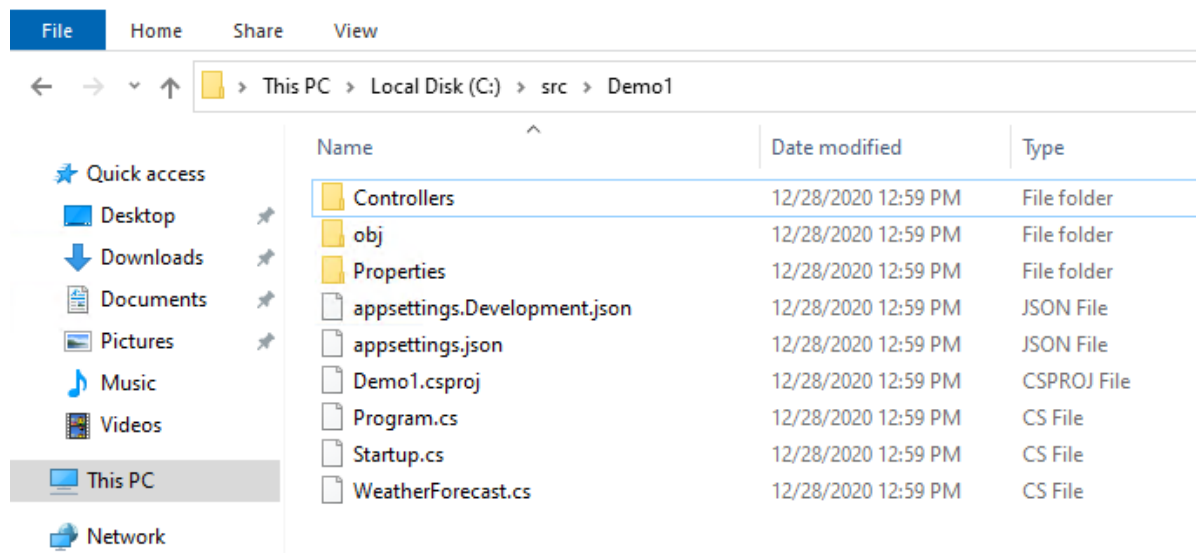
Als u bovenstaande te zien krijgt is alles goed geïnstalleerd en kan u verder. Maak nu een folder aan op je systeem. Ik werk meestal in de map **C:\src of c:\source**. Daarin staan alle GitHub repositories die ik gebruik. Op Mac maak ik gebruik van de map **/Documents/GitHub** en daar sla ik alle repositories op die ik gebruik. Maak in uw source map een nieuwe map **Demo1** aan. Het vollende path is dan **C:\source\Demo1** of **/Documents/GitHub/Demo1**. Ga nu in de map staan via de command prompt en ik

```
dotnet new webapi
```



```
C:\src\Demo1>dotnet new webapi
```

Nu zal er een nieuw **ASP.NET Core Web API** project aangemaakt worden. Bekijk ook eens zelf welke andere soorten projecten kan aanmaken door gewoon **dotnet new** te tikken. Via de Verkenner (Windows) of finder (OSX) zien we dat een heel wat bestanden zijn aangemaakt. In de theorie hebben we deze in detail besproken.



We dit project bevat ook een demo API. We kunnen deze reeds testen. Om te controleren of er geen compilatie problemen zijn tikken we:

```
dotnet build
```

Dit zou geen fouten mogen opleveren.

```
C:\src\Demo1>dotnet build
Microsoft (R) Build Engine version 16.8.0+126527ff1 for .NET
Copyright (C) Microsoft Corporation. All rights reserved.

Determining projects to restore...
All projects are up-to-date for restore.
Demo1 -> C:\src\Demo1\bin\Debug\net5.0\Demo1.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:02.74

C:\src\Demo1>
```

Het project starten kan je doen via:

```
dotnet run
```

```
C:\src\Demo1>dotnet run
Building...
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: https://localhost:5001
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://localhost:5000
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\src\Demo1
```

Dotnet zal terug een **build** maken van het project maar daarna ook opstarten. We krijgen ook de poort te zien waar naar we kunnen surfen: <https://localhost:5001>. We moeten wel de volledige URL opgeven. Deze is <https://localhost:5001/WeatherForecast>. Test dit uit met **Postman**. De **HTTP Verb is GET**. Later zien we nog in detail hoe we URL's kunnen opbouwen.

The screenshot shows the Postman application interface. On the left, the 'History' tab is active, showing a list of recent requests. The main area displays an 'Untitled Request' with the method 'GET' and the URL 'https://localhost:5001/WeatherForecast'. The 'Query Params' section is empty. The 'Body' tab is selected, showing a JSON response in 'Pretty' format:

```
{
  "date": "2020-12-29T13:11:06.2520648+01:00",
  "temperatureC": 17,
  "temperatureF": 62,
  "summary": "Freezing"
}
```

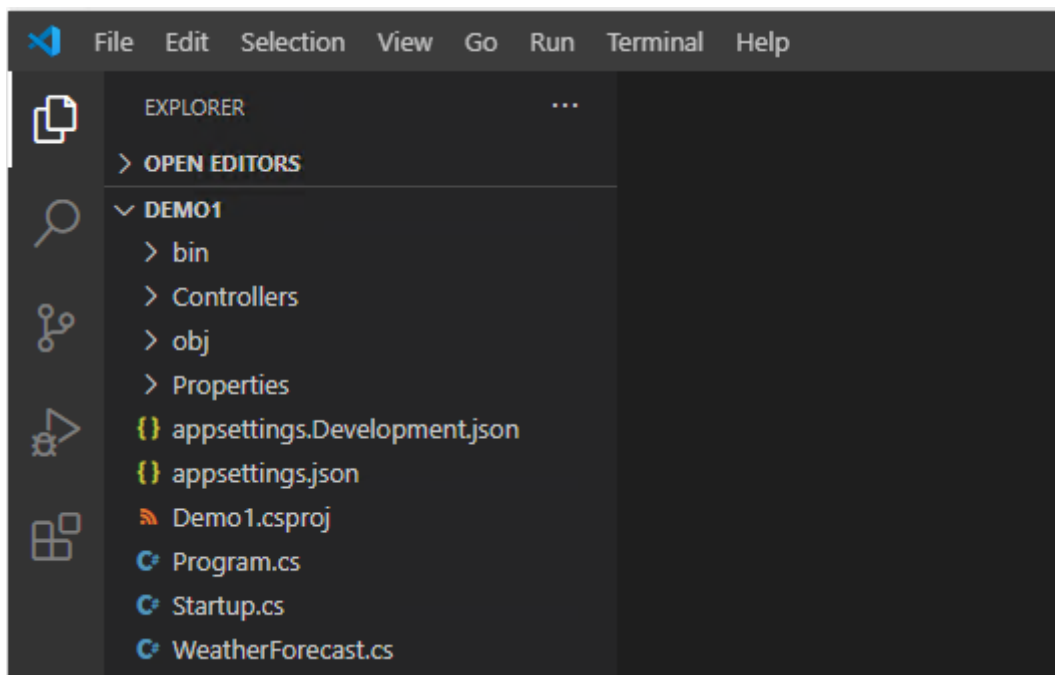
Enkele andere interessant commando's zijn:

```
dotnet new --help
```

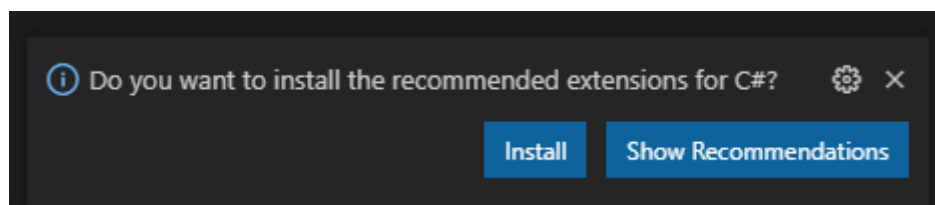
```
dotnet new webapi --name naamvanuwproject
```

## Visual Studio Code Configureren for ASP.NET Core debuggen

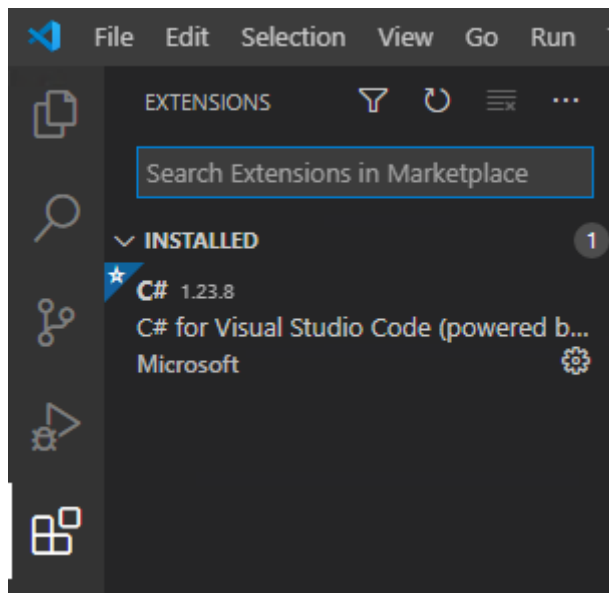
Eigenlijk kan je gelijke welke texteditor gebruiken voor het schrijven van de C# code. Wij maken echter gebruik van Visual Studio Code aangezien deze op alle platformen (Windows, OSX, Linux) ongeveer hetzelfde zal werken. We moeten echter wel een aantal zaken instellen en extra installeren. Start Visual Studio Code en open de map **Demo1** waarin onze test applicatie staat. Dit ziet er ongeveer als volgt uit:



Visual Studio Code zal ons normaal zelf reeds vragen of we de C# extensie wensen te installeren. We kiezen voor **Install**



Extensies kan je altijd toevoegen via het volgende icoon: je krijgt een overzicht van de reeds geïnstalleerde extensies en kan ook zoeken op andere extensies vb: Python.



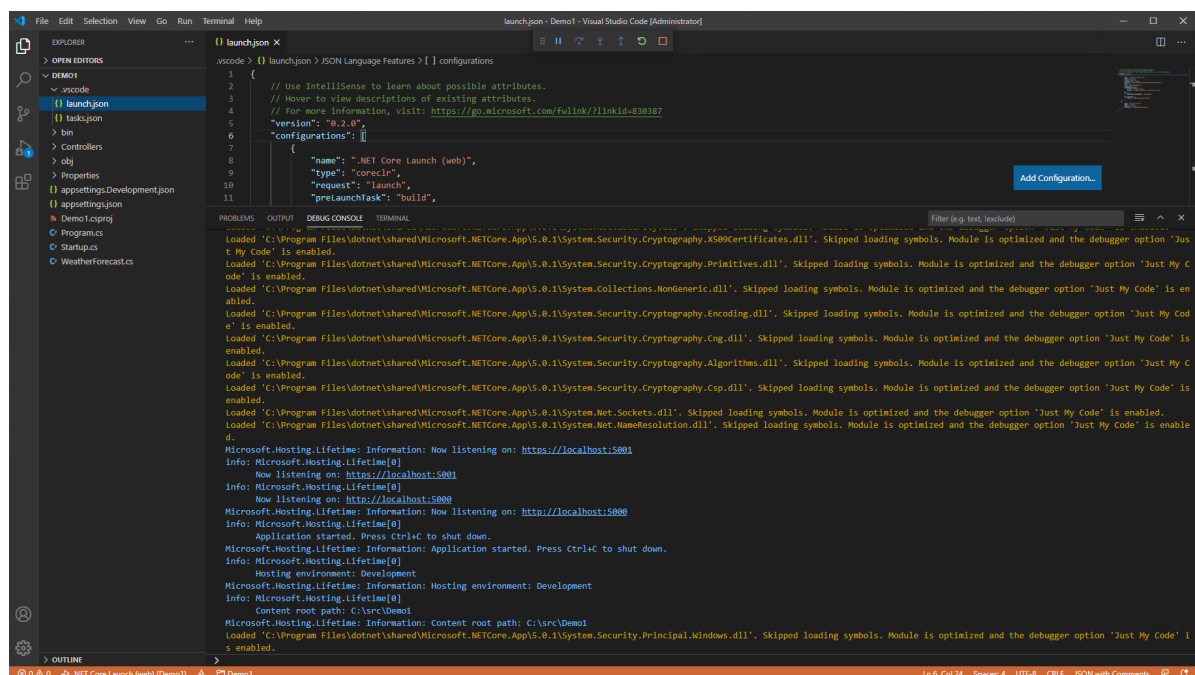
Een tweede handige extensions is

[VS Sharper for C# - Visual Studio Marketplace](#)

Gelieve deze ook te installeren.

Kies nu in het menu **Run** → **Start Debugging** en kies **.NET Core** uit de lijst. Er zal een mapje **.vscode** gemaakt worden waarin er specifieke Visual Studio Code files komen te staan.

**Launch.json** is zo een file en bevat info om vanuit Visual Studio Code onze toepassing te kunnen starten of debuggen. Kies nu terug **Run** → **Start Debugging** of druk **F5**. De toepassing zal nu opstarten:

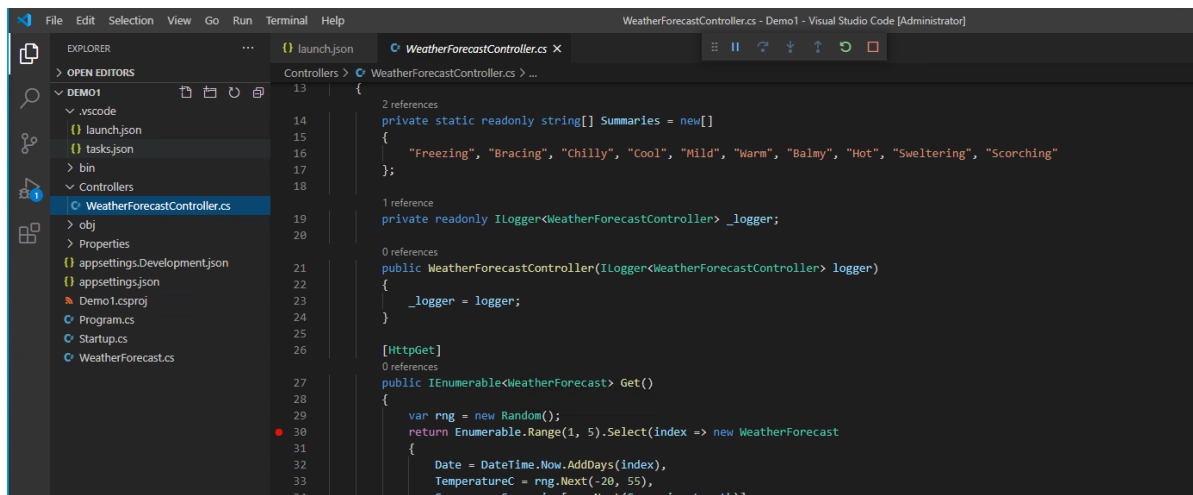


Test nu terug met **Postman** of de API call werkt.

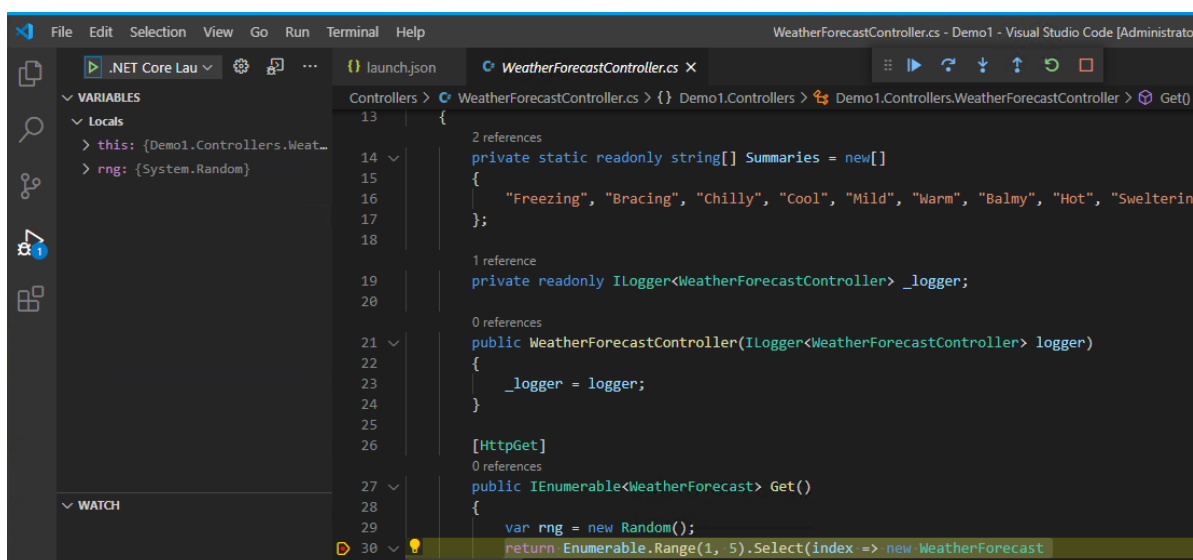
Het debuggen doen we via de balk bovenaan: u zou dit reeds moeten kennen uit vorige modules. De werking is dezelfde, we kunnen stoppen, stap voor stap verder gaan etc...



Om een **breakpoint** te plaatsen openen we de file **WeatherForecastController.cs**. We plaatsen een **breakpoint** op bvb: lijn 30 We doen dit door links van het lijnnummer te klikken. Er zal een rode bol verschijnen.



Roep nu terug de API aan via **Postman**. Er zal een gele lijn verschijnen en het programma zal stoppen: Links bij de iconen druk je op het **pijltje** (voorlaatste icoon). We krijgen toegang tot het debug venster. We kunnen oa. variabelen inkiijken of een watch plaatsen op een variabele.

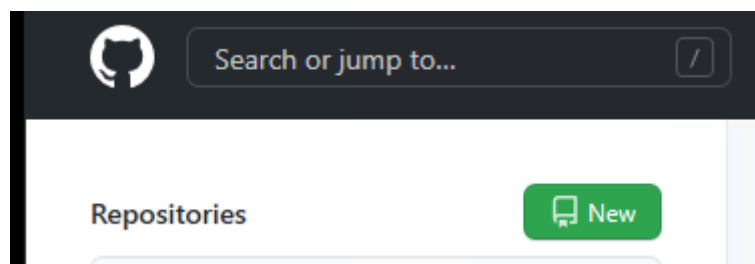


Stopzetten doen we door te drukken op het **rode vierkant** in de balk bovenaan. Bekijk ook de ander opties in de balk en zorg dat je deze begrijpt en kent.

## GitHub Dekstop of GitKraken

Onderstaande is met **GitHub Desktop** gemaakt maar u mag ook andere tools zoals **GitKraken** gebruiken. Ik zal beide door elkaar gebruiken in de lessen.

In deze module gaan we **GitHub** zoveel mogelijk gebruiken. In dit stuk herhalen we nog eens hoe je dit kan aanpakken. Ga naar Github en login. Daarna maken we een nieuw **Repository** aan.



Vul nu onderstaande gegevens in en maak aan.

# Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

## Repository template

Start your repository with a template repository's contents.

No template ▼

Owner \*

dieterhowest ▼

Repository name \*

demo2

Great repository names are short and memorable. Need inspiration? How about [animated-octo-parakeet?](#)

Description (optional)

☐ Public

Anyone on the internet can see this repository. You choose who can commit.

☒ Private

You choose who can see and commit to this repository.

## Initialize this repository with:

Skip this step if you're importing an existing repository.

☒ Add a README file

This is where you can write a long description for your project. [Learn more.](#)

☐ Add .gitignore

Choose which files not to track from a list of templates. [Learn more.](#)

☐ Choose a license

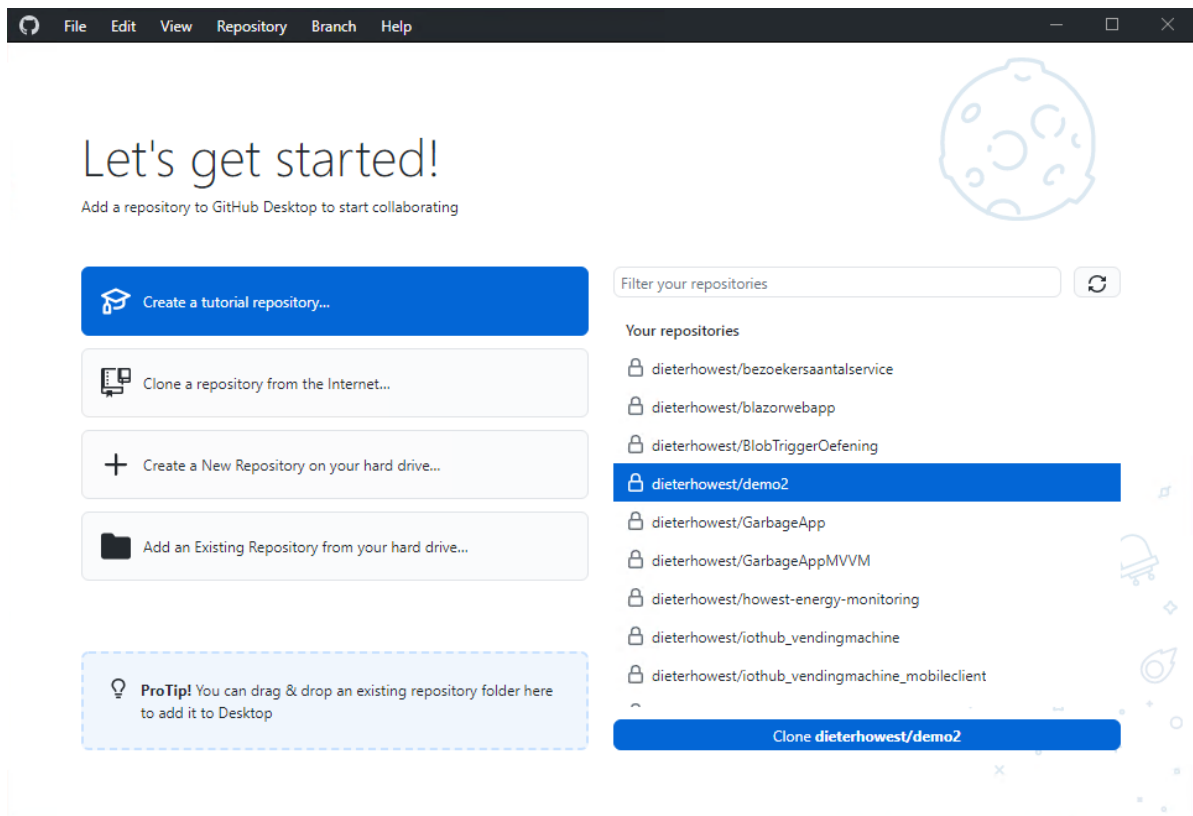
A license tells others what they can and can't do with your code. [Learn more.](#)

Create repository

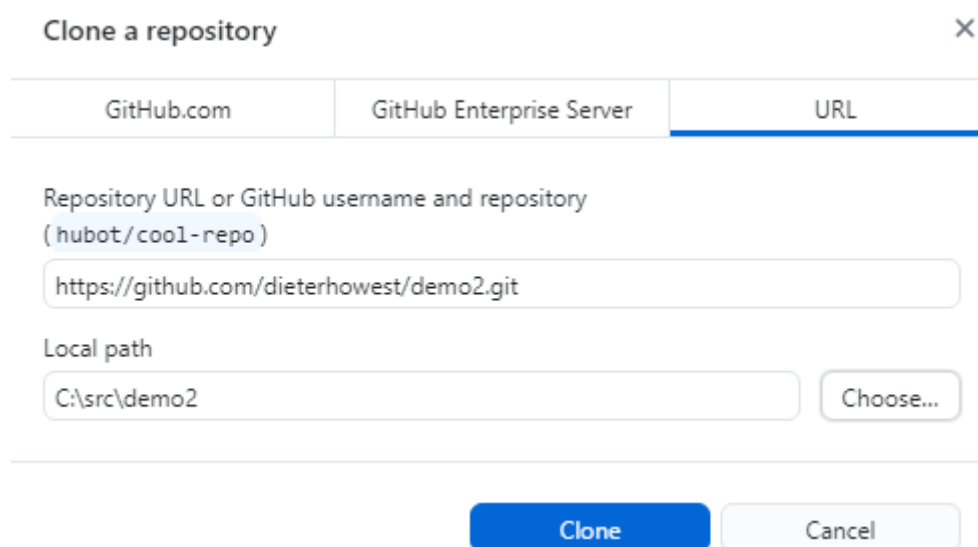
Nu gaan we de lege repository lokaal klonen. Kies **Code** en daarna kopiëren we de URL.

The screenshot shows the GitHub repository creation page. The repository is named 'demo2' by user 'dieterhowest'. The 'Code' button is highlighted, and a dropdown menu is open showing options to clone the repository. The 'Clone' option is selected, and the URL 'https://github.com/dieterhowest/demo2.' is displayed. The dropdown menu also includes options for 'Open with Codespaces', 'Open with GitHub Desktop', and 'Download ZIP'. The repository content shows a README.md file with the text 'demo2'.

Ik maak gebruik van **GitHub Desktop** om de repository lokaal te klonen. Open GitHub desktop en voer de nodige configuratie info in zoals inloggen in GitHub. Als dit gelukt is kunnen we de repository kiezen uit het lijstje en kiezen we onderaan **Clone ....**



Daarna moeten we nog opgeven naar waar we deze willen klonen op onze PC. Ik kies voor **C:\src**.



De lege map zal aangemaakt worden. Maak nu een nieuw web api project aan zoals we reeds gezien hebben in deze lege map.

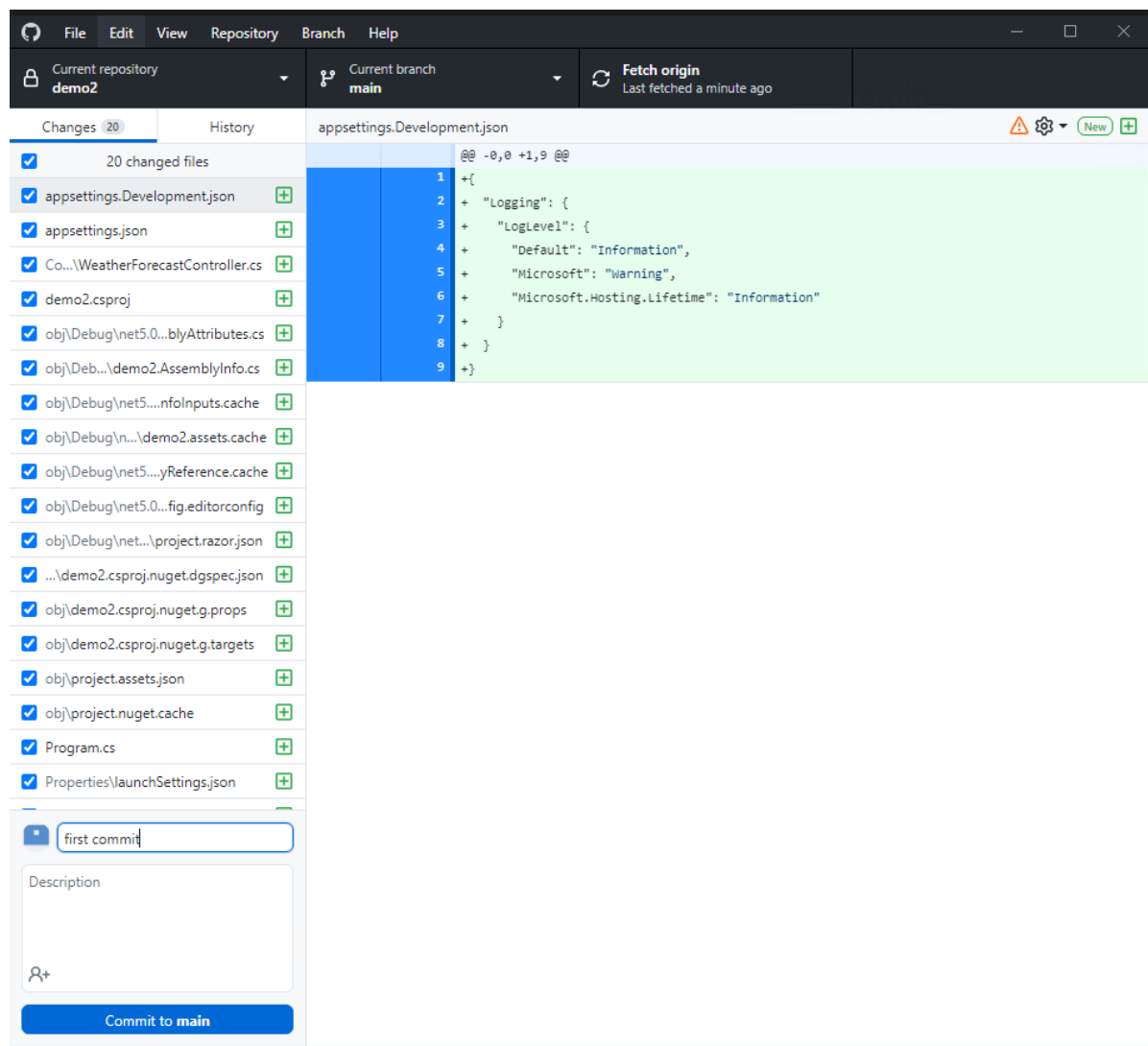
```
C:\src\demo2>dotnet new webapi
The template "ASP.NET Core Web API" was created successfully.

Processing post-creation actions...
Running 'dotnet restore' on C:\src\demo2\demo2.csproj...
  Determining projects to restore...
  Restored C:\src\demo2\demo2.csproj (in 219 ms).
Restore succeeded.

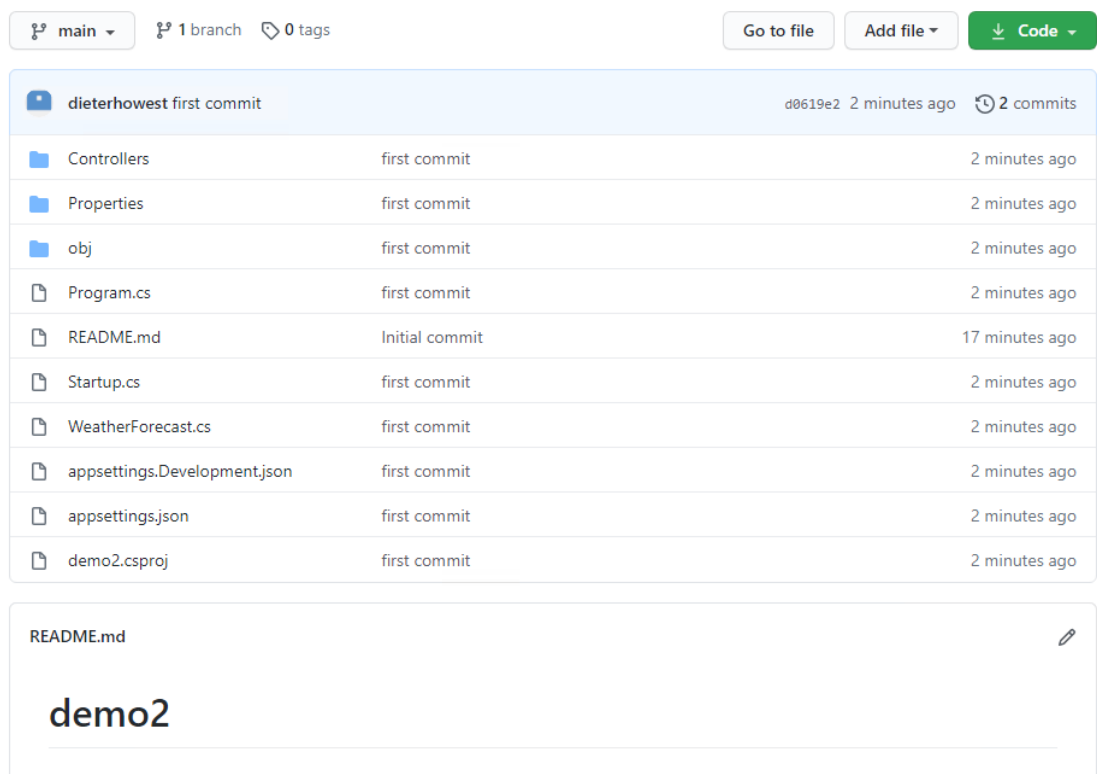
C:\src\demo2>
```

Als we nu kijken naar **GitHub** desktop dan zien we daar alle veranderingen. Onderaan geven we een **commit message** in bv.: **first commit**. Daarna drukken we op **Commit to main**.



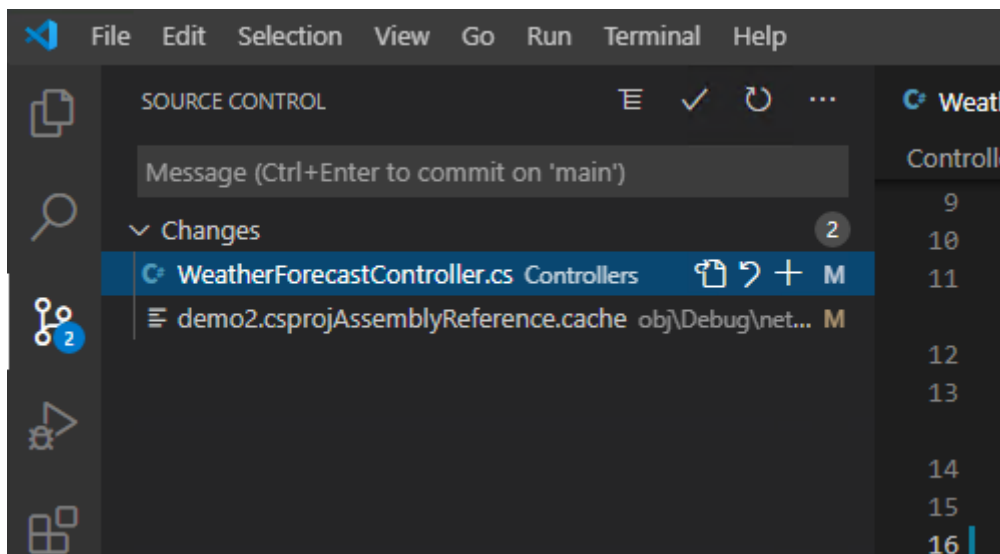


Zoals je reeds weet gaan we via een **Commit** de files eerst lokaal toevoegen. We moeten deze nu nog **pushen** naar **GitHub**. Kies hiervoor **Push origin** in GitHub desktop. Als je nu gaat kijken naar GitHub zouden de files online moeten staan in de repository:

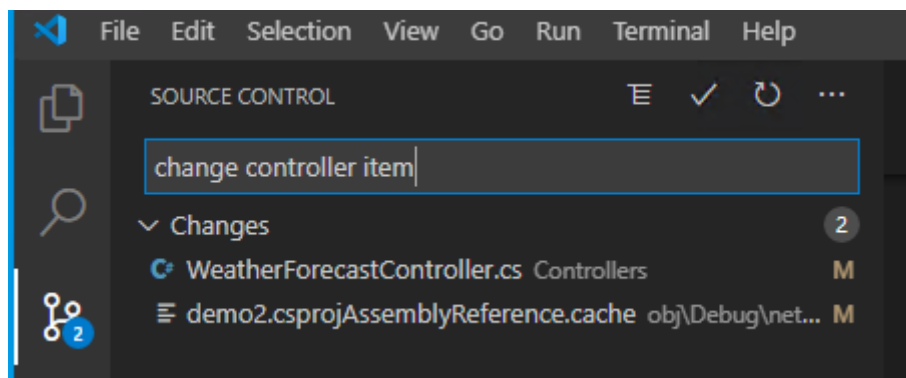


# GitHub en Visual Studio Code

Het is ook mogelijk via **Visual Studio Code** commits te doen naar GitHub. Via het derde icoon rechts zie je eventuele veranderingen in de files.



Bovenaan bij **Message** kan je dan een bericht tikken en op het ✓ icoon drukken.



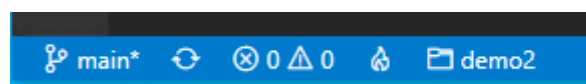
Daarna krijg je onderstaande melding en kies je **Yes**.



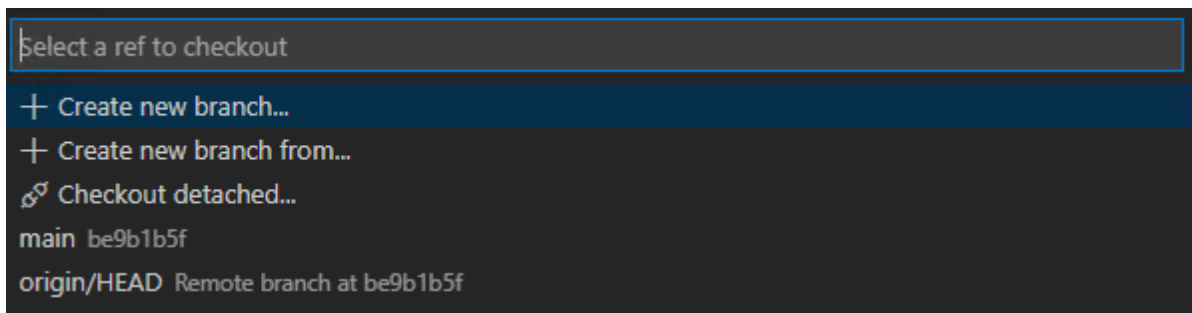
Als laatste moeten we dit nog pushen naar GitHub. We drukken op de ... waardoor we een menu te zien krijgen en kiezen voor **Push**. U kan de vraag krijgen om in te loggen in GitHub, u mag dit doen. Daarna zal de **Push** plaatsvinden.

## Branching & merging

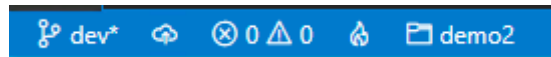
Tot nu toe werken we steeds op de **main** branch wat eigenlijk niet ok is. We maken best een **dev** branch of **development** branch aan. We kunnen dit via **GitHub** aanmaken maar ook via **Visual Studio Code**. Druk onderaan op **main**.



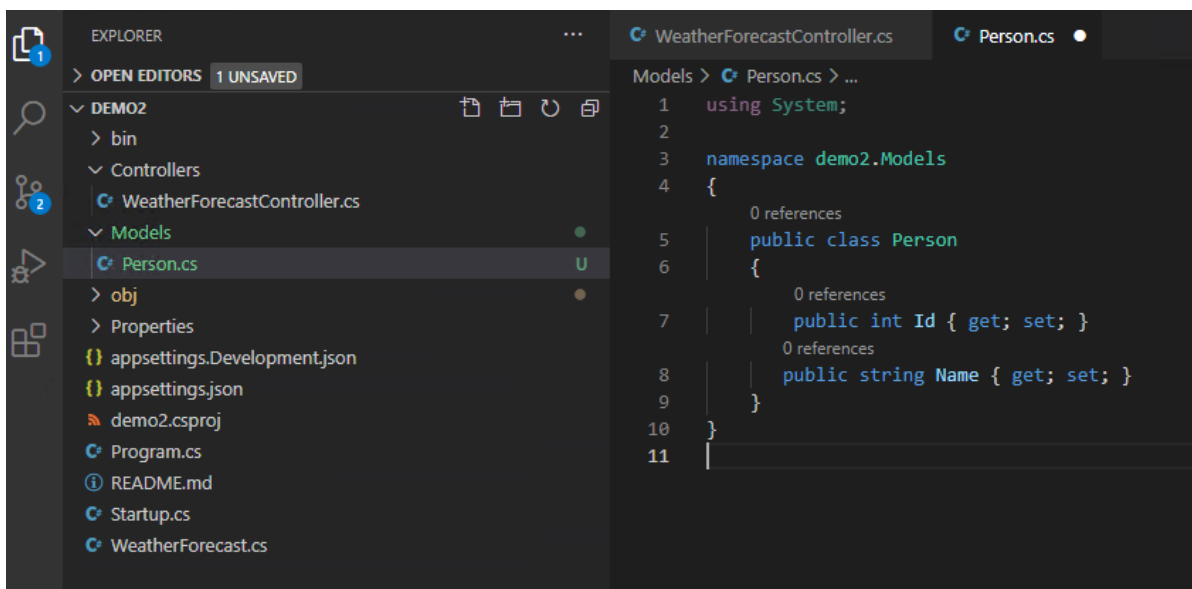
Bovenaan zal volgende verschijnen:



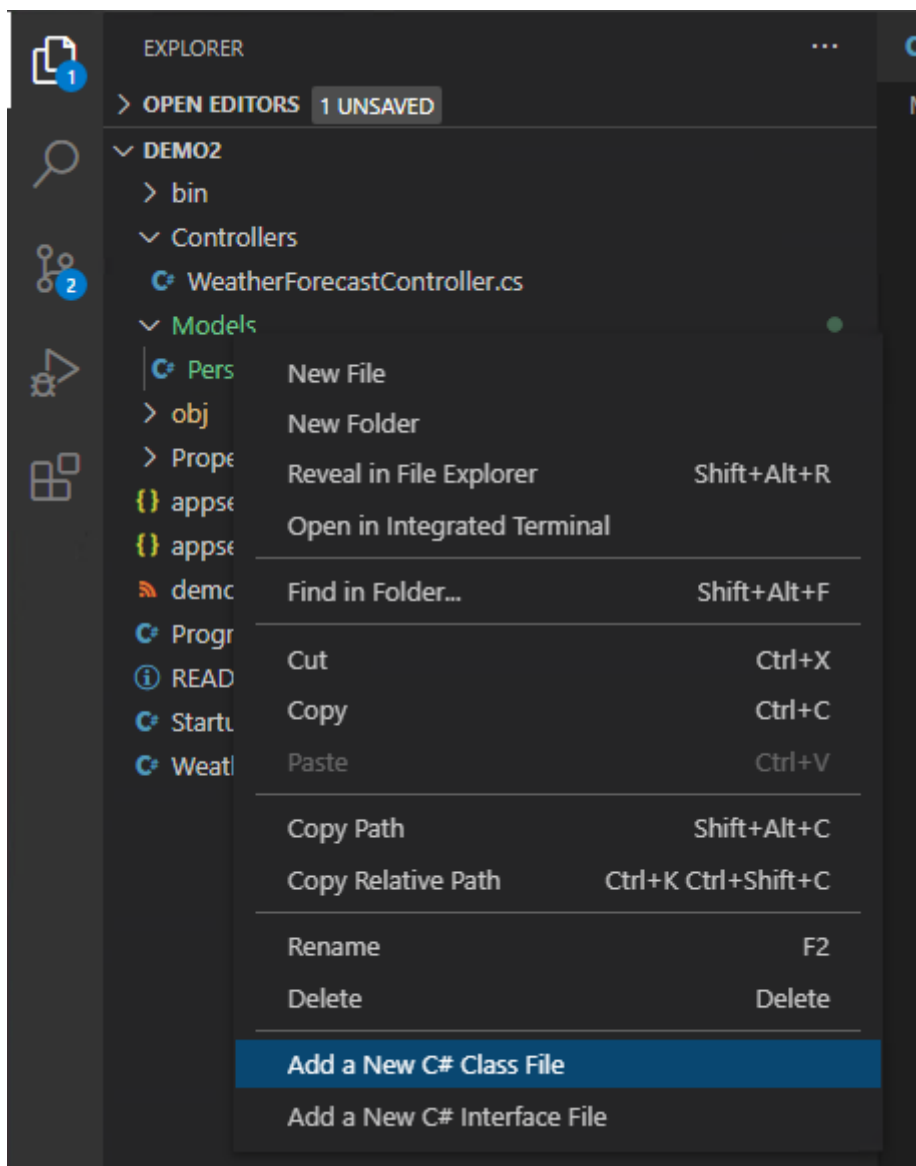
Kies nu voor **+ Create new branch from...**. Daarna krijg je de vraag om een naam in te vullen, bv.: **dev**. In het volgende scherm kies je de **branch** die je wenst te gebruiken als basis. Kies hier voor **main**. Daarna zal de **dev** branch aangemaakt worden en als default ingesteld worden.



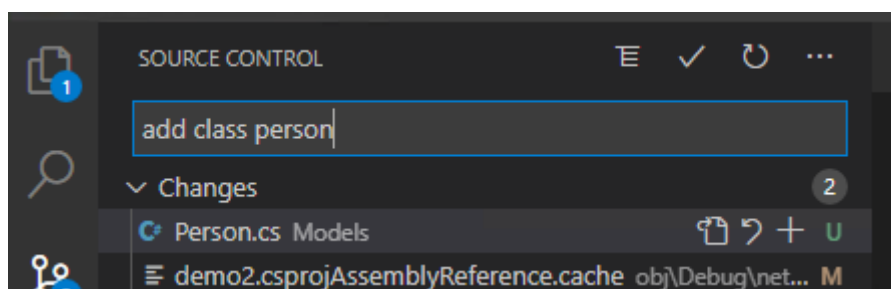
Voeg nu eens een folder **Models** toe aan het project en een klasse **Person** met daarin twee properties




Een klasse kan je snel als volgt toevoegen (mits je VS Sharper extensie hebt staan):



**Commit** nu alles naar de **dev branch lokaal**. **push** de branch naar GitHub



Als we nu gaan kijken in **GitHub** zien we de nieuwe **branch** ook staan:

 dev had recent pushes 3 minutes ago

Compare & pull request

dev 2 branches 0 tags

Go to fileAdd fileCode

Switch branches/tags

Find or create a branch...

BranchesTags

main default

✓ dev

View all branches

4edcde6 4 minutes ago 4 commits

change controller item	19 minutes ago
add class person	4 minutes ago
first commit	38 minutes ago
add class person	4 minutes ago
first commit	38 minutes ago
Initial commit	1 hour ago
first commit	38 minutes ago
first commit	38 minutes ago
first commit	38 minutes ago
first commit	38 minutes ago
first commit	38 minutes ago
first commit	38 minutes ago

Properties

obj

Program.cs

README.md

Startup.cs

WeatherForecast.cs

appsettings.Development.json

appsettings.json

demo2.csproj

README.md

demo2

We zien in **GitHub** ook de melding dat we een **pull request** kunnen doen. Dit laat ons toe om de **dev** branch te integreren in de **main** branch. Klik op **compare & pull request** in GitHub. Daarna krijg je een scherm waarin je commentaar kan geven en de veranderingen ziet. Kies hier voor **Create pull request**.

## Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

base: main ← compare: dev ✓ Able to merge. These branches can be automatically merged.

add class person

Write

Preview

H

B

I

≡

<>

🔗

⋮

⋮

☑

@

📎

↶

alles ok

New

 Video support! Upload MP4 and MOV file types. Attach files by dragging & dropping, selecting or pasting them.

Create pull request

Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

Reviewers

No reviews

Assignees

No one—assign yourself

Labels

None yet

Projects

None yet

Milestone

No milestone

Linked issues

Use [Closing keywords](#) in the description to automatically close issues

Helpful resources

[GitHub Community Guidelines](#)

1 commit

2 files changed

0 comments

1 contributor

Commits on Dec 28, 2020

add class person

4edcde6

changed files with 10 additions and 0 deletions.

Unify

Models/Person.cs

+1,10 @@

```
1 + using System;
2 +
3 + namespace demo2.Models
4 + {
5 +     public class Person
6 +     {
7 +         public int Id { get; set; }
8 +         public string Name { get; set; }
9 +     }
10 + }
```

Nu krijgen we onderstaande scherm, de pull request is goedgekeurd nu kunnen we de merge doen. We kiezen **Merge pull request** en kiezen dan **Confirm merge**.

## add class person #1

Open dieterhowest wants to merge 1 commit into `main` from `dev`

Conversation 0 Commits 1 Checks 0 Files changed 2

dieterhowest commented now

alles ok

add class person 4edcde6

Add more commits by pushing to the `dev` branch on dieterhowest/demo2.

This branch has no conflicts with the base branch  
Merging can be performed automatically.

Merge pull request You can also open this in [GitHub Desktop](#) or view [command line instructions](#).

Write Preview

Leave a comment

Video support! Upload MP4 and MOV file types. Attach files by dragging & dropping, selecting or pasting them.

Close pull request Comment

Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

**ProTip!** Add comments to specific lines under [Files changed](#).

Als we nu gaan kijken in GitHub naar onze **main** branch dan zien we dat de map **Models** nu ook aanwezig is met de klasse **Person**. We hebben dus de aanpassingen in de **dev** branch ook ingevoegd in de **main** branch.

**Probeer in projecten NOOIT op de main branch te werken, altijd op een dev branch of feature branch.**

Bovenstaande kan u ook doen via de onderstaande Visual Studio Code Extension indien u dit wenst.

The screenshot shows the Visual Studio Code interface with the Extensions Marketplace open. The search bar contains 'GitHub Pull Requests and Issues'. The extension is listed as 'GitHub Pull Requests and Issues' by 'GitHub', with version 'v0.22.0' and a rating of 3.5 stars. It is marked as 'Recommended' and 'This extension is recommended because of the current workspace configuration'. The extension is currently installed, and the 'Uninstall' button is visible. The 'Details' tab is selected, showing the extension's description: 'Pull Request and Issue Provider for GitHub'. The 'Azure Pipelines' status is shown as 'succeeded'.

## Oefening 1

In deze oefening gaan we een eenvoudig API ontwikkelen waarmee volgende zaken mogelijk moeten zijn:

- Toevoegen van een nieuwe wijn
- Verwijderen van een wijn

- Alle wijnen terugkeren
- Alle wijnen van een bepaald kleur terugkeren
- Een wijn wijzigen

Een wijn bestaat uit volgende eigenschappen:

- naam (name)
- jaartal (year)
- kleur (color)
- prijs (price)
- land (country)
- druiven (grapes)

Maak nu zelf eerst een nieuwe **Github** repository aan bv.: **backend-labo01-wijn**. Clone deze lokaal en open in Visual Studio Code (of een andere editor naar keuze).

Eerste wat we gaan doen is het opstellen van ons **Model**. Het model zal beschrijven hoe een wijn object er zal uitzien. Dit komt één op één overeen met de **JSON** die we zullen versturen. Maak een **folder Models** in het project en voeg een **klasse Wine** toe.

```
public class Wine
{
    public int WineId { get; set; }
    public string Name { get; set; }
    public int Year { get; set; }
    public string Country { get; set; }
    public string Color { get; set; }
    public decimal Price { get; set; }
    public string Grapes { get; set; }
}
```

Zoals we reeds weten zal alle communicatie gebeuren via een **Controller**. Een **Controller** is het entry point van onze API. Voeg een nieuwe **Controller** toe met als naam **WineController**. Dit doen we door een nieuwe **klasse** toe te voegen aan het project. De naam is belangrijk en **moet** eindigen op **Controller**.



```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using backend_labo01_wijn.Controllers.Models;

namespace backend_labo01_wijn.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class WineController : ControllerBase
    {
        private readonly ILogger<WineController> _logger;

        public WineController(ILogger<WineController> logger)
        {
            _logger = logger;
        }
    }
}

```

Enkel belangrijke aspecten zijn:

- [ApiController] attribuut zal aanduiden dat het om een API controller gaat maar is niet verplicht als je een route op een endpoint gaat plaatsen
- [Route] attribuut zal de route aanduiden van de API (de url). We vullen hier standaard [Controller] in, dit wil zeggen dat we de naam van de controller zullen gebruiken. Je mag dit weglaten als je route op endpoint gaat plaatsen
- De klasse **moet** erven van **ControllerBase**
- We voegen ook een **Constructor** toe met dezelfde naam als de klasse (dit moet zo in C#) en injecteren een ILogger. Op deze manier kunnen we vrij eenvoudig logging toevoegen. We moeten hiervoor ook een private field declareren van hetzelfde type

Aangezien we nog geen databases gezien hebben gaan we een **static** lijst gebruiken om de data bij te houden. We gaan deze aanmaken in de **constructor** van de controller. We maken deze **static** zodat deze zal blijven bestaan gedurende de tijd onze applicatie actief is. Wijzig zoals in onderstaande screenshot:

```

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

    private readonly static List<Wine> _wines = new List<Wine>();
    private readonly ILogger<WineController> _logger;

    public WineController(ILogger<WineController> logger)
    {
        _logger = logger;
        if(_wines == null || _wines.Count() == 0){
            _wines.Add(new Wine(){
                WineId = 1,
                Name = "Sangrato Barolo",
                Country = "Italie",
                Price = 35,
                Color = "red",
                Year = 2005,
                Grapes = "Nebiollo"
            });
        }
        _logger.LogInformation("ctor");
    }
}

```

Nu kunnen we een eerste **endpoint** toevoegen om alle wijnen op te halen:

```

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

```

Test dit nu uit in **Postman**.

- wat zal de **route** zijn die je moet gebruiken ?
- welke statuscode krijgen we terug in Postman ?

Voeg nu een tweede **endpoint** toe om een nieuwe wijn toe te voegen, een **HTTP POST**.

```

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

```

Test dit nu uit in **Postman**. Aangezien we een nieuwe wijn wensen toe te voegen moeten we deze meesturen in de **body** van het HTTP Request. We moeten dus een instantie van de klasse **Wine** meesturen in JSON formaat. Je kan dit omzetten via [C# to JSON Converter: Convert C# classes to JSON \(csharp2json.io\)](https://csharp2json.io/). Ik heb dit reeds voor jullie gedaan met, je kan dit als JSON gebruiken:

```
{
  "name": "Muller Kogl",
  "year": 2019,
  "country": "Oostenrijk",
  "color": "white",
  "price": 12.5,
  "grapes": "Gruner Veltliner"
}
```

- wat zal de **route** zijn die je moet gebruiken ?
- welke statuscode krijgen we terug in Postman ?

Probeer nu ook het vorige endpoint uit, de **HTTP GET** en kijk of je nu twee wijnen als resultaat krijgt.

Het volgende **endpoint** is een **HTTP DELETE**. De eerste optie is een wijn doorsturen in de body die we wensten verwijderen. De code ziet er als volgt uit:

```
[HttpDelete]
public Wine RemoveWine(Wine wine){
    Wine _wine = _wines.Find(delegate(Wine w){
        return w.WineId == wine.WineId;
    });
    _wines.Remove(_wine);
    return wine;
}
```

Bovenstaande code krijgt een **Wine** object binnen in de body. We moeten dit zoeken in de lijst. We doen dit via een **anonieme** functie(**delegate**). Dit is een functie zonder naam die we lokaal definiëren en meegeven aan de **Find**. Deze functie zal het gevonden **Wine** object terugkeren zodat we dit via **Remove** kunnen verwijderen. We keren ook het verwijderde object terug naar Postman. Test dit uit. Vergeet niet dat in je Postman JSON in de body moet plaatsen met het **WineId** ingevuld. vb:

```
{
  "wineId" : 2,
  "name": "Muller Kogl",
  "year": 2019,
  "country": "Oostenrijk",
  "color": "white",
  "price": 12.5,
  "grapes": "Gruner Veltliner"
}
```

Stel dat we nu een fout WineId meegeven dat we niet kunnen vinden dan moeten we dit opvangen en een andere status code terugsturen. Pas de code als volgt aan zodat we bij het niet vinden van de wijn een 404 terugsturen.

```

[HttpDelete]
public ActionResult RemoveWine(Wine wine){
    Wine _wine = _wines.Find(delegate(Wine w){
        return w.WineId == wine.WineId;
    });

    if(_wine != null){
        _wines.Remove(_wine);
        return new OkObjectResult(wine);
    }
    else{
        return new StatusCodeResult(404);
    }
}

```

Het laatste **endpoint** die we toevoegen is een update **HTTP PUT**. We wensen een wijn te updaten in de lijst. Voeg onderstaande code toe en test uit via **Postman**. We zoeken eerst de wijn die we wensen te wijzigen. Indien we deze vinden wijzigen we de property **Name**. Je mag natuurlijk zelf ook extra properties toevoegen die we wil wijzigen.

```

[HttpPut]
public ActionResult UpdateWine(Wine wine){
    Wine _wine = _wines.Find(delegate(Wine w){
        return w.WineId == wine.WineId;
    });

    if(_wine != null){
        _wine.Name = wine.Name;
        return new OkObjectResult(wine);
    }
    else{
        return new StatusCodeResult(404);
    }
}

```

Wijzig nu de route zodat alles via **wines** zal verlopen en test uit of alles nog werkt via **Postman**.

```

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

```

We gaan nu nog een tweede manier toevoegen om een wijn te verwijderen. We kunnen ook gewoon het **WineId** meegeven. Voeg onderstaande endpoint toe. Merkt op dat we nu ook een route toevoegen met daarin de parameter **WineId**. Deze parameter moeten we ook voorzien in onze C# functie **RemoveWineById**.

```

[HttpDelete]
[Route("wine/{wineId}")]
public ActionResult RemoveWineById(int wineId){
    Wine _wine = _wines.Find(delegate(Wine w){
        return w.WineId == wineId;
    });

    if(_wine != null){
        _wines.Remove(_wine);
        return new OkObjectResult(_wine);
    }
    else{
        return new StatusCodeResult(404);
    }
}

```

Voeg nu zelf een **endpoint** toe om één wijn op te vragen. Je zal dus een **WineId** moeten meegeven als parameter. Als de wijn niet gevonden is met je een **NotFound** terugkeren. Probeer dit nu zelf te schrijven.

bekijk nu ook de Swagger documentatie door achter de URL /swagger te tikken, vb: <https://localhost:5001/swagger/>

## Oefening 2

Deze oefening moet u zelfstandig maken op basis van onderstaande omschrijving.

We hebben een **Brand** en **CarModel** klasse. Een Brand bestaat uit een **BrandId**, **Name**, **Country** en Logo. Een **CarModel** bestaat uit een **CarModelId**, **Name** en **Brand**. We wensten een API te bouwen die volgende **endpoints** moet aanbieden

- alle brands ophalen
- alle brands van een country ophalen
- een brand toevoegen
- één specifiek brand ophalen
- alle carmodels ophalen met daarin hun brand
- alle carmodels van een brand ophalen
- één specifiek carmodel ophalen
- alle carmodels van een land ophalen