# Inrichting Ontwikkelomgeving

Documentatie

Anthony Delgado, Ayoub Lfatmi, Per Taffijn en Mustafa Warsame

Datum:

*11 december 2023*

Deskundige docenten(en):

*H. Middelkoop en P. van den Bosch*

PDP-coach(es):

*M. Dijkstra*

# **Inhoudsopgave**

**Inleiding** [**1**](https://docs.google.com/document/d/1LXsYfToCRYOSGY_xY4buPkIz8oluGb6cwdAqAyQoIs4/edit#heading=h.p6m8fms7pvn9)

[**Inhoudsopgave**](#_gyri42sqr2a) **x**

**Ontwikkelomgeving x**

**Code Conventies****x**

Benamingenx

*PascalCase**x*

*camelCase**x*

Commentaarx

**Structuur x**

[**Bronnen**](https://docs.google.com/document/d/1LXsYfToCRYOSGY_xY4buPkIz8oluGb6cwdAqAyQoIs4/edit#heading=h.1y3aaxplqyyx) **x**

# **Inleiding**

Week 15 markeert het spannende begin van de praktische fase waarin we als team voorbereidingen hebben getroffen om de visie van Stichting Accessibility om te kunnen zetten in een tastbare webapplicatie. De afgelopen dagen stonden in het teken van cruciale beslissingen op het gebied van technologische keuzes, het vaststellen van code conventies, het implementeren van versiebeheer, en het opzetten van een robuuste CI/CD pipeline. Daarnaast hebben we ook gekeken naar de voorkeuren van elk teamlid voor het gebruik van een ‘Integrated Development Environment’, kortweg IDE genoemd. Dit verslag belicht in detail onze overwegingen en activiteiten tijdens deze belangrijke voorbereidingsweek.

# **Ontwikkelomgeving**

Een eenvoduige en goed geïntegreerde ontwikkelomgeving (IDE) is van onschatbare waarde bij het efficiënt ontwikkelen van code. Voor ons project, waarin we zowel Csharp voor de backend als React voor de frontend zullen gebruiken, hebben we besloten om Visual Studio 2022 (Community Edition) als onze primaire IDE te gebruiken. Deze keuze is gebaseerd op verschillende functionaliteiten die Visual Studio biedt, specifiek afgestemd op C# en React.

### Voordelen van Visual Studio 2022 (Community Edition)

**Uitgbreide ondersteuning voor C#-ontwikkeling**

Visual Studio biedt een uitgbreide set tools en functies die specifiek zijn ontworpen voor C#- ontwikkeling. Dit omvat functies zoals debuggmogelijkheden, automatische codeaanvulling, refactoring-tools, automatisch gegenereerde codetemplates en integratie met versiebeheersystemen zoals Git. Voor meer informatie over Git kunt u de website [via deze link](https://git-scm.com/) raadplegen.

**Integratie met ASP.NET Core**

Visual Studio integreert naadloos met ASP.NET Core, wat essentieel is voor ons project waarin we de backend zullen ontwikkelen door gebruik te maken van de functionaliteiten die dit framework biedt. Zo kun je met een aantal knoppen al een ASP.NET Core-project opzetten.

**Ondersteuning voor React en Javascript**

Visual Studio biedt ook uitstekende ondersteuning voor het ontwikkelen van frontend-applicaties met React en JavaScript. Het bevat geavanceerde JavaScript-tools, JSX-ondersteuning en integratie met populaire frontend-frameworks.

**Integratie van CI/CD-tools:**

Visual Studio kan worden geïntegreerd met verschillende Continuous Integration (CI) en Continuous Deployment (CD) tools, waaronder Azure DevOps. Dit vergemakkelijkt de implementatie van CI/CD-pipelines voor automatische bouw- en implementatieprocessen. Dit kan ons uiteindelijk helpen om het project makkelijk en zonder te veel moeite te kunnen deployen.

Kortom, Visual Studio biedt een uitgebreide ontwikkelomgeving die ideaal is voor ons project, waarin we werken met C# en React. Het stelt ons in staat om efficiënt te ontwikkelen, samen te werken en de kwaliteit van onze code te waarborgen. Voor extra informatie over Visual Studio in combinatie met C# kunt u de volgende bron raadplegen: [Get started with Visual Studio](https://learn.microsoft.com/en-us/visualstudio/get-started/csharp/visual-studio-ide?view=vs-2022)

# 

# 

# **Code Conventies**

Een uniforme codebase is essentieel voor een gestroomlijnde samenwerking binnen het team. Daarom hebben we uitvoerig stilgestaan bij het opstellen van code conventies. Deze richtlijnen, variërend van naamgeving van variabelen tot commentaarstijlen, zijn niet willekeurig gekozen. We hebben inspiratie gehaald uit gerenommeerde bronnen, zoals de [Microsoft's C# Coding Conventions](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions), om er zeker van te zijn dat onze code leesbaar, consistent en van hoge kwaliteit is.

Hieronder worden uitgebreide code conventies verder beschreven:

### Benamingen

Bij het kiezen van namen voor klassen, methode, eigenschappen of constanten, volgen we een strikte maar flexible benadering. We maken gebruik van PascalCase; deze conventie wordt toegepast op klassen, methoden, eigenschappen, en constanten. Het gebruik van PascalCase bevordert duidelijkheid en consistentie in de codebase. Voorbeeld: PublicMethod, ClassName, Constants. Daarnaast hanteren we de regel om betekenisvolle namen te gebruiken die de functie en betekenis van de variabele, methode of klasse duidelijk weergeeft.

Hieronder hebben we een voorbeeld van het gebruik van PascalCase:

| public class UserController {  public void PublicMethod()  {  *// Implementatie voor de PublicMethod() methode*  }   private string PrivateProperty { get; set; } } |
| --- |

Voor lokale variabelen en parameters binnen functies maken we gebruik van camelCase. Dit wordt gebruikt om de componenten, zoals parameters en variabelen, te onderscheiden van klassen en methoden. Bijvoorbeeld: localVariable, parameterName.

Hieronder staat een voorbeeld waarbij gebruik wordt gemaakt van camelCase:

| public void ProcessData(string inputParameter) *// parameter in camelCase* {  string localVariable = inputParameter;   *// Code hier* } |
| --- |

### 

### Commentaar

Goed geplaatst commentaar is van onschatbare waarde voor ontwikkelaars om de code beter te begrijpen. Het is belangrijk om frequent en doelgericht commentaar toe te voegen om codesmells te voorkomen. Het plaatsen van commentaar in de geschreven code zorgt er ook voor dat mede-ontwikkelaars de functionaliteit van de geschreven code eenvoudiger kunnen begrijpen.

**In-line commentaar:**

Gebruik van ‘//’ voor in-line opmerkingen. Bijvoorbeeld:

| const userServiceComponent = () => {  const getUserDetails = (userId) => {  *// Roep backend API aan voor gebruikersdetails*  *// en verwerk de respons.*  *// Uitgebreide code hier*  }; }; |
| --- |

| public class UserService {  public void GetUserDetails(int userId)  {  *// Haal gebruikersdetails op basis van de meegegeven userId.*  *// Uitgebreide code hier*  } |
| --- |

**Blokcommentaar:**

Gebruik ‘/\*......\*/’ voor blokopmerkingen. Gebruik deze voor een overzicht van klassen, methoden of andere complexe functionaliteiten.

| */\*  UserServiceComponent.js  Component voor het beheren van gebruikersgerelateerde functionaliteiten. \*/* const userServiceComponent = () => {  *// Uitgebreide code hier* }; |
| --- |

Commentaar moet niet alleen de werking van de code uitleggen, maar ook informatie verschaffen over de reden achter bepaalde beslissingen of complexe stukken code. Zorg ervoor dat het commentaar up-to-date wordt gehouden en zich aanpast aan eventuele wijzigingen in de code.

# 

# **Structuur**

Een gestructureerde codebase begint natuurlijk met een logische en consistente projectstructuur. We organiseren namespaces, packages, klassen en andere functionele componenten logisch om niet alleen de leesbaarheid van de code te waarborgen, maar ook om de vindbaarheid van gerelateerde functionaliteiten te vergroten.

### Namespaces, klassen en andere componenten

Een overzichtelijke organisatie van namespaces en klassen helpt bij het logisch groeperen van gerelateerde functionaliteiten binnen een webapplicatie. Denk hierbij bijvoorbeeld aan het groeperen van alle functionele componenten voor de user-registratie binnen een bepaalde namespace. Het gebruik van hiërarchieën binnen namespaces draagt bij aan een begrijpelijke en overzichtelijke codebase.

Hieronder hebben we een kort voorbeeld in code, waarin een aantal klassen die gerelateerd aan elkaar zijn, zijn ondergebracht binnen een gezamelijke namespace:

| namespace WebApplication.Services.User  {  public class UserService  {  *// Code implementatie hier*  }    public class ProfileService  {  *// Code implementatie hier*  } } |
| --- |

### 

### 

### 

### 

### 

### 

### Logische mapstructuur

Een goed gestructureerde mapstructuur (of bestandsindeling) is cruciaal voor het begrijpen, onderhouden en schalen van de webapplicatie. Dit geldt vooral wanneer het een project is met meerdere componenten, zoals een ASP.NET core backend en een React voor de frontend. Hieronder wordt een globale bestandsindelingconventie voor ons project opgesteld:

| /WebApplication  /App  /public  /src  App.cs  App.js  ...  Index.js  /components  AppHeader.css  AppHeader.js  ...  /services  Authentication.js  Api.s   ... /Controllers  LoginController.cs  UserController.cs  IdentityController.cs  ...  /utilities  /wwwroot  /Services  /User  UserService.cs  /Profile  ProfileService.cs  /Models  UserModel.cs  ProfileModel.cs  /Repositories  UserRepository.cs  /Utilities  /Helpers  ValidationHelper.cs  /Properties  launchSettings.json |
| --- |

Deze gedetailleerde bestandsindeling voor zowel de backend als de frontend van de applicatie bevordert de leesbaarheid, onderhoudbaarheid en samenwerking binnen ons ontwikkelteam. Het stelt ons in staat snel relevante bestanden te vinden en vermindert de complexiteit bij het begrijpen van de applicatiestructuur. De ***./src/app*** is het deel van het project waarin de react-componenten staan voor de frontend. Binnen de ***components-folder*** is het verder ook nog handig om te specificeren om welke soort component het gaat, denk hierbij bijvoorbeeld aan de footer, inlogpagina etc. Dit zullen we dan ook nog verder specificeren met behulp van directories zodra dat nodig is.

Verder onderaan hebben we de benodigde directories voor de [ASP.NET core](https://dotnet.microsoft.com/en-us/apps/aspnet) backend. Dit wordt echter al voor een groot deel automatisch aangemaakt door het ASP.NET core framework wanneer je een Web API project aanmaakt. Dit kan eenvoudig via de userinterface in [Visual Studio](https://visualstudio.microsoft.com/) of in de terminal met de volgende commands:

| dotnet new webapi -n {JouwWebApiProjectNaam}  *// Benodigde packages* dotnet add package Microsoft.EntityFrameworkCore dotnet add package Microsoft.EntityFrameworkCore.Design dotnet add package Microsoft.EntityFrameworkCore.Tools dotnet add package Microsoft.EntityFrameworkCore.SqlServer |
| --- |

Het laatste command is variabel aangezien er gebruikgemaakt kan worden van diverse soorten databases, zoals InMemorydatabases, Sqlite etc.

# **Bronnen**

Anandmeg. (2023, 10 maart). *Overview for C# developers - Visual Studio (Windows)*.

Microsoft Learn. <https://learn.microsoft.com/en-us/visualstudio/get-started/csharp/visual-studio-ide?view=vs-2022>

Swain, C. (z.d.). *Folder structure of ASP.NET Core MVC 6.0 project*. <https://www.c-sharpcorner.com/article/folder-structure-of-asp-net-core-mvc-6-0-project/>

Mir, M. A. (2023, 2 april). What are the advantages and disadvantages of using visual Studio code or Atom? *Medium*. <https://medium.com/@ssc.ahmed.926748/what-are-the-advantages-and-disadvantages-of-using-visual-studio-code-or-atom-d3132bf1af85>

Rick-Anderson. (2023, 24 augustus). *Getting started - EF Core*. Microsoft Learn. <https://learn.microsoft.com/en-us/ef/core/get-started/overview/first-app?tabs=netcore-cli>

*Getting started: .NET core with command line*. (z.d.). xUnit.net. <https://xunit.net/docs/getting-started/netcore/cmdline>

Wadepickett. (2023, November 1). *Publish an ASP.NET Core app to Azure with Visual Studio*. Microsoft Learn. <https://learn.microsoft.com/en-us/aspnet/core/tutorials/publish-to-azure-webapp-using-vs?view=aspnetcore-7.0>