

Overdrachtsdocument

ACI Rental System
Fontys Hogeschool | Tilburg

Auteur: Job Verwiël, Siebren Kraak, Marco Ketelaars, Myron Antonissen, Angelo Bruggemans
Locatie: Tilburg
Versie: 1.0
Datum: 15 Maa 2021
Aangepast door: Daan Brouwer, Cyrion van Dongen, Michael Bednarek, Sam Volkers, Bart Pelsma, Xander Pluimert

1 Versiebeheer

Versie	Wijziging	Datum
0.1	Opzet document	15-03-2021
0.2	'Proxy' verwijderd, virusscanner info toegevoegd bij front-end unit testing	23-03-2021
0.3	CORS toegevoegd, tussenkopjes geüpdate met cijfers.	24-03-2021
1.0	Verwijderen mogelijke privé gegevens	01-06-2021
1.1	Aanpassingen door S2	29-09-2021

2 Inhoud

1	Versiebeheer	2
2	Inhoud	3
3	Inleiding	5
4	Angular	6
4.1	Keuze	6
4.2	Opzet	6
4.2.1	Prerequisites	6
4.3	Componenten	6
4.3.1	Package.json	6
4.3.2	Material module	6
4.3.3	app.module.ts	7
4.3.4	app.component.ts	7
4.3.5	Assets folder	8
4.3.6	Routing	8
4.3.7	Api service	8
4.3.8	Component	8
4.3.9	Testen	9
4.3.10	AuthGuard	9
4.4	Beheerbaar	9
4.5	Uitbreiden	9
5	.NET core	11
5.1	Keuze	11
5.2	Opzet	12
5.2.1	Database opzet	12
5.2.2	Migrations	15
5.2.3	Omgevingsvariabelen	15
5.2.4	Visual studio opzet	16
5.2.5	Nieuwe microservice	Fout! Bladwijzer niet gedefinieerd.
5.2.6	Cors	26
6	CI/CD	27
6.1	Inleiding	27
6.2	Keuze	27
6.3	Opzetten GitHub Actions	27
6.4	Yml file opbouwen	28

6.4.1	On	28
6.4.2	Env	29
6.4.3	Jobs.....	30
6.5	Opzetten SonarCloud	31
6.6	Pushen naar DockerHub	36
6.6.1	Dockerfile.....	36
6.6.2	DockerHub.....	37
7	Deployment op NetLab met een extern IP-adres.....	38
7.1	Het aanvragen van een PROF-account	38
7.2	Opzetten van een VM.....	38
7.3	YAML-bestanden	39
7.3.1	Secrets	39
7.3.2	Services.....	39
7.3.3	Ingress	40

3 Inleiding

Binnen de opleiding ACI is het mogelijk om apparatuur te lenen uit de catalogus die zij hebben. Op dit moment is er een website om dit mogelijk te maken voor leerlingen echter loopt deze website niet in lijn met wat de medewerkers en leerlingen van ACI willen. Daarom wordt er een nieuwe applicatie gemaakt die dit ook mogelijk maakt voor leerlingen en leraren. In deze nieuwe applicatie zijn bepaalde functionaliteiten uit de oude applicatie weggelaten en zijn nieuwe functionaliteiten toegevoegd.

Omdat dit project uitgewerkt gaat worden door meerdere projectgroepen is het belangrijk dat het project overdraagbaar is. Om deze overdraagbaarheid te bevorderen wordt dit document opgesteld. In dit document wordt van de gebruikte services/omgevingen uitgelegd: waarom ze gekozen zijn, hoe ze opgezet worden, hoe ze beheerd worden en hoe uitgebreid kan worden.

4 Angular

Angular is een front-end framework gebaseerd op TypeScript.

4.1 Keuze

Voor het project is er onderzoek gedaan naar verschillende front-end talen en frameworks.

Uiteindelijk is gekozen voor Angular met de gedachten dat de initiële opzet lastig is om te doen, maar het daarna laagdrempelig is om te gebruiken. Het eerste team doet de opzet waardoor het voor de opvolgende teams met weinig ervaring toch makkelijk is om op te pakken.

4.2 Opzet

In dit hoofdstuk zal niet worden uitgelegd hoe een Angular project stap voor stap wordt opgezet. Er wordt uitgelegd wat het Angular project bevat en wat nodig is om gebruik te maken van dit Angular project.

4.2.1 Prerequisites

Om gebruik te kunnen maken van Angular zijn een aantal dingen nodig. Het eerste is NodeJS. NodeJS zorgt ervoor dat gebruik gemaakt kan worden van de NPM commands. NPM is een package manager waarmee onder andere ook Angular geïnstalleerd kan worden. NodeJS kan [hier](#) worden gedownload en dat is een .exe die het automatisch ook in het %PATH% zet.

Na het installeren van NodeJS kan Angular geïnstalleerd worden. Dat kan gedaan worden door in een terminal de command "npm install -g @angular/cli" te runnen. Zodra dit gelukt is, is het mogelijk om bijvoorbeeld een nieuwe Angular app te maken.

4.3 Componenten

4.3.1 Package.json

In de package.json staan alle dependencies die het project gebruikt. Ook staat hier informatie over bepaalde commands die gebruikt kunnen worden. Een van deze commands is npm start. Deze is in het project omgezet tot het volgende:

```
"start": "ng serve --proxy-config proxy.conf.json --open",
```

Figuur 1 npm start in package.json

Dit zorgt ervoor dat de applicatie ook de proxy gebruikt. Hierdoor gaan de calls naar het goede adres.

4.3.2 Material module

In het project wordt gebruik gemaakt van material design. Om in de HTML files gebruik te maken van bijvoorbeeld een material button moet deze geïmporteerd worden. Dit gebeurt in de material.module.ts. Eerst wordt bovenin de file een onderdeel geïmporteerd:

```
2 import { CommonModule } from '@angular/common';  
3 import { MatCardModule } from '@angular/material/card';
```

Figuur 2 importeer MatCardModule

Na het importeren komen ze vervolgens in de array van imports en exports in dezelfde file:

```
5
6 @NgModule({
7   declarations: [],
8   imports: [
9     CommonModule,
10    MatDividerModule,
11    MatCardModule,
12    MatInputModule,
13    MatFormFieldModule,
14    MatSidenavModule,
15    MatSelectModule,
16    MatSnackBarModule,
17    MatDialogModule,
18    MatCheckboxModule,
19    MatSlideToggleModule,
20    MatTooltipModule,
21    MatSortModule,
22    MatPaginatorModule,
23    MatTabsModule,
24    MatProgressSpinnerModule,
25    MatStepperModule,
26    MatExpansionModule,
27    MatListModule,
28    MatTableModule
29  ],
30  exports: [
31    MatDividerModule,
32    MatCardModule,
33    MatToolbarModule,
34    MatButtonModule,
35    MatIconModule,
36    MatInputModule,
```

Figuur 3 import en export MatCardModule

4.3.3 app.module.ts

Naast de material module is er ook nog een app module. Deze app module heeft net zoals de material module ook een import echter geen export. Eerst worden hier weer modules geïmporteerd en vervolgens tussen de imports gezet. Echter gaat het hier om modules die niet tot material behoren. Dit is om een duidelijke scheiding te houden tussen de material imports en overige imports. Ook wordt in deze app.module.ts de material module geïmporteerd om gebruik te maken van die module. Tussen de imports staat ook een translateModule. Deze module wordt hier geïmporteerd en geïnitieerd met andere modules. Dit is om later het vertalen mogelijk te maken. Ook is er een declarations array, in deze array worden de componenten aangegeven. Dit gebeurt meestal al automatisch als een component wordt gegenereerd. Als een pagina niet weergegeven wordt zou het er aan kunnen liggen dat hij nog niet tussen de declarations staat.

4.3.4 app.component.ts

In de app.component.ts wordt een deel van het stuk vertalen geregeld. Hier worden de beschikbare talen toegevoegd en wordt een standaard taal ingesteld. Hier wordt de taal in de local storage gezet van de gebruiker.

4.3.5 Assets folder

In het project zit een asset folder. In deze folder zitten de JSON files van de talen. De angular applicatie krijgt een bepaalde naam binnen bijvoorbeeld "SIDE_BAR_TITLE". Vervolgens kijkt de applicatie welke taal geselecteerd is in de local storage. Daarna kijkt angular in de corresponderende JSON of deze key bestaat en wordt de bijbehorende tekst opgehaald.

4.3.6 Routing

In de app-routing-module.ts staan alle routes van de applicatie. In deze file staat een routes array. De meeste routes hebben een path en een component. Path is een string waar de gebruiker heen moet navigeren om die pagina te bereiken dus bijvoorbeeld '/login'. De component is welk component opgehaald wordt als de gebruiker daarheen navigeert. Een eventuele derde parameter is de canActivate parameter. Hier wordt de AuthGuard aan meegegeven. Dit betekent dat de gebruiker ingelogd moet zijn om die pagina te gebruiken of voor sommige pagina's juist dat de gebruiker niet ingelogd moet zijn zoals de login pagina. Ook zijn er routes om de gebruiker te redirecten als ze niks invullen:

```
{ path: '**', redirectTo: 'home', pathMatch: 'full' },  
{ path: '', redirectTo: 'home', pathMatch: 'full' },
```

Figuur 4 redirect paths

Als een gebruiker dus '/' in de URL invult dan wordt de gebruiker naar de home route gestuurd.

4.3.7 Api service

De applicatie bevat een API service. Dit zijn de calls die de front-end maakt naar de backend. API calls zien er als volgt uit:

```
getAllCountries(): Observable<HttpResponse<Array<ICountry>>> {  
  return this.http.get<Array<ICountry>>('/api/user/getcountries', { observe: 'response' });  
}  
  
registerAccount(registerRequest: IRegisterRequest): Observable<HttpResponse<string>> {  
  return this.http.post('/api/user/register', registerRequest, { observe: 'response', responseType: 'text' });  
}
```

4.3.8 Component

Angular werkt met componenten. Elk component bestaat uit 4 files: html file, scss file, ts file en spec file. Om een nieuw component aan te maken moet de "ng generate component <naam>" command gebruikt worden. Deze command genereert automatisch deze files en voegt de component toe aan de app.module.ts.

4.3.8.1 HTML

De HTML file binnen een component is vrij standaard. Het is niet nodig om de <html> of <body> tags te gebruiken aan het begin van een pagina. Om een component van material te gebruiken kan gekeken worden op [deze](#) pagina. Op de pagina staan voorbeelden van de componenten met de HTML, CSS en TS erbij. In een Angular HTML file is het simpel om gebruik te maken van de variabelen vanuit de TS file. Als in de TS file bijvoorbeeld een variabele genaamd "titel" is met als waarde "dit is een titel" en dit moet weergegeven worden op de pagina kan dit bereikt worden door bijvoorbeeld <h1> {{ titel }} </h1> te doen. Tussen de haken kunnen TS variabelen worden aangeroepen. Dit wordt ook gebruikt om teksten in verschillende talen te weergeven dan komt tussen de haken het volgende te staan: {{ MENU_BAR_TITLE | translate }}. Zoals eerder uitgelegd is de MENU_BAR_TITLE de key waarnaar gezocht wordt in de JSON files van de talen. Vervolgens wordt er gebruik gemaakt van een "pipe" en dit zorgt ervoor dat na de key de translate wordt aangeroepen. In dit geval is translate de naam van de translateService die wordt aangemaakt in de TS file. Op deze manier kan dus menu bar zijn titel dus worden weergegeven in het Engels en Nederlands afhankelijk van welke taal de gebruiker heeft gekozen.

4.3.8.2 SCSS

SCSS is een taal die gecompileerd wordt naar CSS (net zoals TypeScript gecompileerd wordt naar JavaScript). Echter bevat SCSS een aantal extra features zoals style nesting. Echter is het mogelijk om de normale CSS syntax te blijven gebruiken.

4.3.8.3 TS

De TS file is waar de 'logica' van de pagina komt. Hier komen methodes te staan, worden services geïmporteerd die nodig zijn voor de HTML / TS en hier staan variabelen. De TS file heeft een constructor waar services binnen kunnen komen doormiddel van dependency injection:

```
constructor(  
  private translate: TranslateService,  
  private snackbarService: MatSnackBar,  
  private apiService: ApiService,  
  private router: Router  
) { }
```

Figuur 5 Voorbeeld van een constructor

Ook zijn er methodes om iets uit te voeren in een levenscyclus van een pagina. Zo kan tijdens het initialiseren van de pagina iets berekend worden in de "ngOnInit()" methode.

```
/*  
  Initialise categories and catalog numbers when page starts loading  
  See https://angular.io/guide/lifecycle-hooks for more information  
*/  
ngOnInit(): void {  
  this.initialisePage();  
}
```

Figuur 6 Voorbeeld van een ngOnInit

4.3.9 Testen

Voor end-to-end test wordt er gebruik gemaakt van Cypress. Met Cypress is het mogelijk om een pagina te bezoeken en op knoppen te drukken. Hierdoor kan de flow van de applicatie getest worden en meer een gebruiker zijn perspectief getest worden in plaats van de werkelijke functionaliteit. Cypress wordt automatisch geïnstalleerd wanneer het 'npm install' commando wordt uitgevoerd omdat dit een development-dependency is. Dit betekent dat Cypress niet meegeleverd wordt bij de release build. Om Cypress te starten kan het commando 'npm run e2e' uitgevoerd worden.

Alle Cypress testbestanden staan in de 'cypress/integration' map.

4.3.10 AuthGuard

De authguard maakt gebruik van de authentication service. De authguard kijkt voor de huidige URL of de gebruiker is ingelogd of niet als de gebruiker niet is ingelogd wordt de gebruiker naar de login gestuurd. Daarbij zijn er bepaalde routes met een uitzondering aangezien voor de bijvoorbeeld de registratie en login pagina een gebruiker niet ingelogd mag zijn. De gebruiker wordt dan naar de home pagina gestuurd.

4.4 Beheerbaar

4.5 Uitbreiden

Om de applicatie uit te breiden zijn nieuwe componenten nodig. Om een nieuw component aan te maken moet de "ng generate component <naam van component> --module=app.module.ts" bij het maken van componenten wordt een algemene naming convention aangehouden. Die luidt als volgt: app-<pagina-naam> en als het een echte pagina is komt er -page achter. Dus bijvoorbeeld app-login-page en app-menu-bar.

Zodra de component is aangemaakt en het is een pagina moet er een route worden toegevoegd aan de `app-routing.module.ts`. Als de pagina alleen bereikt mag worden als de gebruiker ingelogd is moet er een `canActivate` bij voor de `authGuard`.

5 .NET core

Als backend wordt gebruikt gemaakt van .NET core 5.0 in combinatie met EntityFramework.

5.1 Keuze

Er is met verschillende redenen gekozen voor .NET core. Er is een afweging gekomen gemaakt met andere backend talen en frameworks. Echter bleken deze allemaal in de context minder goed te passen dan .NET core. Daarnaast wordt er binnen Fontys veel gebruik gemaakt van Microsoft producten wat de keuze voor .NET core ondersteunt. Ook wordt er binnen de opleiding in het begin gefocust op C# wat ook helpt bij de overdraagbaarheid van dit project wat ook de keuze voor .NET core ondersteunt.

Binnen het project wordt gebruik gemaakt van microservices. Er is gekeken naar andere architecturen voor schaalbaarheid echter kwam daar geen beter resultaat is. Daarnaast wordt op deze manier ook een opzet gemaakt om microservices te leren.

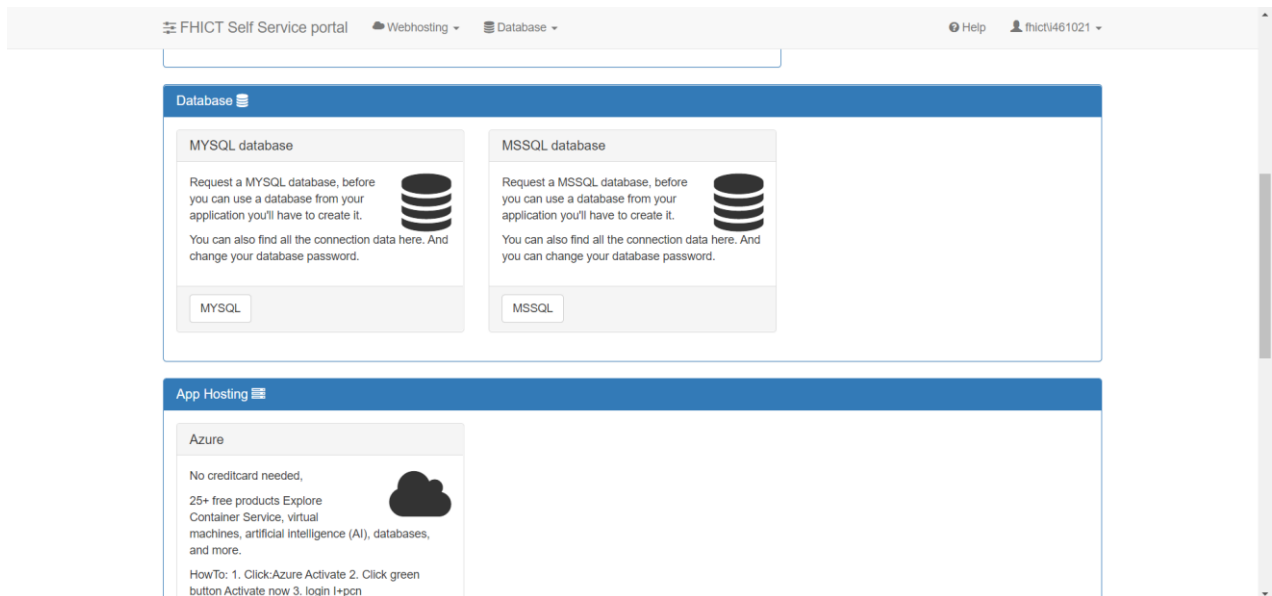
5.2 Opzet

Dit hoofdstuk zal gaan over het opzetten van het project in Visual Studio en de database opzetten.

5.2.1 Database opzet

In het originele overdrachtsdocument moet je een lokale database aanmaken, tijdens onze opzet van alle punten hebben wij gemerkt dat het ons niet lukte met een lokale database. Wij zijn gebruik gaan maken van de Selfservice van Fontys voor het aanmaken en gebruiken van de online database.

Voor het aanmaken van de database heb je een Fontys PCN nodig en het wachtwoord. Je gaat naar <https://selfservice.app.fhict.nl/>, je scrolt naar beneden en klikt op “MSSQL”



Daarna voer je voor je MSSQL server een wachtwoord in. Hieronder zie je hoe het eruit zou moeten zien als je je online database hebt opgezet.

Mssql

Account data

Host	mssql.fhict.local		
Database	dbi461021		
Username	dbi461021		
Password	The password you choose		

Visual studio

MSSQL connection string

Server=mssql.fhict.local;Database=dbi461021;User=Id=dbi461021;Password=YourChosenPassword;

You also have some extra databases:

Database	User	Action
dbi461021	dbi461021	<button>Reset this password</button>
dbi461021_i461021fh	dbi461021_i461021fh	<button>Reset this password</button>

(Re)set password

Database name (for extra databases)

Only letters and digits. 10 characters max.

i461021@fh

Password

.....

Set password

Connect from local machine?

To connect this database from your local machine, you'll need an VPN connection to vdi.fhict.nl. The manual can be found on [Studentenplein > VPN](#)

Na het opzetten van een database op de Selfservice van Fontys, maak je gebruik van SSMS (SQL Server Management Studio) om op de database in te loggen. Hierbij moet je gebruik maken van Cisco AnyConnect Secure Mobility Client, je logt in om te kijken of de database werkt.

Connect to Server

SQL Server

Server type: Database Engine

Server name: mssql.fhict.local

Authentication: SQL Server Authentication

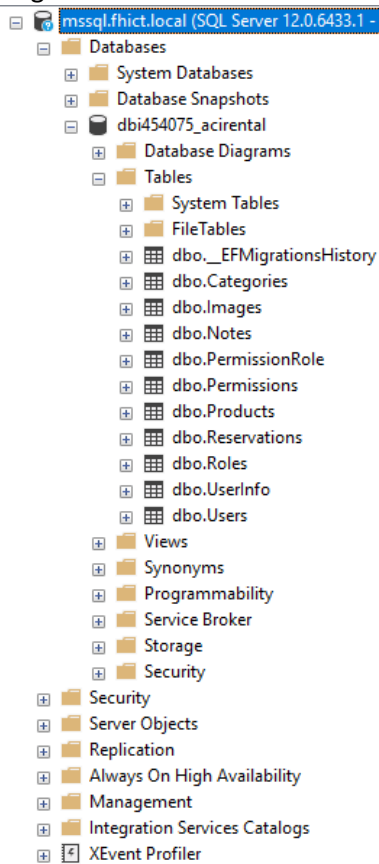
Login: dbi461021

Password: *****

☒ Remember password

Connect Cancel Help Options >>

Als je de front-end en back-end samen opstart zouden er migrations uitgevoerd moeten worden en zouden er database tables gemaakt moeten worden, hieronder staat een foto waarbij de migrations zijn uitgevoerd.



5.2.2 Migrations

Migrations uitvoeren, migrations zijn eigenlijk het verplaatsen van gegevens of software van het ene systeem naar het andere systeem. Om de migrations uit te voeren moet je eerst in het project de volgende NuGet Package installeren: "Microsoft.EntityFrameworkCore.Design". Deze package moet je in elke microservice installeren (versie 5.0.4). In de terminal van Visual Studio 2019 typ je het volgende, het is belangrijk dat je in de **FOLDER** van het project werkt.

- > dotnet tool install --global dotnet-ef
- > dotnet tool update --global dotnet-ef

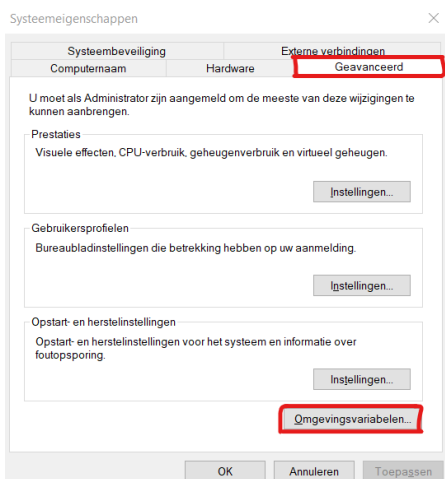
Hierna voor iedere service:

Cd naar de service

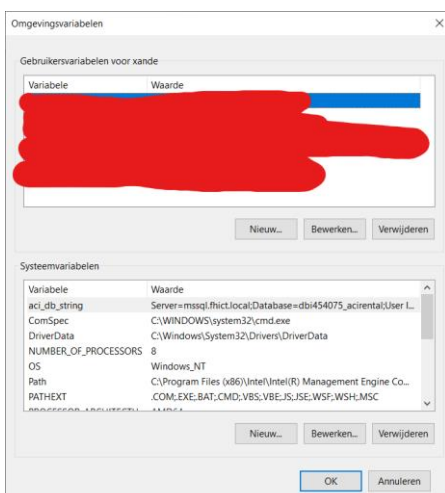
- > dotnet ef migrations add InitialCreate
- > dotnet ef database update

5.2.3 Omgevingsvariabelen

De omgevingsvariabelen gebruik je om de database te laten verbinden met de back-end, hiervoor gebruik je de connection string van je database en die zet je in je omgevingsvariabelen van je laptop. Je gaat eerst naar "Omgevingsvariabelen van systeem bewerken" dan klik je bovenin op "Geavanceerd", daarna klik je beneden op omgevingsvariabelen.



Je maakt daar een nieuwe omgevingsvariabele aan met de naam "aci_db_string" daar voer je de connectionstring van je database in. Als je klaar bent met deze stappen dan herstart je je computer.

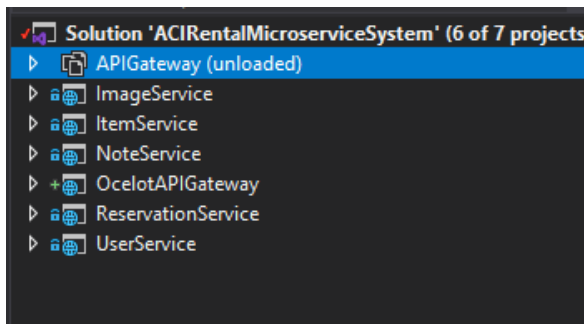


5.2.4 Visual studio opzet

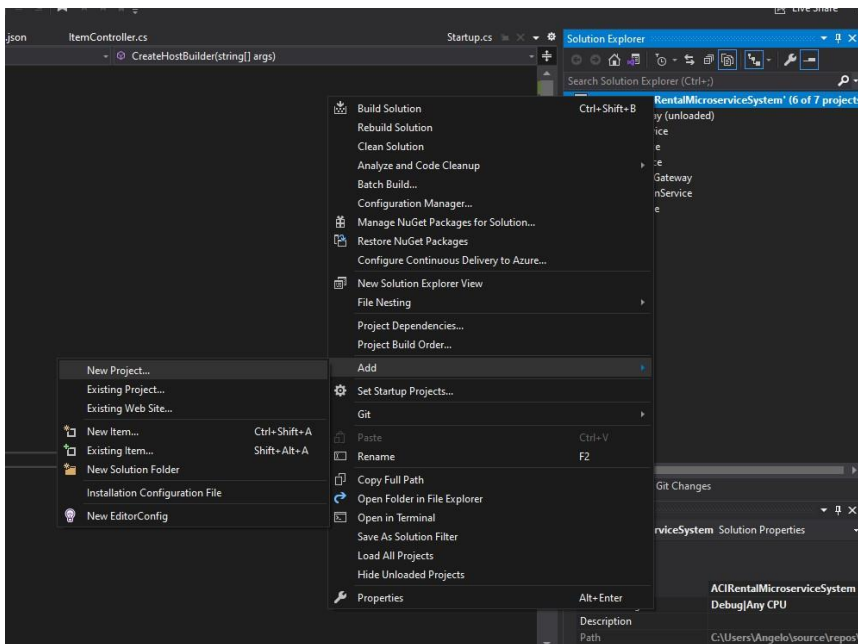
Voor het maken van een nieuwe microservice binnen de ACIRentalMicroserviceSystem zijn een aantal stappen vereist, voordat de opzet begonnen kan worden moet het project via git clone in Visual studio geïmporteerd worden, na het importen van het project kan er een nieuwe microservice worden toegevoegd aan de solution.

Het project bestaat uit een solution met meerdere microservices daarbinnen in. Binnen de solution heeft elke microservice een aanspreek punt met de OcelotAPIGateway, deze gateway dient ervoor om requests van de frontend naar de gateway te sturen en de gateway handelt deze requests dan weer naar de correcte microservice.

Een overzicht van de architectuur van de services is zichtbaar in het architectuursdocument.



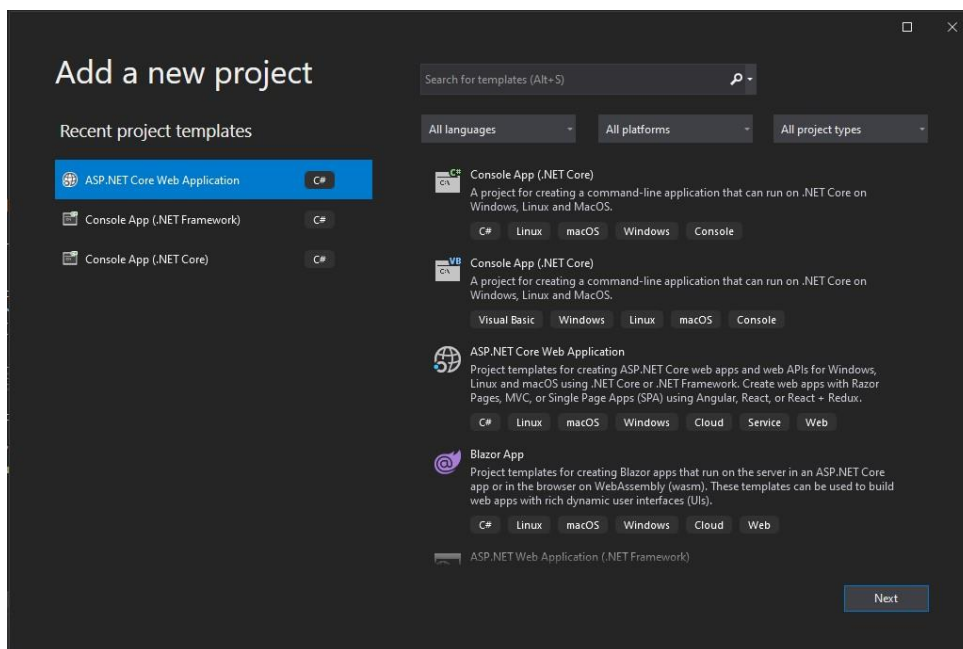
Figuur 7 De huidige services



5.2.5 Nieuwe microservice

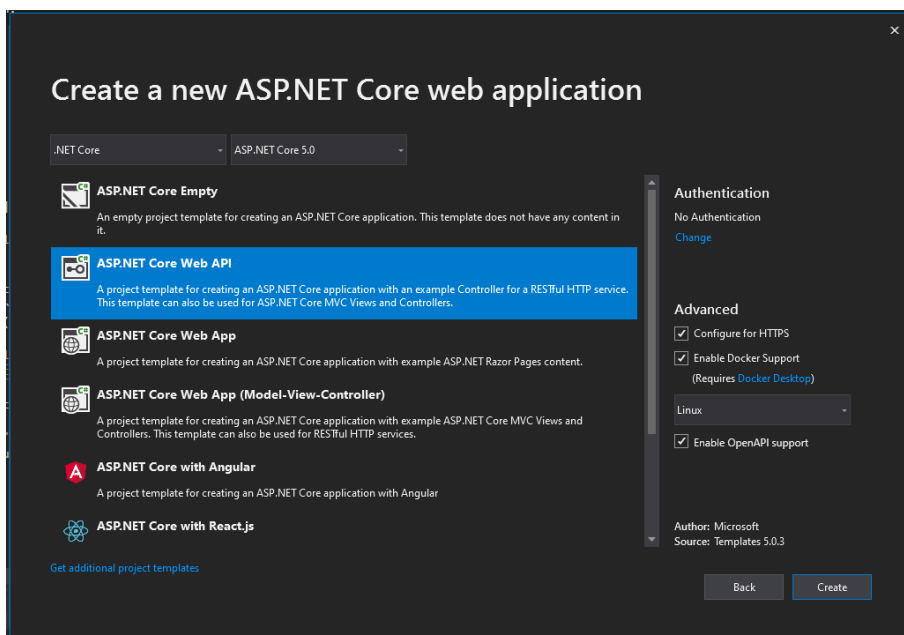
Figuur 8 Toevoegen van een nieuw project

Voor een nieuwe microservice, add een nieuw project toe aan de solution niet aan een bestaande microservice.



Figuur 9 Selecteren van het project-type

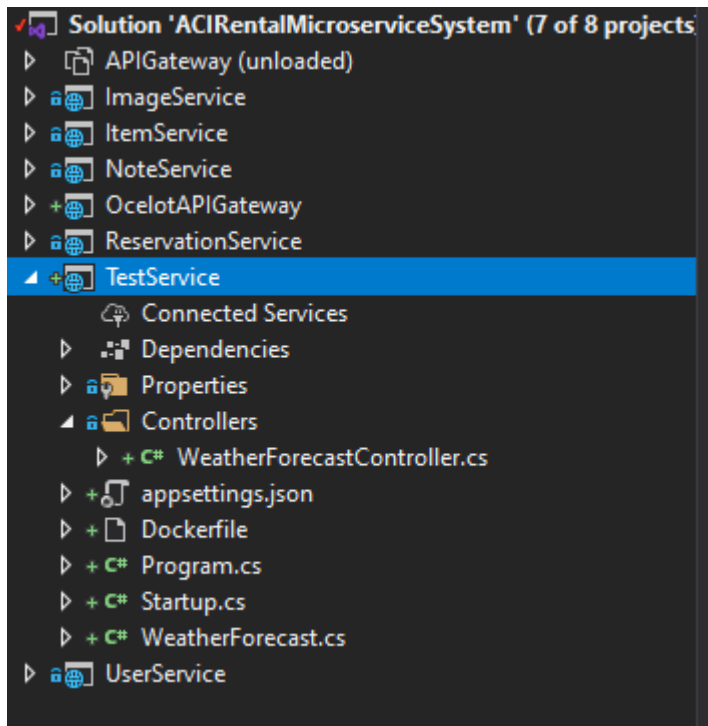
Na het klikken op een nieuw project komt er een add a new project scherm omhoog waarbij maak je een nieuw project met als template "ASP.NET Core Web Application"



Figuur 10 Selecteren van het project-type

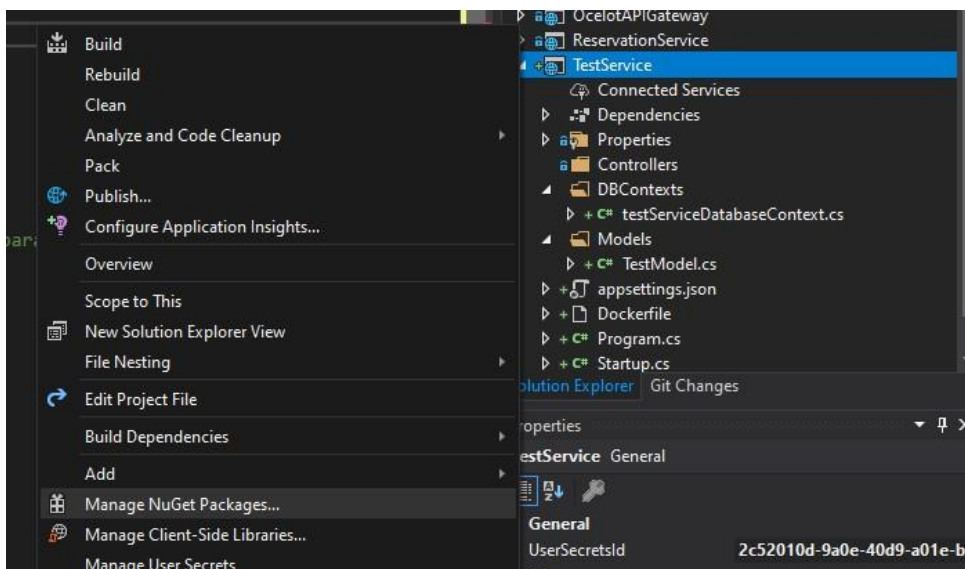
Zodra ASP.NET Core web API is gekozen moet nog niet op create worden gedrukt. Eerst moet zeker zijn dat bovenin ASP.NET Core 5.0 is gekozen. Daarnaast moeten aan de rechterkant bij advanced "Configure for HTTPS", "Enable Docker Support" en "Enable OpenAPI support" alle 3 aangevinkt zijn. Dit ziet er dan als volgt uit.

Als dit klopt kan er op de create knop gedrukt worden.



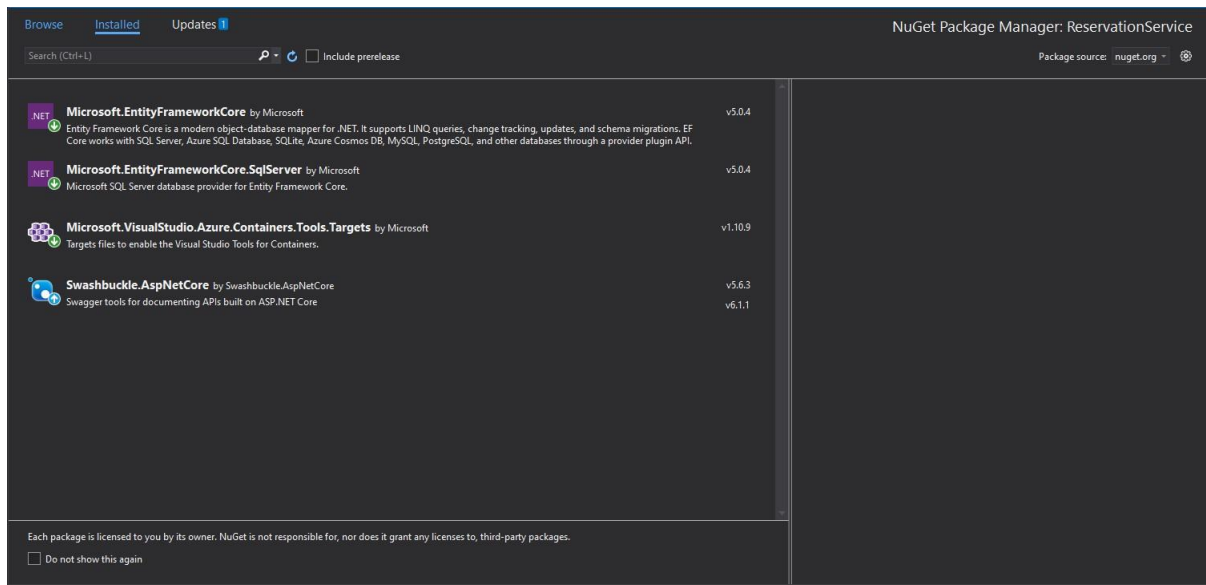
Figuur 11 Nieuwe service is aangemaakt

Na het aanmaken van de nieuwe microservice zullen de volgende files aanwezig zijn binnen de microservice: Controllers, Program, Startup en Properties deze files zijn het belangrijkst daarnaast worden er ook WeatherForecast file aangemaakt maar die zijn niet relevant en kunnen verwijderd worden.



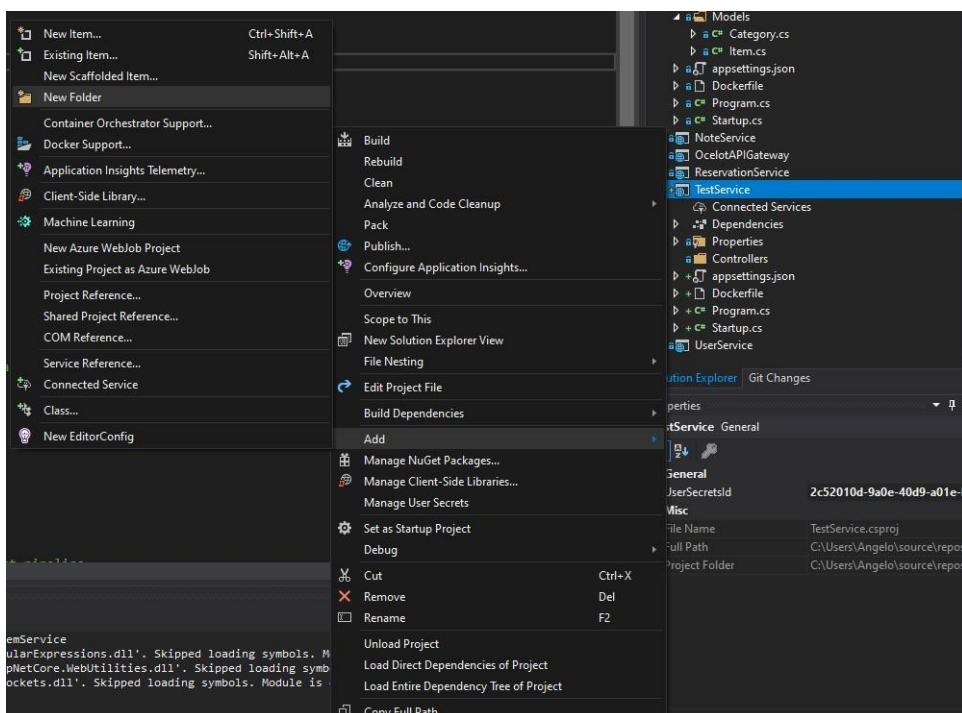
Figuur 12 Manage NuGet packages optie

Voor de service zijn een aantal nuget packages benodigd om deze te installeren wordt er rechterklik op de service gedaan en "manage nuget packages" gekozen.



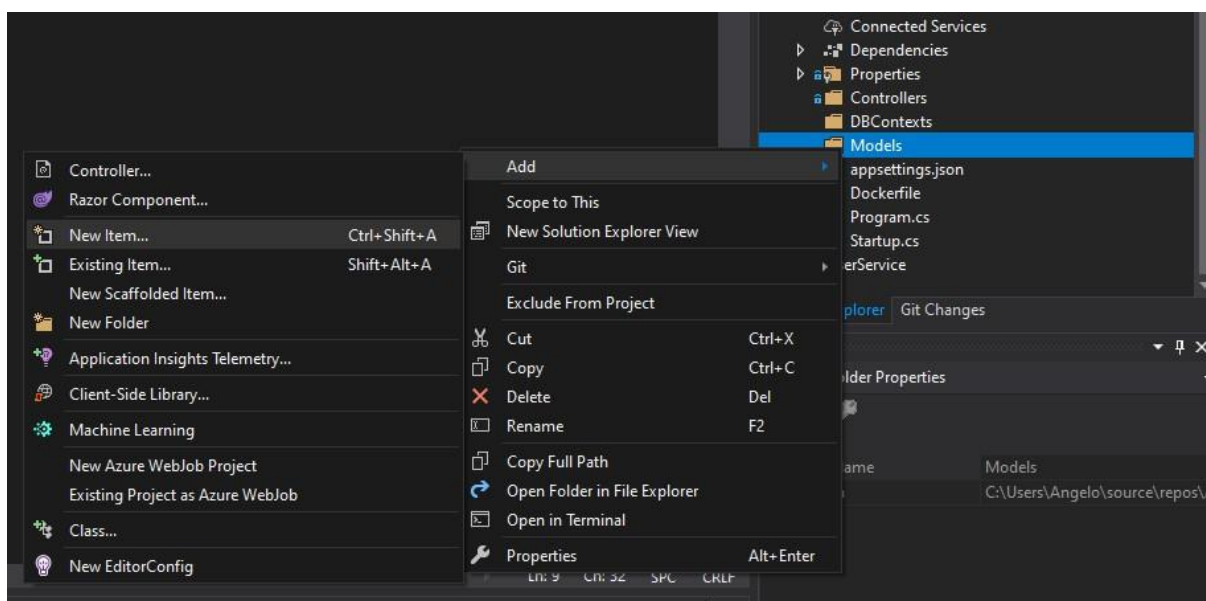
Figuur 13 De geïnstalleerde NuGet packages

Installeer hier EFCore. En SQL Server voor EFCore, deze zijn nodig om de database en entity framework te laten werken binnen het project.



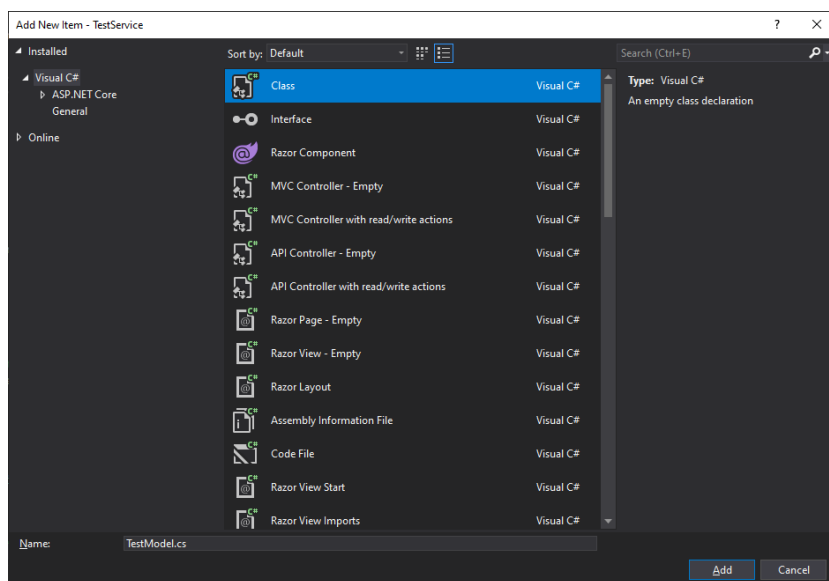
Figuur 14 Toevoegen nieuwe map

Voeg nu 2 nieuwe folders toe aan de service genaamd: Models en DBContexts. In deze 2 folders komen de files voor de nieuwe service, de Models folder gaat gebruikt worden om de database models te genereren voor de database, en de DBContexts zal gebruikt worden om de database context te maken die praat met de database.



Figuur 15 Toevoegen nieuwe class

Om een nieuwe class toe te voegen aan de folder, rechtermklik op de folder en voeg dan een new item toe.



Figuur 16 Toevoegen nieuwe class

Selecteer class, en geef het onderaan een naam. Dit proces is hetzelfde voor de DBContexts folder.

```

/// <summary>
/// OnConfiguring builds the connection between the database and the API using the given connection string
/// </summary>
/// <param name="optionsBuilder">Used for adding options to the database to configure the connection between it and the API</param>
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    if (!optionsBuilder.IsConfigured)
    {
        var dbString = Environment.GetEnvironmentVariable("aci_db_string");
        if (string.IsNullOrEmpty(dbString))
        {
            throw new MissingFieldException("Database environment variable not found.");
        }

        optionsBuilder.UseSqlServer(Environment.GetEnvironmentVariable("aci_db_string").Replace("DATABASE_NAME", "ImageService"));
    }

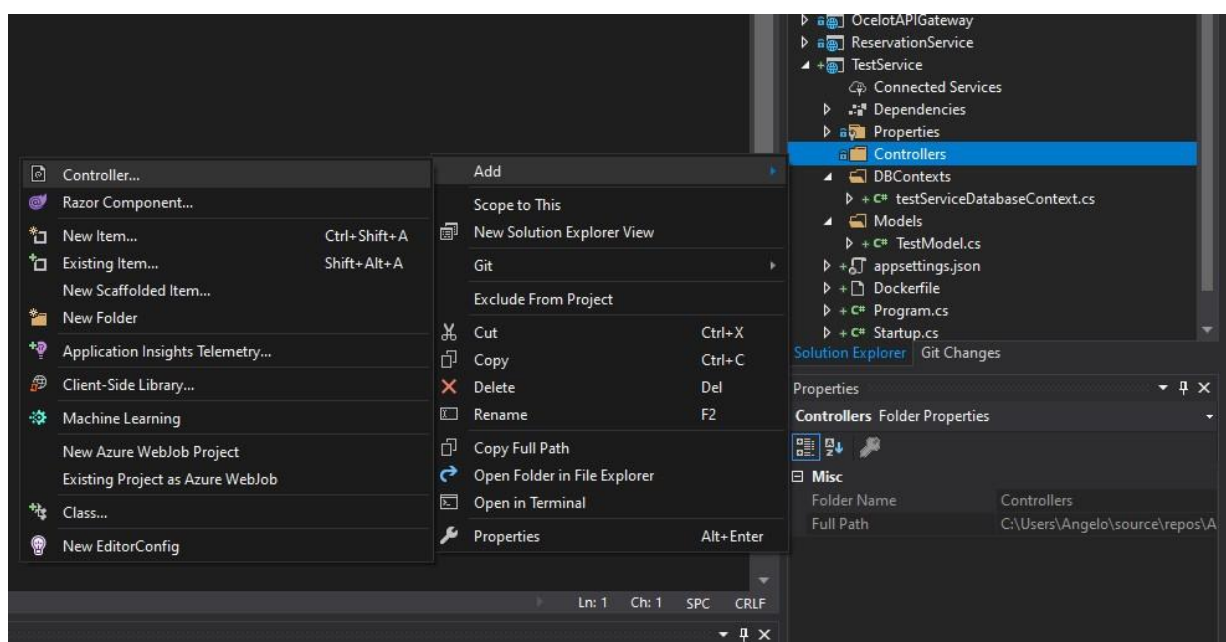
    base.OnConfiguring(optionsBuilder);
}

```

Figuur 17 Voorbeeld van een DbContext

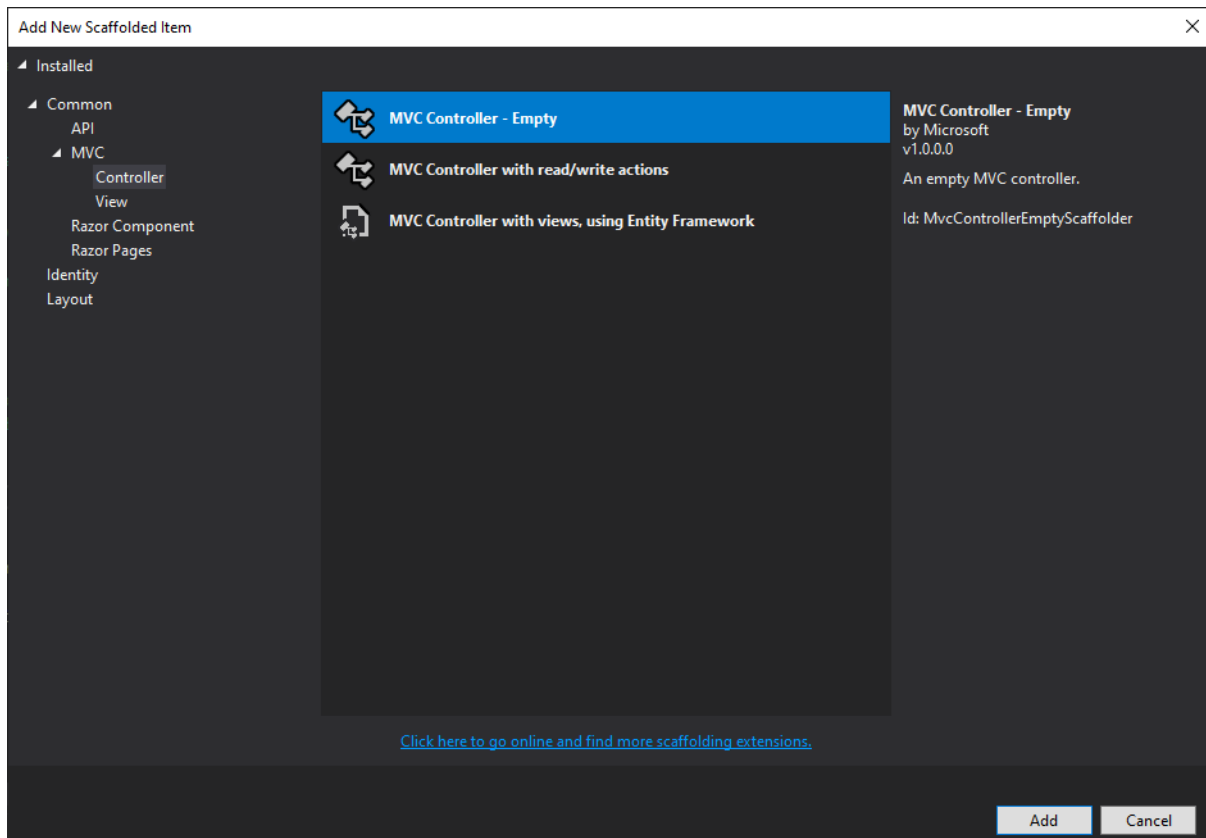
In de test servicedatabasecontext kun je de models toevoegen die gebruikt gaan worden in je database.

Dit zorgt ervoor dat hij de lokale database gaat gebruiken met de models.



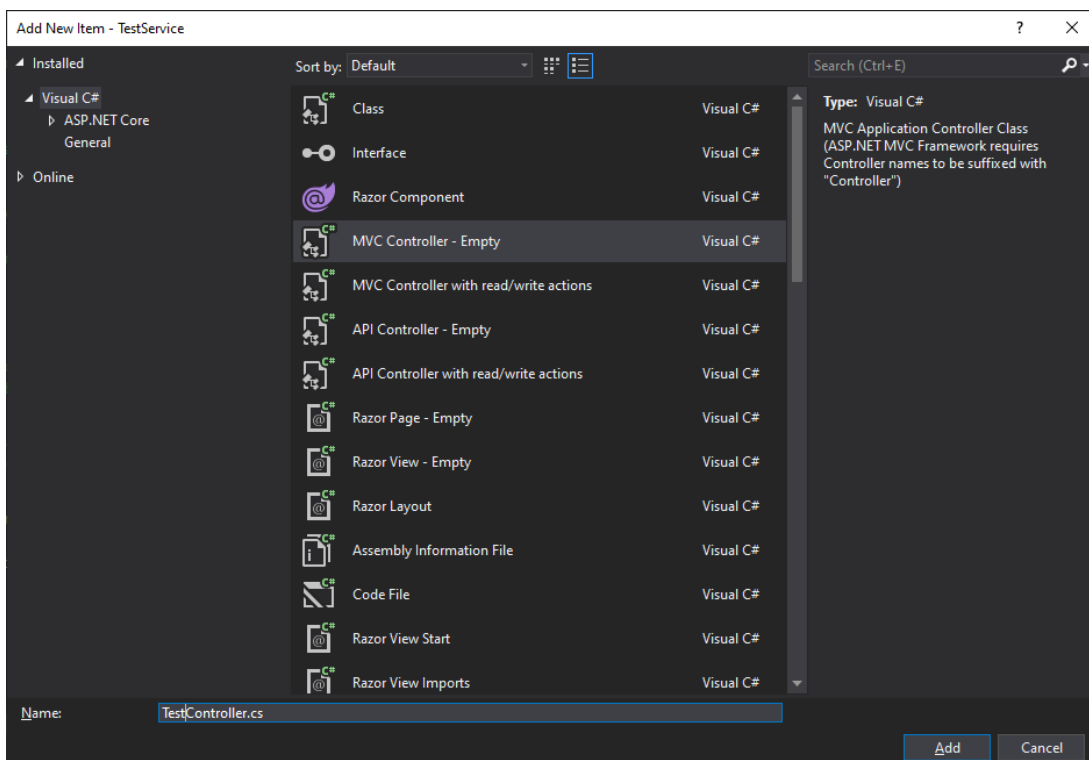
Figuur 18 Toevoegen controller

Voor de controller maak een nieuwe controller aan door rechterklik op de controllers folders te doen en dan een controller toe te voegen.



Figuur 19 Toevoegen controllers

Selecteer de MVC Controller-empty en dan add.



Figuur 20 Toevoegen controller

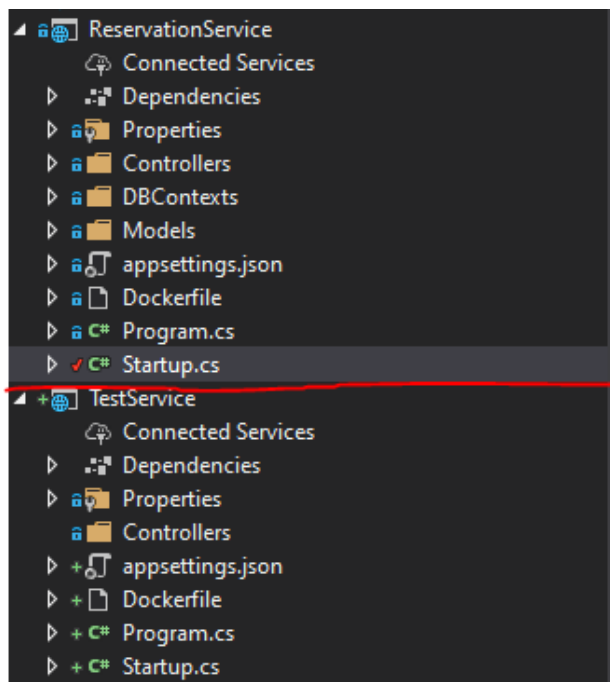
Geef de controller een naam, die eindigt op Controller.cs

```
namespace TestService.Controllers
{
    /// <summary>
    /// Item controller this controller is used for the calls between API and frontend for managing the items in the ACI Rental system
    /// </summary>
    [ApiController]
    [Route("[controller]")]
    public class TestController : ControllerBase
    {
        /// <summary>
        /// Database context for the item service, this is used to make calls to the item table
        /// </summary>
        public readonly testServiceDatabaseContext _dbContext;

        /// <summary>
        /// Constructor is used for receiving the database context at the creation of the item controller
        /// </summary>
        /// <param name="dbContext">Database context param used for calls to the item table</param>
        public TestController(testServiceDatabaseContext dbContext)
        {
            _dbContext = dbContext;
        }
    }
}
```

Figuur 21 Voorbeeld controller

Voeg in de controller de [ApiController] [Route] en : ControllerBase toe aan de bovenkant van de class. Daarnaast voeg je de database context toe met de constructor die hem aanmaakt. Vanuit hier kun je in de controller calls gaan bouwen die de database oproepen.



Figuur 22 Startup locatie

De startup.cs moet aangepast worden naar de database contexten, dit betekent dat de startup.cs gebruikt gaat worden om de database context op te roepen en zodra er een nieuwe service aangemaakt wordt zal er ook een nieuwe context toegevoegd worden.


```

namespace ItemService
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddControllers();
            services.AddSwaggerGen(c =>
            {
                c.SwaggerDoc("v1", new OpenApiInfo { Title = "ItemService", Version = "v1" });
            });
            services.AddDbContext<ItemServiceDatabaseContext>();

            using var itemContext = new ItemServiceDatabaseContext();
            itemContext.Database.EnsureCreated();
        }
    }
}

```

Figuur 23 Startup.cs met de database context

5.2.6 Cors

Voor het project is onderzoek gedaan naar CORS. Met CORS kan ervoor gezorgd worden dat requests alleen mogelijk zijn vanaf bepaalde adressen zoals bijvoorbeeld: <https://google.com>. Om de veiligheid van de applicatie te verbeteren is hiernaar gekeken om de CORS te beperken naar een aantal adressen zodat er vanaf weinig sites requests kunnen worden gestuurd. Dit is echter niet gelukt om werkend te krijgen binnen het project, wel is er kennis opgedaan tijdens het onderzoek die hier gedeeld gaat worden.

CORS kunnen ingesteld worden op 3 verschillende niveaus: voor de hele applicatie, voor één specifieke controller of voor één methode. De manier waarop deze niveaus bereikt kunnen worden is verschillend. Allereerst moet er ongeacht van welk niveau een CORS policy worden aangemaakt in de `startup.cs` in de `ConfigureServices()` methode. In deze CORS policy staat welke adressen geaccepteerd worden. Ook kunnen hier nog meer beperkingen opgelegd worden of juist minder door bijvoorbeeld methodes als `AllowAnyHeader()` te gebruiken. Deze policy krijgt een naam en daar kunnen vervolgens met de `withOrigins()` methode adressen aan worden toegevoegd. Zodra een policy is aangemaakt kan deze worden toegepast op de verschillende niveaus.

Om een policy te laten gelden voor de hele applicatie moet in de `Configure()` methode van de `startup.cs` de volgende line worden toegevoegd: `app.UseCors("<NAAM VAN DE POLICY>");`. De plek waar deze line wordt neergezet is van belang. De lijn moet na de `app.UseRouting()` methode en voor de `app.UseAuthorization()` methode staan. Let op dat door het gebruik van microservices deze policy alleen voor deze specifieke microservice geldt waar de `startup.cs` tot behoort.

Om een policy alleen voor een controller te laten gelden moet boven de naam een extra attribute neergezet worden. Op dit moment staat er al de `[Route]` en `[ApiController]` attributes hierboven moet het volgende geplaatst worden: `[EnableCors("<NAAM VAN DE POLICY>")]`. Het is mogelijk om een methode binnen de controller de CORS policy te laten negeren. Om dit te realiseren moet boven de methode de attribute `[DisableCors]` gezet worden.

Om een policy voor een specifieke methode te laten gelden moet een attribute boven de methode gezet worden. Dit is de `[EnableCors("<NAAM VAN DE POLICY>")]` attribute.

6 CI/CD

6.1 Inleiding

Binnen het project is gebruik gemaakt van Github Actions om een CI/CD op te zetten. Er zijn twee verschillende CI/CD files, een build file en een release file voor elke microservice en de front-end. De build file bouwt een project en runned de testen. Deze wordt uitgevoerd zodra er een pull request wordt gemaakt naar de main branch of de dev branch. De release file zorgt voor het pushen naar Docker hub. Deze wordt uitgevoerd zodra er wordt gepushed naar de main branch.

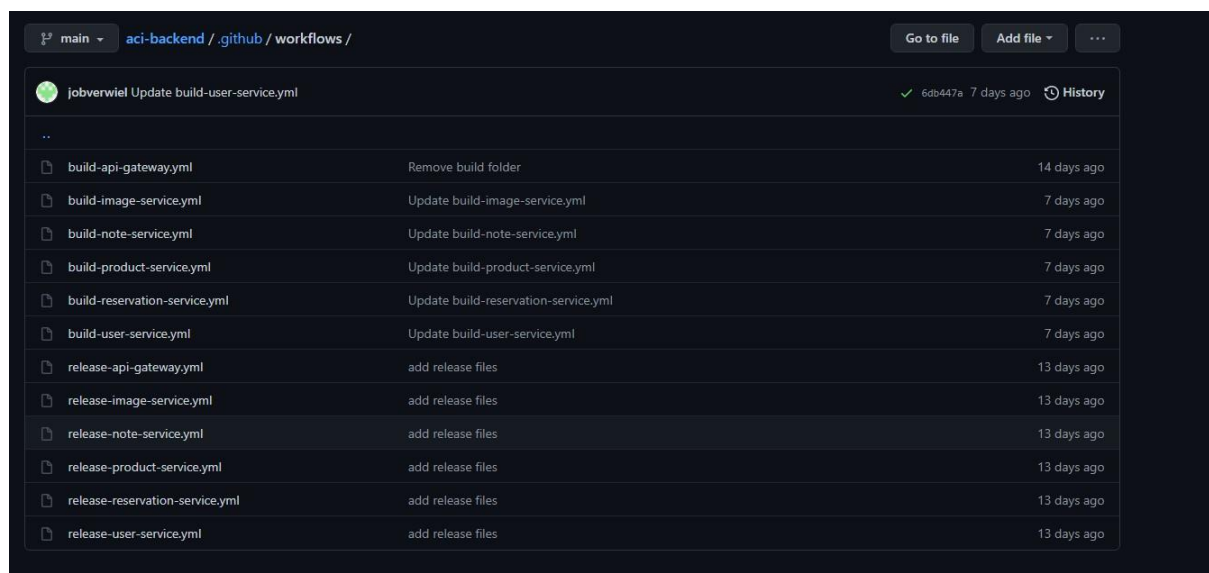
6.2 Keuze

Er is gekeken naar Azure DevOps, Jenkins, GitLab CI/CD en GitHub Actions. Aan de alternatieven van GitHub actions zitten een aantal nadelen zoals: kosten, lokaal runnen en zelf runners voor jobs regelen. GitHub Actions heeft hier geen last van. Daarnaast kan er bij GitHub actions gebruik gemaakt worden van paths. Door deze paths kan het makkelijk per microservice gemaakt worden zodat niet het hele project gebouwd of gepushed hoeft te worden maar dat dit gewoon los per service kan.

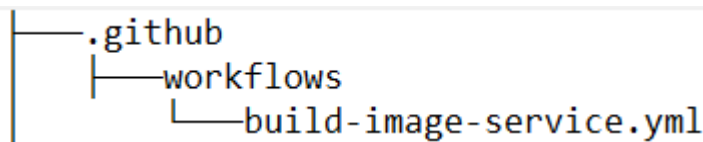
Daarnaast kan er binnen GitHub actions ook gebruik maken van secrets en environment variables. Hierdoor kunnen er nette en veiligere yml files worden aangemaakt voor de pipeline.

6.3 Opzetten GitHub Actions

Om gebruik te maken van GitHub actions **moet** het project op GitHub staan. Om oneindige runners te hebben moet het project ook publiek zijn. Zodra dit klopt moet een '.github' folder worden aangemaakt in de root folder. In de .github folder moet vervolgens een 'workflows' folder worden aangemaakt. In deze workflows folder kunnen de yml files aangemaakt worden. De folder structuur ziet er dus als volgt uit:



Figuur 24 Workflows folder



Figuur 25 file diagram

6.4 Yml file opbouwen

Een yml file bestaat uit drie verschillende onderdelen:

- On: beschrijft wanneer de pipeline moet worden uitgevoerd.
- Env: de environment variabelen
- Jobs: de verschillende stappen van de pipeline.

Alle drie de onderdelen zijn van belang om een nette pipeline op te zetten. De allereerste line van de yml file is **name: <naam van pipeline>**. Let bij het opzetten van de file ook goed op de inspringing van de verschillende lijntjes. Zodra iets een spatie te ver staat kan een pipeline al falen.

6.4.1 On

De eerste tab na de **on** komt wat voor soort actie het moet worden uitgevoerd. Dit kan een pull request zijn, een push of een pull. Hier kunnen er ook meerdere van zijn. Daarna kan er nog een tab worden toegevoegd en kan er met de **branches** de branch worden toegevoegd waarop dit van toepassing moet zijn.

```
on:
  push:
    branches:
      - main
```

Figuur 26 voorbeeld on push

Bij bovenstaande foto wordt de pipeline dus afgetrapt zodra er wordt gepushed naar de main branch. Ook kan er op dezelfde verspringing als branches ook nog **paths** worden toegevoegd. Dit betekent dat naast het pushen naar een branch de push ook nog aan een bepaald pad moet voldoen.

```
on:
  push:
    branches:
      - main
    paths:
      - "ImageService/**"
      - "ImageService.Tests/**"
      - ".github/workflows/build-image-service.yml"
```

Figuur 27 voorbeeld met paths

In bovenstaand voorbeeld moet er dus in een file in de ImageService folder veranderingen zijn én er moet gepushed worden naar de main branch. Dan pas gaat de pipeline af.

6.4.2 Env

Binnen de yml files kunnen er environment variabelen worden aangemaakt. Dit zijn, net zoals in code, variabelen waar je tekst aan koppelt zodat je niet dubbele tekst hoeft te kopiëren en het maar op één plek hoeft aan te passen. Na de **env** kan er een tab gebruikt worden om gelijk variabelen aan te maken. Een variabele ziet er als volgt uit:

```
env:  
  PROJECT_PATH: 'ImageService/ImageService.csproj'  
  TEST_PATH: 'ImageService.Tests/ImageService.Tests.csproj'
```

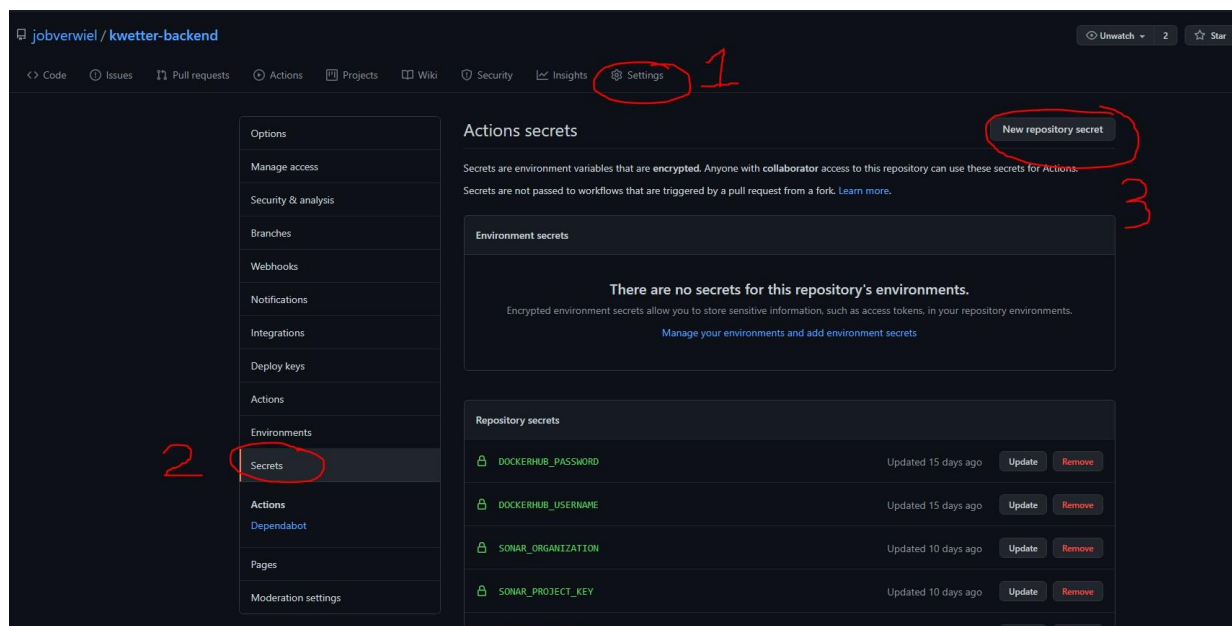
Figuur 28 environment variabelen

Vervolgens kan er later in de file dus gerefereerd worden naar deze variabelen. Dit ziet er als volgt uit:

```
run: dotnet restore ${env.PROJECT_PATH}
```

Figuur 29 refereren naar variabele

De **run** en **dotnet restore** die ervoor staan zijn niet van belang. Om variabele aan te roepen is alleen `${env.VARIABEL_NAAM}` nodig. Ook kan er in het bestand gerefereerd worden naar secrets op een zelfde soort manier. Een secret is een gevoelig stuk informatie, zoals een wachtwoord, waar wel gebruik gemaakt van moet worden maar die niet direct in de yml file leesbaar moet zijn. Om een secret aan te maken wordt er in het GitHub project genavigeerd naar de Settings pagina. Klik daarna op 'Secrets' en dan recht bovenin op 'New repository secret'.



Figuur 30 Secret aanmaken

Vul op de nieuwe pagina een naam voor de secret in en de value die hierbij hoort. **Let op!** De value is niet meer uit te lezen hierna dus onthoud de value goed of zet die tijdelijk ergens ook nog eens dubbel op een uitleesbare plek. Zodra een secret is aangemaakt kan hiernaar gerefereerd worden in de yml file.

Ook kan er een environment variabel aangemaakt worden met de waarde van de secret. Dit ziet er als volgt uit:

```
env:
  PROJECT_PATH: 'ImageService/ImageService.csproj'
  TEST_PATH: 'ImageService.Tests/ImageService.Tests.csproj'
  SONAR_TOKEN: '${ secrets.SONAR_TOKEN_IMAGE_SERVICE }'
  GITHUB_TOKEN: '${ secrets.GITHUB_TOKEN }'
```

Figuur 31 environment variabel met value van secret

6.4.3 Jobs

Na de **jobs** wordt na een tab de **build** toegevoegd. Daarna moet een tab worden toegevoegd met **runs-on** hieraan kan ubuntu-latest voor gebruikt worden. Dit is echter af te stellen naar iets anders. Hierna moet nog een tab worden toegevoegd met **steps**. Vervolgens kunnen een tab later de verschillende stappen worden uitgewerkt. Er is geen vaste blauwdruk voor hoe een step eruit moet zien echter zijn er wel wat voorbeelden:

```
steps:
- name: 'Checkout'
  uses: actions/checkout@v2

- name: 'Install dotnet'
  uses: actions/setup-dotnet@v1
  with:
    dotnet-version: '5.0.x'

- name: 'Restore packages'
  run: dotnet restore ${ env.PROJECT_PATH }

- name: 'Build project'
  run: dotnet build ${ env.PROJECT_PATH } --no-restore --configuration Release

- name: Run tests
  run: dotnet test ${ env.TEST_PATH }

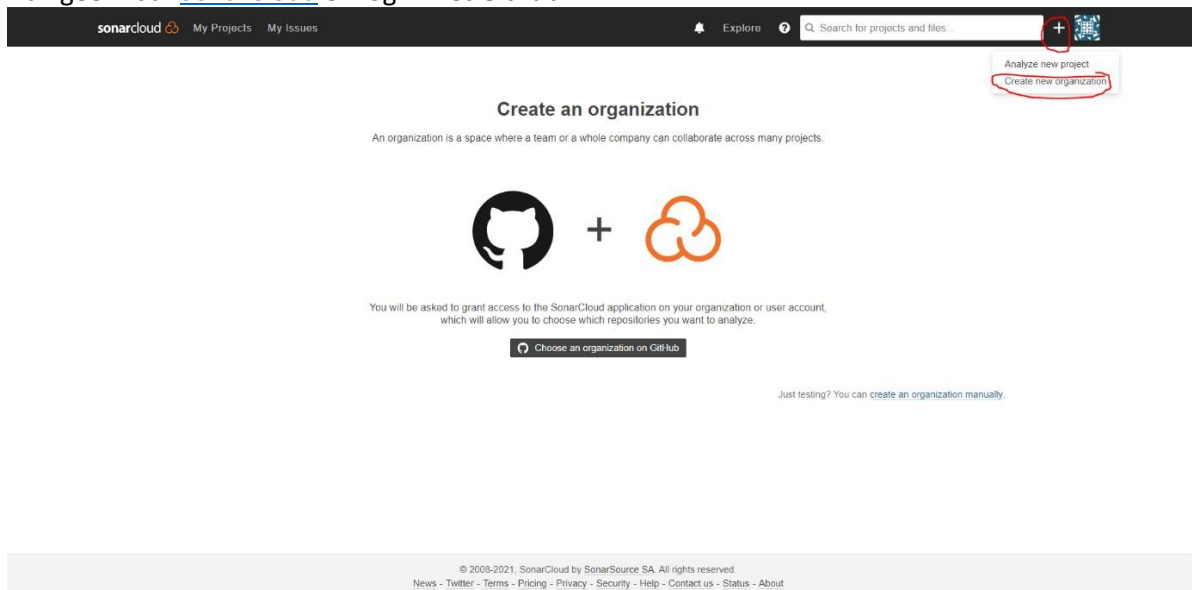
# https://github.com/highbyte/sonarscan-dotnet
- name: SonarScanner for .NET 5
  uses: highbyte/sonarscan-dotnet@2.0
  with:
    sonarProjectKey: siebrum_aci-backend-image-service
    sonarProjectName: aci-backend-image-service
    sonarOrganization: siebrum
    dotnetBuildArguments: ./ImageService
    dotnetTestArguments: ./ImageService.Tests
```

Figuur 32 stappen in pipeline

Een stap begint gebruikelijk met een **name** om de naam van de stap aan te geven. Zoals te zien is in het voorbeeld kunnen CLI's worden geïnstalleerd om in de pipeline te gebruiken. Zo wordt in het voorbeeld dotnet geïnstalleerd en worden er in stappen later dotnet commands gebruikt. Daarnaast kan er ook gebruik gemaakt worden van externe github actions. In het voorbeeld is dit het geval bij de SonarScanner stap. Met de **uses** wordt daar gerefereerd naar een externe Github action. Om handige Github actions te vinden met voorbeelden van hun implementatie kan gekeken worden op de [GitHub actions marketplace](#).

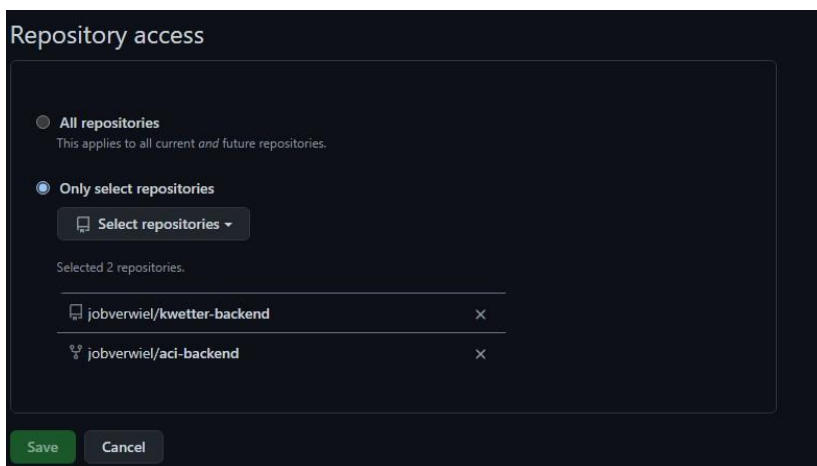
6.5 Opzetten SonarCloud

Binnen de pipeline wordt gebruik gemaakt van SonarCloud om te scannen op bugs, code smells en duplicate code. Een voordeel van SonarCloud in tegenstelling tot sonarqube is dat je niks lokaal hoeft te draaien of op een server. SonarCloud is gratis voor open source projecten en kost geld voor private repo's. In een microservice omgeving maak je binnen SonarCloud gebruik van een monorepo. Navigeer naar [SonarCloud](#) en login met Github.



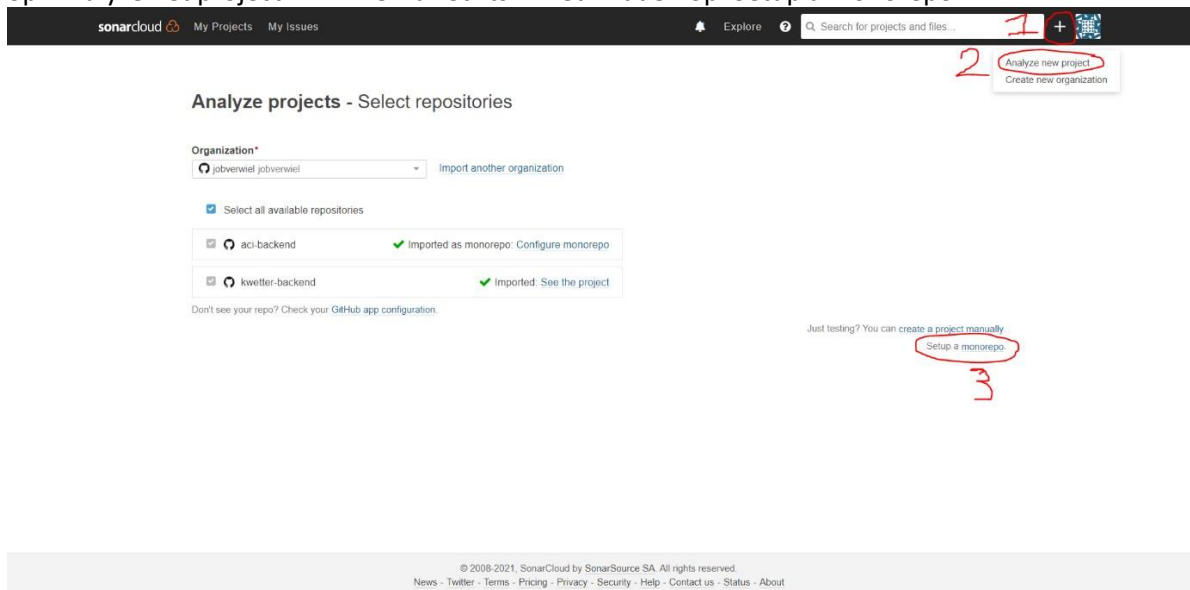
Figuur 33 create net organization

Klik rechtsbovenin op de het + teken en klik op Create net organization. Zorg ervoor dat het zo geconfigureert staat dat het GitHub account de organisatie is. Als er op de 'Choose an organization on GitHub' knop gedrukt wordt kan er ook voor gekozen worden om alleen bepaalde projecten toe te staan.



Figuur 34 alleen geselecteerde repo's

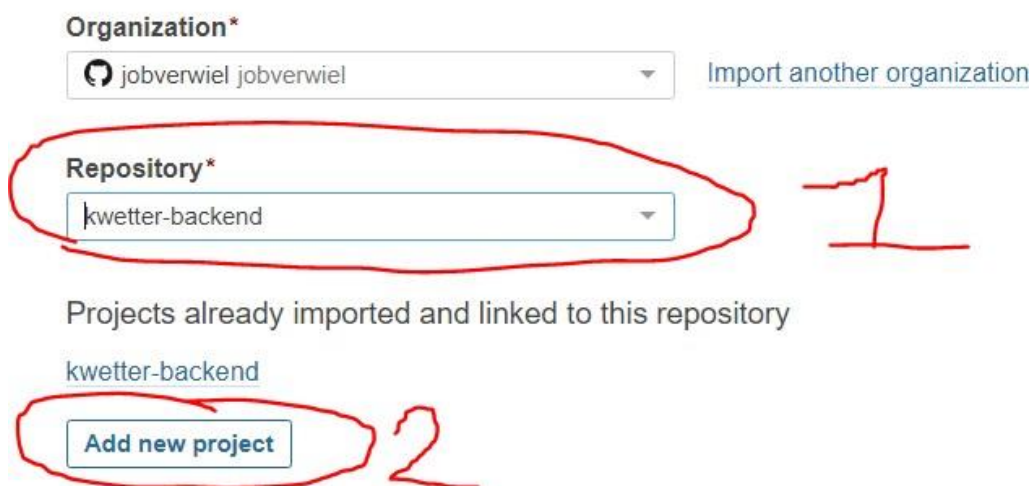
Als dit klopt en er toegang is tot het juiste project, klik rechtsbovenin wederom op het + teken en klik op 'Analyze net project'. Klik hierna rechts in het midden op 'Setup a monorepo'.



Figuur 35 Setup monorepo

Selecteer bij repository het juiste project en klik vervolgens op 'Add net project'

Analyze projects - Import monorepo



Figuur 36 nieuw project toevoegen.

In de context wordt elk project een microservice. Vervolgens moet er een project key gekozen worden voor het project en een display naam. Als voorbeeld wordt image service gebruikt. Houd de syntax aan dus: <github naam>_<project_naam>. In het voorbeeld dus jobverwiel_image-service. Voeg vervolgens voor elke microservice een nieuw project toe. Zorg er bij de benaming voor dat het **consistent** is en dat het makkelijk te **onthouden** is. Later zijn deze project keys namelijk relevant. Een voorbeeld van een project met drie microservices zou dit zijn:

Repository*

kwetter-backend

Projects already imported and linked to this repository

[kwetter-backend](#)

New projects

Project key* ?

jobverwiel_image_service



Up to 400 characters. All letters, digits, dash, underscore, period or colon.

Display name* ?

jobverwiel_image_service



Up to 255 characters

Project key* ?

jobverwiel_note_service



Up to 400 characters. All letters, digits, dash, underscore, period or colon.

Display name* ?

jobverwiel_note_service



Up to 255 characters

Project key* ?

jobverwiel_product_service



Up to 400 characters. All letters, digits, dash, underscore, period or colon.

Display name* ?

jobverwiel_product_service



Up to 255 characters

[Add new project](#)

Figuur 37 voorbeeld projecten voor monorepo

Zodra alle microservices er tussen staan klik op 'Save configuration'. Nu moet de SonarCloud aan de pipeline worden toegevoegd. **Let op** dat de volgende configuratie geldt voor een monorepo op GitHub voor de microservices en dat de er een pipeline is **per microservice**. In de pipeline komt de volgende stap te staan:

```
# https://github.com/highbyte/sonarscan-dotnet
- name: SonarScanner for .NET 5
  uses: highbyte/sonarscan-dotnet@2.0
  with:
    sonarProjectKey: jobverwiel_image_service
    sonarProjectName: image_service
    sonarOrganization: jobverwiel
    dotnetBuildArguments: ./ImageService
    dotnetTestArguments: ./ImageService.Tests
```

Figuur 38 SonarCloud stap

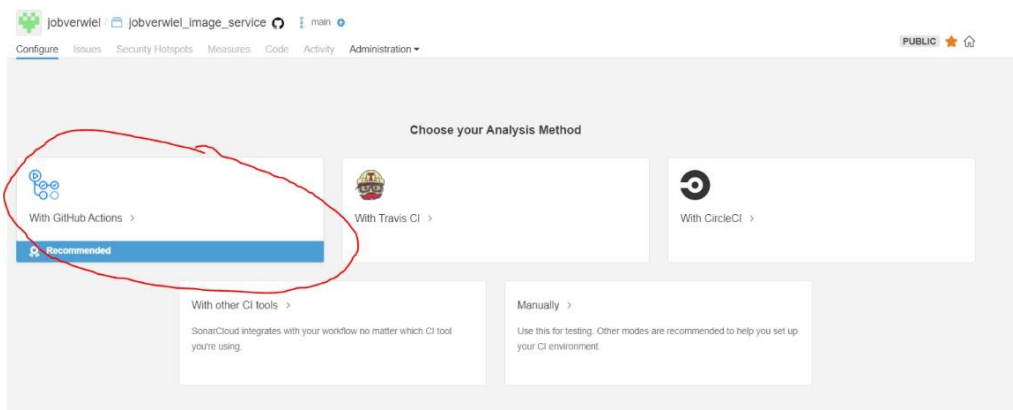
De **name** en **uses** zijn hier standaard. Bij de **with** parameters is configuratie nodig. **sonarProjectKey** is de eerder aangemaakte key. **sonarProjectName** is de project key **zonder** <organization naam>_. En **sonarOrganization** is de project key zonder de _<project naam>. Vervolgens moet er bij **dotnetBuildArguments** het pad worden meegegeven naar de service. Dit is vanuit de root directory houd daar rekening mee. **dotnetTestArguments** is een optionele parameter om de testen mee te sturen naar de SonarCloud omgeving. Bij zowel **dotnetBuildArguments** als **dotnetTestArguments** is extra configuratie mogelijk met de standaard dotnet parameters. Check voor meer informatie [de GitHub pagina van de GitHub action](#).

Ga vervolgens in de yml file naar het **env** gedeelte. Voeg hier de volgende variabelen toe:

```
env:
  SONAR_TOKEN: ${ secrets.SONAR_TOKEN_IMAGE_SERVICE }
  GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }
```

Figuur 39 SonarCloud variabelen

De **SONAR_TOKEN** en **GITHUB_TOKEN** variabelen zijn beide nodig om naar SonarCloud te publiceren. Deze namen zijn ook niet aanpasbaar. De **GITHUB_TOKEN** secret hoeft **niet** aangemaakt te worden. Deze staat standaard al ergens op GitHub echter wordt er op deze manier naar gerefereerd. De **SONAR_TOKEN** secret moet wel apart worden toegevoegd. Navigeer naar de [SonarCloud projecten](#). Klik hier vervolgens op het bijbehorende project voor de microservice. Op de overzicht pagina van het project, klik op 'With GitHub Actions':



Figuur 40 'With GitHub Actions' knop

Kopieer de value die hier staat:

1 Create a GitHub Secret

In your GitHub repository, go to [Settings > Secrets](#) and create a new secret with the following details:

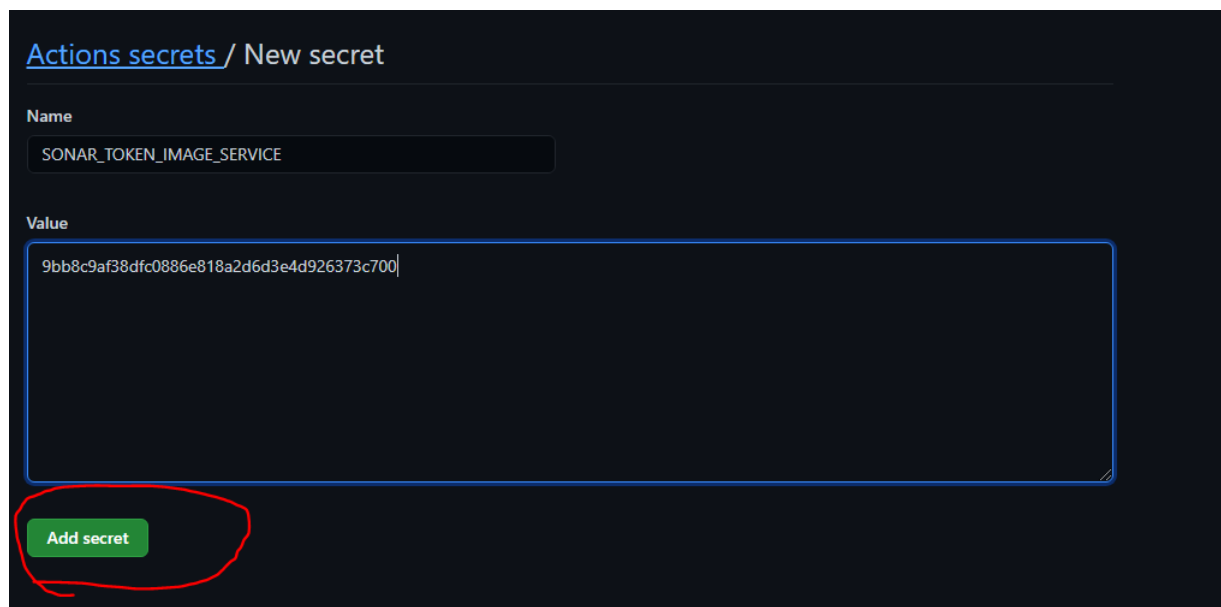
1 In the Name field, enter `SONAR_TOKEN`

2 In the Value field, enter `9bb8c9af38dfc0886e818a2d6d3e4d926373c700`

Continue

Figuur 41 Sonar token value

Dit is de sonar token van het project. Navigeer nu naar de secrets in GitHub van het project. Maak hier een nieuwe secret aan. Geef de secret een logische naam zoals: **SONAR_TOKEN_<NAAM VAN SERVICE>**. Dit ziet er als volgt uit:



[Actions secrets](#) / New secret

Name

SONAR_TOKEN_IMAGE_SERVICE

Value

9bb8c9af38dfc0886e818a2d6d3e4d926373c700

Add secret

Figuur 42 nieuwe SONAR_TOKEN secret

Klik op de 'Add secret' knop. Zodra de secret aangemaakt is ga weer terug naar de yml file. Kijk wederom naar de variabelen:

```
env:  
  SONAR_TOKEN: ${ secrets.SONAR_TOKEN_IMAGE_SERVICE }  
  GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }
```

Figuur 43 variabelen

Kijk of de `${ secrets.<NAAM> }` bij `SONAR_TOKEN` overeenkomt met de **net aangemaakte secret**.

6.6 Pushen naar DockerHub

Om uiteindelijk van de projecten gebruik te maken in een Kubernetes Cluster moeten de projecten gepushed worden naar DockerHub. DockerHub is een platform waar Docker images naar toe gepushed worden. Vervolgens is het mogelijk om overal ter wereld waar de gebruiker ook is deze images te pullen en te runnen in een Docker container.

6.6.1 Dockerfile

Om een applicatie om te zetten naar een Docker image wordt er gebruik gemaakt van een Dockerfile. Een Dockerfile bouwt de applicatie/project tot een pakketje. Vervolgens wordt dit pakketje gepushed naar Docker hub waardoor het een Docker image wordt. De Dockerfile staat in dezelfde directory als de microservice zijn **program.cs** en **startup.cs**. Een Dockerfile voor een .NET Core 5.0 microservice ziet er als volgt uit:

```
FROM mcr.microsoft.com/dotnet/aspnet:5.0 AS base
WORKDIR /app
EXPOSE 80

FROM mcr.microsoft.com/dotnet/sdk:5.0 AS build
WORKDIR /src
COPY ["ImageService.csproj", "ImageService/"]
RUN dotnet restore "ImageService/ImageService.csproj"
WORKDIR "/src/ImageService"
COPY . .
RUN dotnet build "ImageService.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "ImageService.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "ImageService.dll"]
```

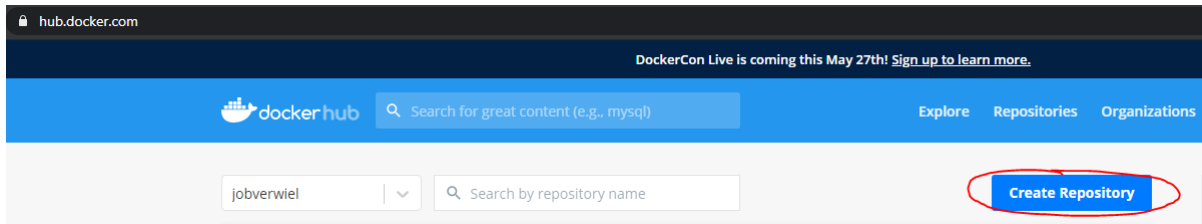
Figuur 44 image service Dockerfile

De hele Dockerfile is van belang echter zijn er twee belangrijke zaken waar aandacht vereist is:

- Expose 80: exposed de port van de container. In een standaard applicatie is de poort die aangesproken wordt port 80. Als later de docker image gebruikt wordt moet deze 80 gemapped worden. Als deze expose verkeert staat is de API niet bereikbaar. Het kan zijn dat in de **Program.cs** van een applicatie `.UseUrls(https://<een random port>)` gebruikt wordt. In dit geval zijn er 2 opties. Of haal de `.UseUrls` weg waardoor de standaard port 80 gebruikt wordt. Of maak in de Dockerfile de EXPOSE ... gelijk aan de port die in de `UseUrls` functie staat.
- Omdat deze Dockerfile van de image microservice is staat overal `ImageService`. Dit moet aangepast worden naar de situatie waarin de Dockerfile staat. Als dit de Dockerfile zou zijn voor de `NoteService` moet 'ImageService' overal aangepast worden naar 'NoteService'.

6.6.2 DockerHub

Navigeer naar [DockerHub](https://hub.docker.com) en maak een account aan als die er nog niet is. Log vervolgens in en klik op de hoofdpagina op de 'Create repository' knop.



Figuur 45 create repository knop

Vul op de nieuwe pagina een naam in voor de repository en kies als visibility voor public. Klik vervolgens op 'Create'. Nu de docker hub repository bestaat kan er naar toe gepushed worden om een docker image te maken. Om nu een gepushte image te runnen kan de volgende command gerunnen worden “**docker run -p 8080:80 <dockerhub naam>/<dockerhub repository>**”. Op deze manier is de service bereikbaar op localhost:8080.

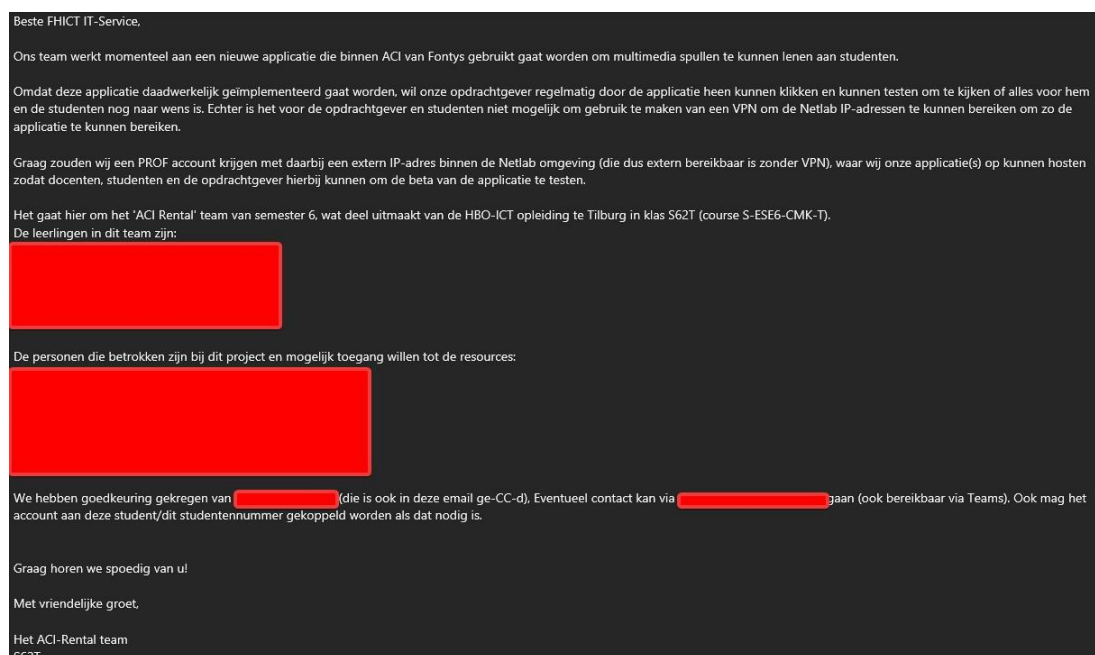
7 Deployment op NetLab met een extern IP-adres

7.1 Het aanvragen van een PROF-account

Het is voor alle studenten mogelijk om in te loggen op de NetLab omgeving van FHICT. Echter hebben deze machines standaard geen extern IP adres, wat betekend dat deze machines zonder VPN niet bereikbaar zijn. Dit resulteert in dat opdrachtgevers en docenten niet bij de applicatie kunnen en er bijvoorbeeld geen gebruikers-tevredenheidtests of beta-tests gedaan kunnen worden. Ook is een server met een extern IP-adres handig om te gebruiken als een 'staging' omgeving, een omgeving die hetzelfde is ingericht als de 'echte' live applicatie, om zo te testen of alles werkt zoals verwacht voordat de software gereleased word.

Een PROF account kan aangevraagd worden door een mailtje te sturen naar fhictit-service@fontys.nl. In dit mailtje moeten de volgende dingen voorkomen:

- Waarom er een PROF account aangevraagd word
- Welke leerlingen aan het project werken (+ Fontys nummers en klas)
- Welke docenten en eventuele andere stakeholders betrokken zijn bij het project
- Welke docent(en) er goedkeuring hebben gegeven om een extern IP-adres aan te vragen (ook deze docenten in de CC zetten)



Figuur 46 Voorbeeld van een mail om een PROF account aan te vragen

7.2 Opzetten van een VM

Op de Sharepoint omgeving van FHICT staat een [handleiding hoe een Kubernetes VM opgezet kan worden](#). Er word aangeraden deze handleiding te volgens TOT EN MET hoofdstuk 4.4 'Kubernetes commands'. Hoofdstuk 4.5 is momenteel nog niet relevant.

Wanneer de handleiding gevolgt is, dient de Kubernetes DNS aangezet te worden. Dit kan met het commando '**sudo mikrok8s enable dns**'. Hiermee krijgen alle machines binnen de Kubernetes omgeving een IP en lokaal domeinnaam en kunnen de machines tussen elkaar communiceren.

Hierna is het van belang de VM te beschermen tegen eventuele bots die over het internet surfen om je server te misbruiken. Hiervoor moet een firewall opgezet worden. Standaard levert FHICT een

template voor een PFSense firewall. Een uitleg hoe deze opgezet kan worden is te vinden onder het document 'Opzetten PFSense in NetLab'. In dit document staat beschreven hoe de firewall ingesteld moet worden en hoe de machine waarop je applicatie draait gebruik kan maken van de firewall.

7.3 YAML-bestanden

Om de containers waarin de services en front-end in zitten aan te zetten, moeten YAML files gebruikt worden. De al gemaakte YAML files zijn meegeleverd in de project documentatie, echter zal hieronder uitgelegd worden hoe deze YAML files in elkaar zitten. Het is belangrijk om te weten dat het type LoadBalancer momenteel niet gebruikt kan worden in de NetLab omgeving.

Huidige gebruikte YAML bestanden zijn beschikbaar in de GitHub repository. Deze kunnen als templates gebruikt worden voor eventueel andere services.

7.3.1 Secrets

Secrets zijn YAML bestanden waarin constante waardes worden gezet die mogelijk gevoelig kunnen zijn. Denk hierbij aan wachtwoorden en database connection-strings. Deze secrets worden opgezet met een YAML file.

Als eerst word de kind aangegeven, wat in dit geval een 'Secret' is. Daarbij word eventueel een apiVersion meegegeven. Die kan het beste standaard op 'v1' gezet worden. Daarna moet er metadata meegegeven worden met onder andere de naam van de secret. Hier kan elke naam gebruikt worden die gewenst is. Als laatst moet de data ingevoerd worden. De data begint eerst ook een naam met daarna de data die opgeslagen moet worden. Deze data dient in base64 gecodeerd te worden.

```
kind: Secret
apiVersion: v1
metadata:
  name: aci-db-string
data:
  connectionstring: >-
    U2VydmVyPXB2R1Y3RzZXJ2aW1mRhdGF1YXN1LndpbmRvd3M=
type: Opaque
```

Figuur 47 Een voorbeeld van een YAML file met als key;value: aci-db-string;connectionstring

7.3.2 Services

Het opzetten van een backend en front-end gaat in twee stappen, het opzetten van een deployment en het opzetten van een service. De service zal uiteindelijk de deployment uitvoeren en aanzetten.

In de deployment word gedefinieerd wat de naam van de container is, hoeveel replica's er standaard moeten draaien, welke image gebruikt moet worden, welke resources nodig zijn, welke port in de container gebruikt word en eventuele systeemvariabelen in de container(s) kunnen gezet worden. Omdat de huidige applicatie gebruik maakt van de systeemvariabelen om de database string op te halen, moet deze variabele in de backend services gezet worden. De database string is in vorig hoofdstuk opgeslagen als een secret en de YAML biedt de mogelijkheid om systeemvariabelen te vullen met secrets.

```
env:
- name: aci_db_string
  valueFrom:
    secretKeyRef:
      name: aci-db-string
      key: connectionstring
```

Figuur 48 Omgevingsvariabelen zetten op basis van een secret

In de service word gedefinieerd welk type service er gebruikt word en welke deployment er gebruikt moet worden. In de voorbeelden van de API word een ClusterIP gebruikt. Dit type krijgt standaard een intern IP adres en een interne DNS naam. Hierdoor is de service bereikbaar voor andere services. Ook word er aangegeven welke port er gebruikt moet worden.

```
apiVersion: v1
kind: Service
metadata:
  name: aci-note-service
spec:
  type: ClusterIP
  ports:
    - port: 80
  selector:
    app: aci-note-service
```

Figuur 49 Voorbeeld van een service

7.3.3 Ingress

Ingress is de tool waarmee de containers online bereikbaar worden gemaakt. Standaard staat Ingress niet ingeschakeld. Hoe Ingress werkt en hoe het ingeschakeld kan worden staat in het hoofdstuk '4.5 Connecting Kubernetes to the outside world' in het [document wat op SharePoint staat](#).

Wanneer Ingress aangezet is, moet er een YAML file aangemaakt worden waarin aangegeven word welke services Ingress moet exposen naar het internet en via welke URL.

```
kind: Ingress
metadata:
  name: http-ingress
  annotations:
spec:
  rules:
    - http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: aci-frontend
                port:
                  number: 80
          - path: /api
            pathType: Prefix
            backend:
              service:
                name: aci-api-gateway
                port:
                  number: 80
```

Figuur 50 Voorbeeld van een Ingress YAML

Hierboven staat een voorbeeld van hoe een Ingress YAML file eruit moet zien. In dit bestand staat dat alle standaard routes naar de 'aci-frontend' service toe wijzen. Dit word gedaan via port 80 (HTTP). Het is van belang dat binnen de front-end container de correcte NGINX instellingen voor Angular worden gebruikt. Deze instellingen moeten aangepast worden in de Dockerfile in de front-end repository. Momenteel staan die al correct ingesteld.

Onder de front-end word ook aangegeven dat alles wat via de /api route gaat, niet naar de aci-frontend service toe moet maar naar de aci-api-gateway service. Op deze manier kan de front-end via de <url>/api route calls maken naar de api-gateway.