Professionele Bachelor Toegepaste Informatica

# Community configurator

Ryan Majorczyk

Promotoren:

Jeff Vaes — Cubigo

Wesley Hendrikx — Hogeschool PXL Hasselt

Bachelorpaper Academiejaar 2018-2019

Professionele Bachelor Toegepaste Informatica

# Community configurator

Ryan Majorczyk

Promotoren:

Jeff Vaes — Cubigo

Wesley Hendrikx — Hogeschool PXL Hasselt

Bachelorpaper Academiejaar 2018-2019

# Dankwoord

Met dit eindwerk eindigt ook mijn schoolloopbaan. Als ik terugkijk naar de afgelopen jaren zijn er momenten geweest, waar ik zonder de steun van een aantal mensen, er nooit doorgeraakt zou zijn. Aan deze personen zou ik graag een woord van dank geven.

Ik wil mijn stagebedrijf Cubigo bedanken, om mij een kans te geven mijn schoolcarrière te eindigen, in een aangenaam en leerzaam bedrijf. Binnen mijn stagebedrijf wil ik de focus vooral leggen op mijn stagebegeleider, Jeff Vaes die mij altijd hielp wanneer, ik ook maar het kleinste probleem of vraag had. Ook wil ik dhr. Ulrich Kelchtermans bedanken, om mij te helpen bij het nalezen van mijn eindwerk.

Daarnaast wil ik mijn hogeschoolpromotor, dhr. Wesley Hendrix, bedanken, voor de begeleiding doorheen de stageperiode. Bij eender welke vraag die ik over mijn eindwerk of onderzoek had, kon ik bij hem terecht.

Als laatste zou ik mijn ouders willen bedanken, voor het nakijken van mijn eindwerk en de steun die ze mij gaven doorheen mijn schoolloopbaan.

# Abstract

Cubigo maakt platformen dat het leven binnen gemeenschappen vergemakkelijkt. Op dit platform kunnen mensen binnen een gemeenschap, gebruik maken van verschillende diensten. Zo kan bijvoorbeeld, een personeelslid een activiteit plannen, en kunnen de bewoners zich hiervoor inschrijven. Een persoon kan via dit platform bijvoorbeeld ook het laatste nieuws bekijken, eten bestellen of transport aanvragen en tevens deze opvolgen. De beschikbare diensten zijn verschillend per gemeenschap.

Het doel van dit project is, het werk van de ontwikkelaars binnen Cubigo te vereenvoudigen. Momenteel moeten ontwikkelaars iedere nieuwe gemeenschap manueel toevoegen aan de hand van scripts. Wanneer er *binnenkort* een groot aantal gemeenschappen bijkomen, gaat dat te veel tijd in beslag nemen. Daarnaast is er ook nog geen manier voorzien, waarop de instellingen van een aangemaakte gemeenschap, aangepast kunnen worden.

Om dit probleem te voorkomen, is er een project ontstaan om een *Angular 7 webinterface* aan te maken. Daardoor kan men op een eenvoudige manier een nieuwe gemeenschap toevoegen. Tevens moet er nog een *webinterface* aangemaakt worden, waarop de instellingen kunnen aangepast worden door de administrator van een gemeenschap. Om deze *interfaces* te kunnen verbinden met de bestaande database, moeten er een aantal zaken toegevoegd worden in de bestaande ASP.NET backend. Naar deze backend worden dan aan de hand van REST *calls*, objecten doorgestuurd of opgevraagd. De backend gebruikt dan *Entity framework* om verbinding te maken met de *PostgreSQL database.*

Cubigo gebruikt nog altijd *Angular 7* als frontend *framework*. Hiermee is de vraag gesteld of het veranderen van frontend *framework* wel voordelig kan zijn voor Cubigo. Om hierop te kunnen antwoorden, zijn er nog een aantal zaken die men moet onderzoeken. Een belangrijke vraag is, kan Cubigo meer functionaliteit halen uit een ander *framework*?

In dit onderzoek worden de drie meest gebruikte *frameworks* onderzocht, namelijk *Angular 7*, *Vue 3* en *React*. Aan de hand van deze *frameworks* worden er verschillende tests uitgevoerd op snelheid, functionaliteit, bruikbaarheid en meer. Aan het resultaat van dit onderzoek kan dan nagegaan worden of het nuttig is voor Cubigo om naar de toekomst toe over te schakelen naar een ander *framework*.

# Inhoudsopgave

## Inhoud

# Lijst van gebruikte figuren

# Lijst van gebruikte tabellen

# Lijst van gebruikte afkortingen

API: Application Programming Interface

CLI: Command Line Interface

CSS: Cascading Style Sheet

CV: Curriculum Vitae

DOM: Document Object Model

GWT: Given When Then

HTML: HyperText Markup Language

JSX: Javascript XML

LINQ: Language Integrated Query

MVC: Model View Controller

REST: REpresentational State Transfer

SCSS: Sassy Cascading Style Sheets

SPA: Single Page Application

SQL: Structured Query Language

UI: User Interface

VS: Verenigde Staten

XML: Extensible Markup Language

# Inleiding

Mijn stageopdracht volbracht ik in het bedrijf Cubigo. Bij Cubigo maakt men een online platform dat het leven binnen gemeenschappen vergemakkelijkt. Op dit platform kunnen mensen binnen een gemeenschap, van verschillende diensten gebruik maken. Zo kan bijvoorbeeld een bewoner, zich inschrijven voor een activiteit die ingepland wordt door een personeelslid. Op dit platform is het evenzeer mogelijk om eten te bestellen, transport aan te vragen en tal van andere nuttige zaken.

Het probleem dat er zich momenteel stelt met dit systeem, is dat er geen gemakkelijke manier bestaat om een nieuwe gemeenschap aan te maken. Hierdoor is de opdracht ontstaan om een webapplicatie te bouwen, waardoor het managementteam binnen Cubigo de mogelijkheid heeft om nieuwe gemeenschappen aan te maken.

Na het voltooien van deze webapplicatie is men met het ontwerpen van de bewerk functionaliteit gestart. Door middel van een nieuwe webpagina is er nu wel de mogelijkheid gecreëerd om bestaande gemeenschappen aan te passen.

Binnen Cubigo gebruikt men momenteel Angular 7 als frontend *framework*. Hierdoor is de vraag ontstaan, of het veranderen van frontend *framework* wel voordelig kan zijn voor Cubigo. Om hierop een antwoord te vinden, is men een aantal zaken gaan onderzoeken.

Men begon met een uitgebreide literatuurstudie, om te ontdekken welke voor-en nadelen verschillende *frameworks* hebben. Vervolgens heeft is er een selectie van drie *frameworks* gemaakt en ben deze dieper met elkaar gaan vergelijken.

Ten slotte zijn er een aantal proefopstellingen gemaakt, om de drie *frameworks* het tegen elkaar op te laten nemen. Door te experimenteren met deze proeven, zijn er verschillende zaken uitgetest, om zo de nodige conclusies te trekken.

# I. Stageverslag

# Bedrijfsvoorstelling

## 1.1 Cubigo

Cubigo is een van de langst gevestigde *software-providers* voor senioren in Europa en de VS. Alles begon bijna tien jaar geleden als een *spin-off* van een Belgische universiteit. De missie was om het leven van ouderen te verbeteren, door gebruik te maken van de nieuwste technologieën. Na een grote gebruikersbasis in Europa te hebben verworven (meer dan 35.000 senioren gebruiken Cubigo), werden drie jaar geleden de eerste stappen op de Amerikaanse markt geleden genomen.

Gedurende een jaar woonde en werkte een van de Cubigo-medewerkers in een Senior Living Community in Californië. Deze unieke, mensgerichte aanpak heeft niet alleen geholpen bij het herontwerpen van een product, dat gebaseerd is op de behoeften van de gebruikers. Maar heeft ook uitgebreid onderzoek naar de implementatie van nieuwe technologieën in *Senior Living Communities* mogelijk gemaakt.

Na verschillende iteraties en herontwerpen van dit baanbrekende community product, werd het in 2018 gelanceerd in twee referentie gemeenschappen. In combinatie met een innovatieve aanpak voor training en uitrol, bleken beide gemeenschappen een groot succes.

Het Cubigo-team bestaat uit een 30-tal mensen en het leiderschap heeft meer dan 100 jaren gecombineerde expertise in de Senior Living sector.

# Stageopdracht

## 1.2 Probleemstelling

Momenteel worden nieuwe klanten bij Cubigo manueel toegevoegd in hun applicatie. Een ontwikkelaar binnen Cubigo voegt deze nog steeds manueel toe aan de hand van scripts. Als er in de nabije toekomst meer klanten bijkomen, neemt dit te veel tijd in beslag.

Om dit probleem op te lossen is er de opdracht ontstaan om dit proces te vereenvoudigen door middel van een *Angular web interface*. Aan de hand van deze webpagina kan men dan op een eenvoudige manier een nieuwe gemeenschap toevoegen. Hierna is het de bedoeling dat er door middel van een andere webpagina, ook eenvoudig de instellingen per gemeenschap beheerd worden. Naar de toekomst toe is het ook de bedoeling dat de gemeenschappen zelf hun eigen instellingen kunnen bepalen. Zo kunnen ze dan bijvoorbeeld hun icoontje of *banner* zelf aanpassen.

## 1.3 Doelstellingen

De stageopdracht bestaat uit twee delen. Het eerste deel, wat ook de hoogste prioriteit heeft, is de interface waarmee een nieuwe gemeenschap toegevoegd kan worden.

Om dit te realiseren gaan er eerst een aantal API-calls toegevoegd moeten worden aan de bestaande ASP.NET backend. Deze backend gaat dan ook zorgen voor de connectie met de database aan de hand van *Entity Framework*. Hierna moet er een webpagina gemaakt worden in Angular waarop de backend aangeroepen kan worden. Deze webpagina moet ervoor zorgen dat er op een eenvoudige en snelle manier een nieuwe gemeenschap kan worden aangemaakt.

Als tweede deel van de opdracht wordt er een webpagina ontworpen, waarop de instellingen van een bestaande gemeenschap op een eenvoudige manier beheerd kunnen worden.

Om dit gedeelte te realiseren moeten er verschillende zaken toegevoegd worden aan de backend waarmee men met simpele *API-calls,* instellingen kan wijzigen. Vervolgens wordt er een eenvoudig te gebruiken *web interface* gemaakt. Aan de hand van deze *web interface* passen administrators van een gemeenschap, dan op een eenvoudige manier de instellingen van hun gemeenschap aan.

## 1.4 Omgeving

Binnen Cubigo gebruikt men verspillende technologieën om hun product zo perfect mogelijk te maken. Ze maken ook altijd gebruik van de meest recente versies van deze verschillende *frameworks* en bibliotheken. Om de stageopdracht te realiseren, wordt er gebruik van verschillende technologieën. Deze zorgen ervoor dat er een afgewerkt product geleverd kan worden.

### 1.4.1 Angular 7

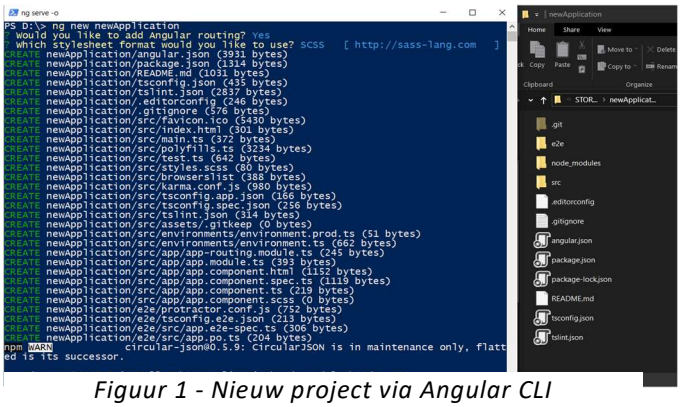
*Angular*, vaak ook beschreven als *Angular 2+,* is een *open source webapplicatie-framework* ontwikkeld door Google. Dit *framework* wordt momenteel onderhouden door een groep van mensen binnen de *Angular* gemeenschap en door Google zelf [1].

Binnen een *Angular-applicatie* wordt er gebruik gemaakt van *Typescript* in plaats van *Javascript*. Door deze taal te gebruiken zal het wat moeilijker worden voor ontwikkelaars die overstappen van andere frontend *frameworks*. Het voordeel van *Typescript*, is dat het duidelijker is om te schrijven. En het ook mogelijk maakt om objectgeoriënteerd te programmeren [2].

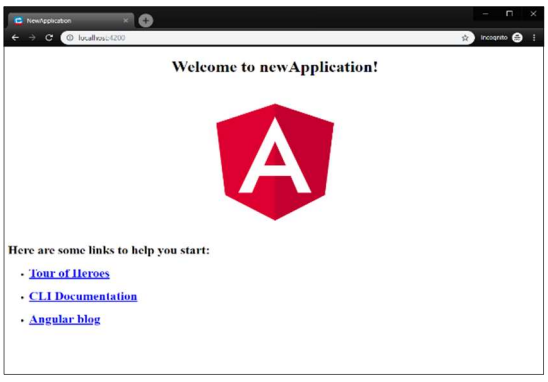
Bij Angular is elke applicatie die gemaakt wordt een *single-page* webapplicatie, dit wil zeggen dat de hele website nooit moet veranderen van webpagina. Wanneer er naar een andere pagina wordt genavigeerd zal dit alleen de veranderde elementen inladen. Dit heeft als voordeel dat de laadtijden op zo een website veel minder lang duren. Om dit te laten werken maakt Angular gebruik van zowel de *browser thread* als de *service worker thread*. Hierdoor kan de *browser thread* gebruikt worden om alleen de DOM (*Document Object Model*) te laden en de *service worker thread* voor alle rest. Door deze manier van renderen toe te passen gaat het inladen van de DOM heel snel bij Angular [3].

### 1.4.2 Angular CLI

Bij Angular heeft men de optie om een *command line interface* (*CLI*) te downloaden waardoor het maken van webapplicaties, intuïtiever en sneller kan. Een CLI is een *command-line interface* waarmee allerlei functies opgeroepen worden, om bijvoorbeeld, het realiseren van een nieuw project, automatisch te laten gebeuren. Hieronder ziet u een voorbeeld van het CLI-commando om een nieuw project aan te maken, en het geleverde resultaat.




*Figuur 1 - Nieuw project via Angular CLI*




*Figuur 2 - Nieuw project resultaat*

Bij Angular biedt deze CLI veel meer mogelijkheden dan alleen het creëren van een nieuw project. Zo kan ook aan de hand van het commando “ng generate component” een nieuwe component aangemaakt worden waardoor er geen tijd verloren gaat met het aanmaken van zo een component. Deze commando’s bestaan er ook voor het opmaken van *services, pipes, classes* en meer. Zonder zo een CLI moeten, al deze zaken manueel toegevoegd worden, bij het aanmaken van een nieuw project. [4]

### 1.4.3 ASP.NET

ASP.NET is een webapplicatie-*framework* is ontworpen, om websites, webapplicaties en webservices eenvoudiger te kunnen creëren. Binnen Cubigo wordt dit *framework* gebruikt om de *Angular applicatie* in uit te voeren. Hierdoor wordt de thuispagina op de server al gerenderd, en laadt deze veel sneller in wanneer iemand de website bezoekt.

Ook de API van Cubigo draait op het *ASP.NET framework* waardoor er op een simpele manier *REST* (*REpresentational State Transfer*) *calls* gemaakt kunnen worden. Dit wordt gedaan, door in de frontend een *REST call* te doen naar deze ASP.NET backend. Deze komt aan in de controller en zal zo de juiste services aanspreken en nodige data terugsturen. Het is ook mogelijk om data naar de ASP.NET backend te sturen, deze zal die data dan verwerken en opslaan in de database.

### 1.4.4 Entity *Framework*

*ASP.NET* backend maakt gebruik van *Entity Framework*. Dit is een *framework* ontworpen, om verbinding met een database te vergemakkelijken. Bij een traditionele database en backend connectie moet de database apart opgemaakt worden en wordt de data opgevraagd aan de hand van *SQL-query's*. Bij *Entity framework* moeten alleen de modellen gemaakt worden in de backend, en deze met elkaar verbinden. Vervolgens zal *Entity framework* aan de hand van deze modellen, automatisch de SQL genereren, om de database te maken.

Met *Entity framework* is het ook mogelijk om op een eenvoudige manier, data uit de database te halen. Zonder dit *framework* moet voor alles wat uit een database gehaald wordt, een query geschreven worden. Bij *Entity framework* wordt dit gedaan door middel van van *LINQ-query's*. Op de afbeeldingen hieronder, ziet u de vergelijking tussen een traditionele *SQL-query* en een *LINQ-query.*

```
var query =
    from c in db.Customers
    where c.Name.StartsWith ("A")
    orderby c.Name
    select c.Name.ToUpper();

var thirdPage = query.Skip(20).Take(10);
```

*Figuur 3 - LINQ-query*

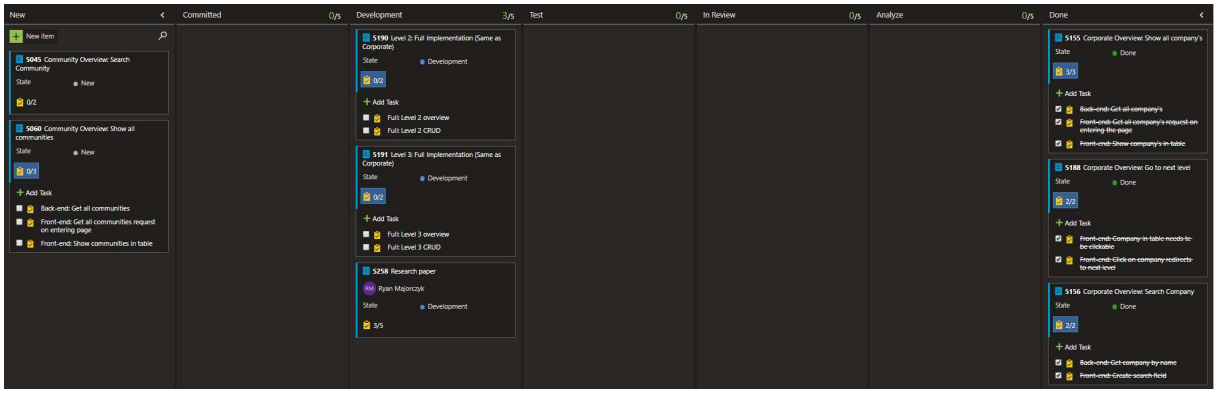
```
SELECT TOP 10 UPPER (c1.Name)
FROM Customer c1
WHERE
    c1.Name LIKE 'A%'
    AND c1.ID NOT IN
    (
        SELECT TOP 20 c2.ID
        FROM Customer c2
        WHERE c2.Name LIKE 'A%'
        ORDER BY c2.Name
    )
ORDER BY c1.Name
```

*Figuur 4 - SQL-query*

### 1.4.5 Azure DevOps

Binnen bedrijven zijn er altijd verschillende afdelingen die met elkaar moeten kunnen samenwerken en communiceren. Binnen IT-bedrijven zijn dit meestal, de ontwikkelaars en het managementteam. Om de communicatie tussen deze verschillende afdelingen zo vlot mogelijk te laten verlopen maken ze binnen Cubigo gebruik van *Azure DevOps*.

Binnen dit online platform kunnen de medewerkers van het managementteam, verschillende taken aanmaken die nog uitgevoerd moeten worden. Deze taken kunnen dan opgesplitst worden in verscheidene sprints van één of meerdere weken. De ontwikkelaars kunnen zich dan taken toe-eigenen en tegelijkertijd uitvoeren. Op dit platform kunnen ze via een *board* de status van hun taak aanpassen zoals op onderstaande afbeelding te zien is.



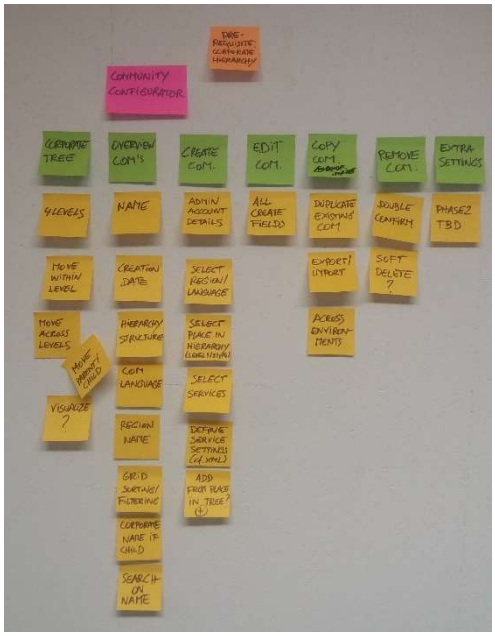

*Figuur 5 - Aanmaken story*

Aan de hand van dit *board,* kan iedereen binnen het bedrijf de status opvolgen, van de verschillende taken die men nog moet uitvoeren. Naar de toekomst toe wil Cubigo meer services gebruiken die Azure te bieden heeft, zoals de *repositories* en *pipelines*.

## 1.5 Uitwerking stageopdracht

### 1.5.1 Analyse

De eerste twee weken van de stageopdracht zijn voorzien om de bestaande doelstellingen te analyseren, om hier de *use cases* uit te halen. De eerste week begint meteen met een meeting, om deze doelstellingen te bespreken en te groeperen. Uit deze meeting zijn onderstaande *stories* ontstaan.




*Figuur 6 - Eerste user stories*

Aan de hand van deze *stories* is men kunnen beginnen met het uitschrijven van de specifieke taken aan de hand van het *Given When Then* principe. Bij het GWT-principe moet er bij iedere taak de acceptatiecriteria uitgeschreven worden. Door deze manier van werken ben je verplicht om elk mogelijk scenario na te gaan, waardoor het dan ook moeilijker wordt om delen te vergeten. Hieronder bevinden zich enkele voorbeelden van acceptatiecriteria, die uitgeschreven zijn voor het aanmaken van een nieuwe gemeenschap.

**Given** *I am a Cubigo staff member*
***When*** *I am on the Add Community page*
*AND I add a new community*
**Then** *The backend checks the data*

**Given** *I am a Cubigo staff member*
***When*** *I am on the Add Community page*
*AND I add a new community with all correct data*
**Then** *The backend checks the data*
*AND adds the community to the database*

*Figuur 7 - GWT 1*

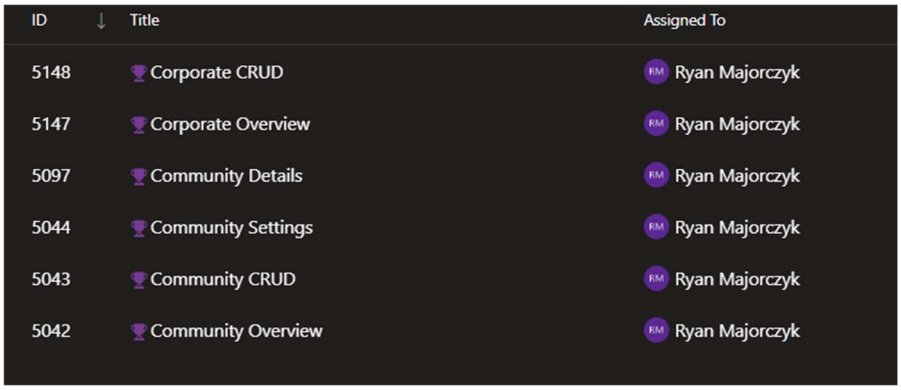
**Given** *I am a Cubigo staff member*
***When*** *I am on the Add Community page*
*AND I add a new community with missing data*
**Then** *The backend checks the data*
*AND gives back a missing data error*

**Given** *I am a Cubigo staff member*
***When*** *I am on the Add Community page*
*AND I add a new community with faulty data*
**Then** *The backend checks the data*
*AND gives back an incorrect data error*

*Figuur 8 - GWT 2*

Aan de hand van deze acceptatiecriteria kreeg het bedrijf beter zicht op wat er allemaal nodig was, om dit project tot een goed eind te brengen. Met deze criteria is men dan begonnen met het toevoegen van *stories* in *Azure DevOps*.

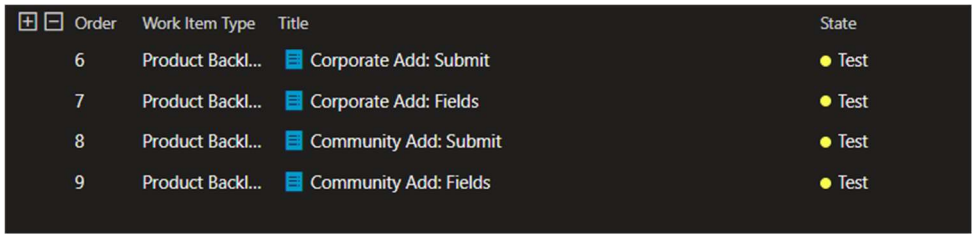
Bij het aanmaken van *stories* gebruikt men binnen Cubigo drie niveaus, namelijk*, features, stories* en *tasks* waarbij *features* het hoogste niveau is. Bij features wordt de functionaliteit van de applicatie beschreven op een aanzienlijk niveau. Hierdoor kan het project opgesplitst worden in verschillende samenhangende delen.



| ID | Title | Assigned To |
|---|---|---|
| 5148 | Corporate CRUD | Ryan Majorczyk |
| 5147 | Corporate Overview | Ryan Majorczyk |
| 5097 | Community Details | Ryan Majorczyk |
| 5044 | Community Settings | Ryan Majorczyk |
| 5043 | Community CRUD | Ryan Majorczyk |
| 5042 | Community Overview | Ryan Majorczyk |

*Figuur 9 - Features*

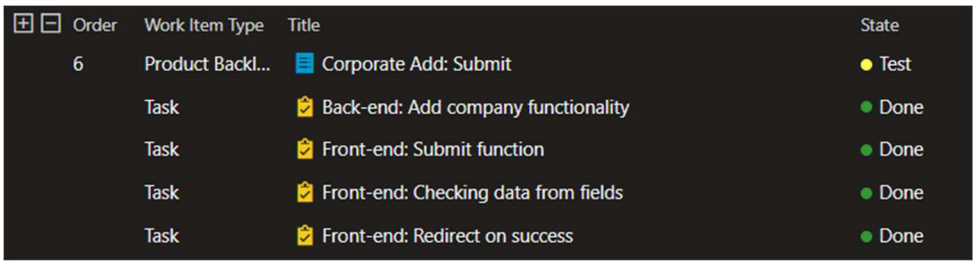
Nadien kunnen er onder deze *features, stories* aangemaakt worden. Deze bevatten meer gedetailleerde beschrijvingen over de functionaliteit van de applicatie. Bij een *story* is het ook de bedoeling, om de tijd te schatten die er nodig is om deze af te werken. Bij Cubigo doen ze dit aan de hand van *story points* die van één tot dertien gaan. Hoe meer *points* een *story* heeft hoe meer tijd er nodig is om deze te implementeren.



| Order | Work Item Type | Title | State |
|---|---|---|---|
| 6 | Product Backl... | Corporate Add: Submit | Test |
| 7 | Product Backl... | Corporate Add: Fields | Test |
| 8 | Product Backl... | Community Add: Submit | Test |
| 9 | Product Backl... | Community Add: Fields | Test |

*Figuur 10 - Stories*

Aan de hand van de acceptatiecriteria die eerder al uitgeschreven waren, kon ik op een eenvoudige manier de verschillende *tasks* afleiden. De bedoeling van een *task* is, een *story* te verdelen in kleinere delen die rechtstreeks geïmplementeerd moeten worden.



| Order | Work Item Type | Title | State |
|---|---|---|---|
| 6 | Product Backl... | Corporate Add: Submit | Test |
| | Task | Back-end: Add company functionality | Done |
| | Task | Front-end: Submit function | Done |
| | Task | Front-end: Checking data from fields | Done |
| | Task | Front-end: Redirect on success | Done |

*Figuur 11 - Taken*

### 1.5.2 Opstart

De eerste week van de stage diende voornamelijk voor de setup van het project, het bedrijf beter leren kennen, het installeren van de nodige software en het doornemen van de backend code waarin gewerkt moest worden. Tijdens meerdere meetings en demo's is het product en de geschiedenis van Cubigo uiteengezet.

In de tweede week was het de opdracht om aan de hand van de gegeven opdracht, de verschillende *stories* uit te schrijven en in Azure DevOps te plaatsen. Deze werden ingedeeld in sprints, en allemaal ingeschat hoe lang het zou duren om ze te implementeren.

Vervolgens werd er gestart met het aanmaken van een nieuw Angular project. En zo werden de eerste pagina's aangemaakt voor de creatie functionaliteit.

### 1.5.3 Delen van de opdracht

De stageopdracht bestaat uit drie grote delen, namelijk het toevoegen van een nieuwe gemeenschap, het bewerken van een bestaande gemeenschap en een overzicht krijgen van bestaande gemeenschappen.

### 1.5.4 Nieuwe gemeenschap toevoegen

Het eerste deel waaraan gewerkt is, is het toevoegen van een nieuwe gemeenschap. Tot heden werden deze gemeenschappen door ontwikkelaars binnen Cubigo toegevoerd aan de hand van XML-bestanden die uitgelezen werden in een *Script*. Door de groei van het bedrijf moeten er meer en meer gemeenschappen aangemaakt worden en is dit niet meer doenbaar op deze manier. Hieronder bevindt zich een deel van een voorbeeld van de huidige manier om een gemeenschap aan te maken.

```
<CommunityDetailsSection Name="Cubigo QA" ShowContextSwitch="false" ParentCommunityId="9e0e1a43-90c6-4b54-a82a-b42bdd782345">
  <add Name="Info" />
  <add Name="Organisation" />
  <add Name="Dining" />
  <add Name="Activities" />
  <add Name="Maintenance" />
  <add Name="Transportation" />
  <add Name="Valet" />
</CommunityDetailsSection>
<AdminDetailsSection Email="admin+cugigoqa@cubigo.com" Password="CubigoQA" FirstName="admin" LastName="Cubigo QA" GenderType="1" CellPhone="" Phone="" />
<RegionalSection Language="en-US" Timezone="America/Los_Angeles" Currency="USD" />
<DiningSection GuestsAllowed="true" PerOrderGuestSurcharge="0" PerMealGuestSurcharge="0" FreeAccompaniments="0" MaxAccompaniments="99" />
<ActivitiesSection>
  <Locations>
    <add Name="Room 1" Description="Room 1" />
    <add Name="Room 2" Description="Room 2" />
    <add Name="Room 3" Description="Room 3" />
    <add Name="Room 4" Description="Room 4" />
  </Locations>
</ActivitiesSection>
<TransportationSection ReturnTime="true">
  <MobilityAids>
    <add Name="None" Description="None" />
    <add Name="Walker" Description="Walker" />
    <add Name="Scooter" Description="Scooter" />
    <add Name="Wheelchair" Description="Wheelchair" />
  </MobilityAids>
  <ExtraHelps>
    <add Name="I need a seat transfer" Description="seat transfer" />
    <add Name="I require escort" Description="escort" />
  </ExtraHelps>
  <BillingReasons>
    <add Name="Other ($0)" Cost="0" Unit="1" IsOutsideRegularHours="false" Description="Other" />
  </BillingReasons>
</TransportationSection>
```

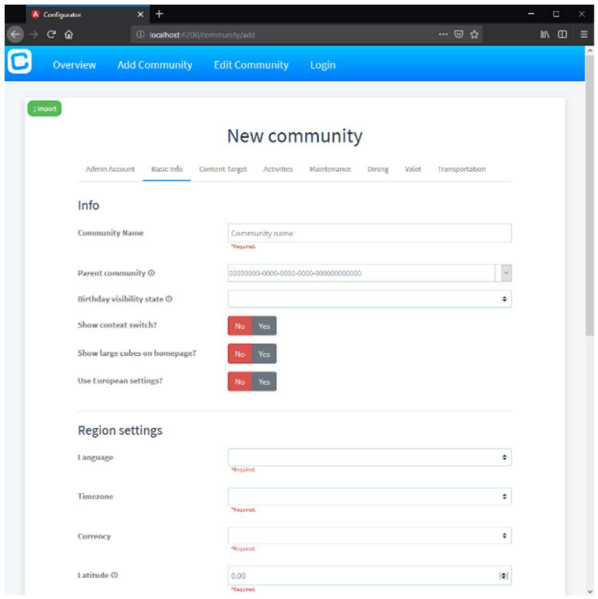
*Figuur 12 - Aanmaak script*

De opzet is, dat iedereen binnen het bedrijf, zowel de ontwikkelaars als de mensen van het productieteam, aan de hand van een webapplicatie een nieuwe gemeenschap kunnen toevoegen. Voor dit op een zo eenvoudig mogelijke manier te laten gebeuren, is er een *creation-wizard* gemaakt waar door middel van velden op te vullen, een gemeenschap aangemaakt kan worden.

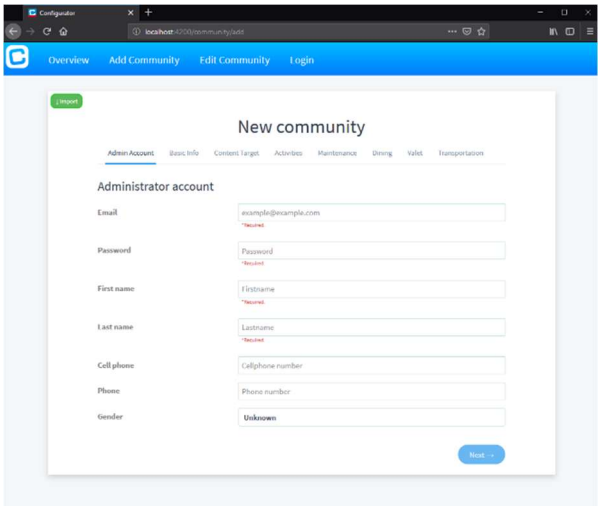
### 1.5.4.1 Aanmaak wizard

De wizard om een nieuwe gemeenschap aan te maken bevat een aantal grote delen. Het eerste is een administrator account, waarmee men kan inloggen [zie figuur 16]. Dit account heeft een aantal vereiste velden die ingevuld worden, om zo opgeslagen te kunnen worden in de database.

Vervolgens, het instellen van basisinformatie zoals de naam van de gemeenschap, branding opties en meer. Er zijn hier ook al een aantal instellingen die ingesteld kunnen worden. Hier kan ook een *parent* gemeenschap gekozen worden, die ervoor zorgt dat er overerving kan gebeuren, voor instellingen die leeg gelaten worden of die nog niet in de wizard zitten.



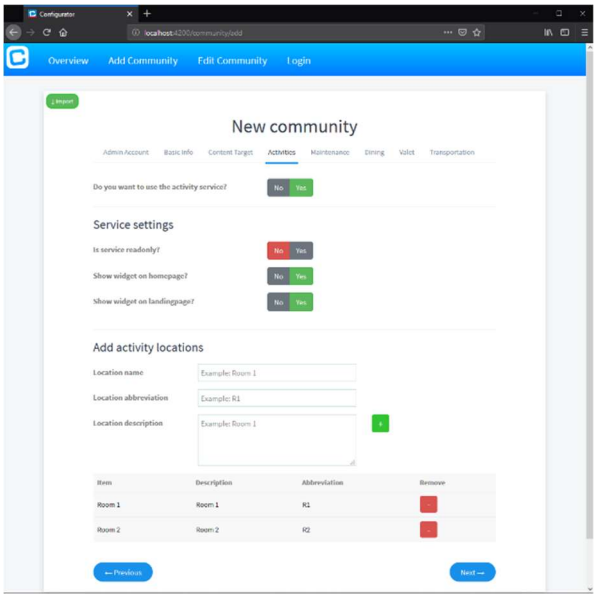

*Figuur 14 - Aanmaken basisinformatie*



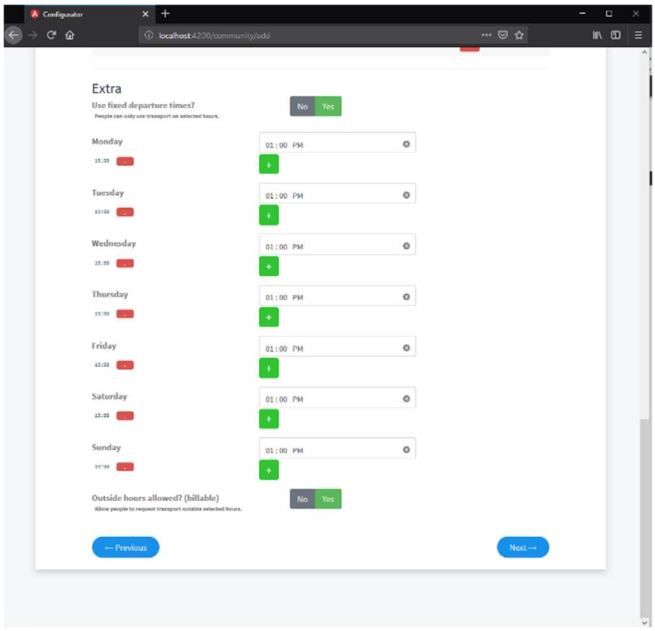

*Figuur 13 - Aanmaken administrator account*

Het volgende grote deel, zijn de verschillende services die ingesteld kunnen worden. Cubigo biedt op dit moment activiteiten, onderhoud, eten en transportatie. Deze services hebben allemaal hun eigen unieke instellingen. Bij activiteiten stelt men in, in welke kamers activiteiten kunnen plaatsvinden en bij transportatie op welke uren dit kan aangevraagd worden. Ook zijn er een aantal instellingen die voor elke service hetzelfde zijn.

Het is niet altijd dat een gemeenschap, al deze services aanbiedt aan de bewoners. Hiermee is het mogelijk om in de wizard bepaalde services niet aan te klikken. Deze zullen dan bij het aanmaken van de gemeenschap genegeerd worden en niet beschikbaar zijn bij die bepaalde gemeenschap.

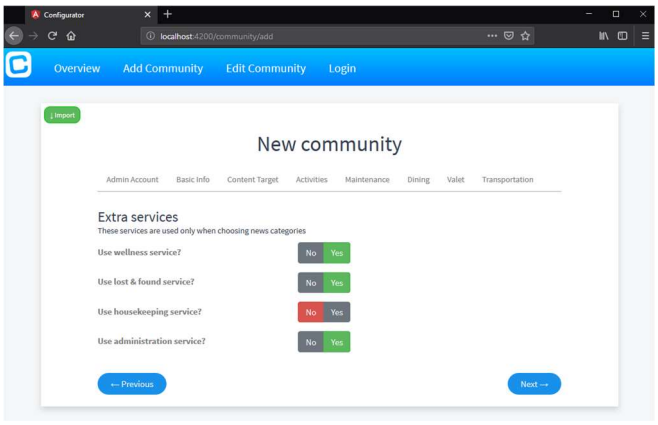



*Figuur 15 - Aanmaken activiteiten*

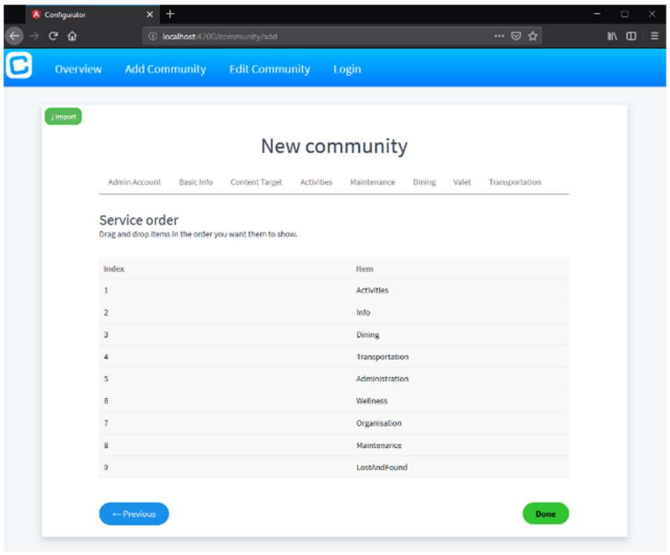



*Figuur 16 - Aanmaken transportatie*

Als laatste zijn er nog een aantal extra services die aan of uit gezet kunnen worden. Deze worden alleen gebruikt bij het publiceren van nieuws, om zo aan te tonen wat het thema is. Vervolgens kan de volgorde van de services nog bepaald worden. Deze volgorde is belangrijk omdat deze gelinkt staan op de hoofdpagina van het Cubigo platform.




*Figuur 17 - Extra services*




*Figuur 18 - Service vollegorde*

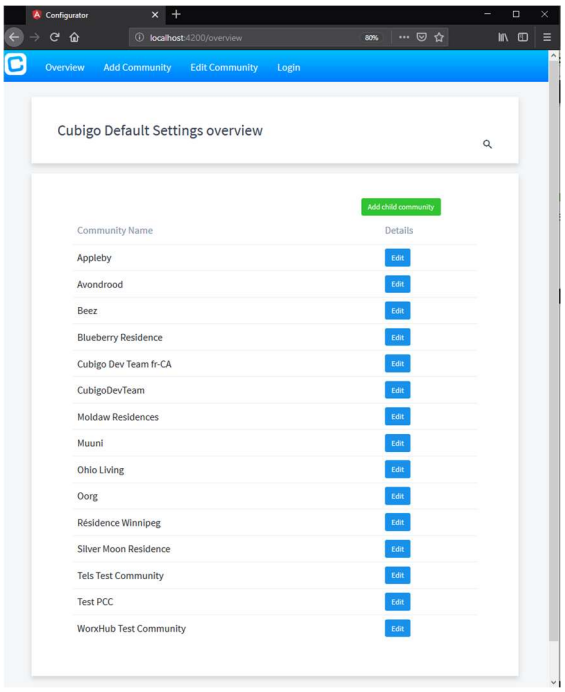
Na het doorlopen van de volledige wizard worden alle data samengenomen, en zo naar de backend gestuurd. Deze zal dan de nodige controles uitvoeren en zo de gemeenschap aanmaken. Voor alle instellingen die leeg gelaten zijn, zoals bevoordeeld locaties voor bepaalde activiteiten, gaat er aan overerving gedaan worden van de *parent* gemeenschap. De data worden niet gekopieerd, maar bij het aanvragen van die data, moet men dan een niveau dieper gaan om de data op te halen.

### 1.5.5 Overzicht van gemeenschappen

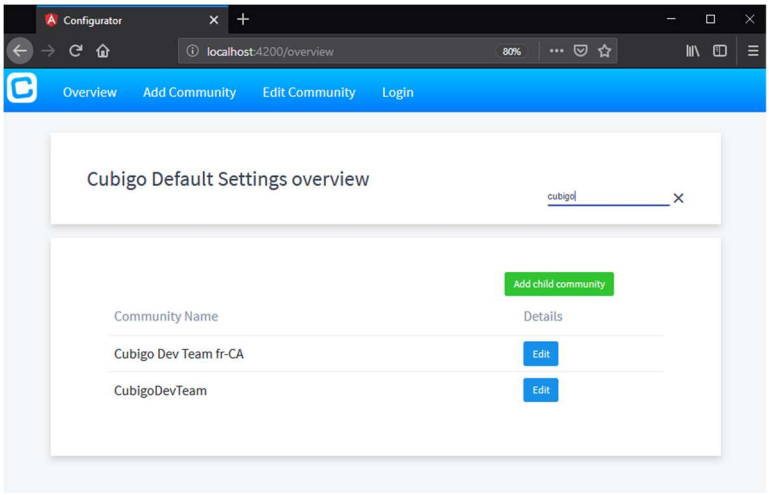
Het volgende deel van de stageopdracht was het maken van een overzicht van de verschillende gemeenschappen. Dit was een taak die op het eerste zicht eenvoudig leek, maar waar er toch heel wat uitdagingen aan verbonden waren. Het probleem hierbij was dat er gewerkt moest worden met de verschillende niveaus waarin een gemeenschap zich kan bevinden. Tot heden is het nog niet nodig geweest om communities aan te maken met verschillende niveaus. Dit gaat pas echt nodig zijn wanneer er zich een klant aanbiedt die meerdere gemeenschappen heeft. Wat eind juni ook het geval gaat zijn.

Vooraleer hieraan begonnen wordt, is er een meeting met de UX-designer binnen Cubigo. Samen met hem wordt er gekeken naar hoe die boomstructuur op zo een eenvoudig mogelijke manier getoond kan worden. Uiteindelijk is men tot een akkoord gekomen, om te werken met pagina's die altijd een niveau dieper gaan wanneer er op een gemeenschap wordt geklikt. Ook bestaat er nu een optie, om de volledige boomstructuur te tonen op één enkele pagina. Dit heeft als voordeel dat het nu heel overzichtelijk is welke gemeenschap overerft van welke, maar het nadeel is, dat de snelheid sterk achteruitgaat wanneer er later honderden gemeenschappen ingeladen worden.

Als men de pagina opent, worden er 15 gemeenschappen ingeladen. Maar het is ook mogelijk om te zoeken naar gemeenschappen op dat niveau. Met de "Add child community" knop gaat men via een omweg, naar de "add community" pagina, waarin de *parent-id* al wordt ingevuld. Op deze kunnen er later snel gemeenschappen aangemaakt worden zonder handmatig te moeten zoeken naar de correcte parent.




*Figuur 19 - Overzicht van gemeenschappen*




*Figuur 20 - Overzicht zoekfunctionaliteit*

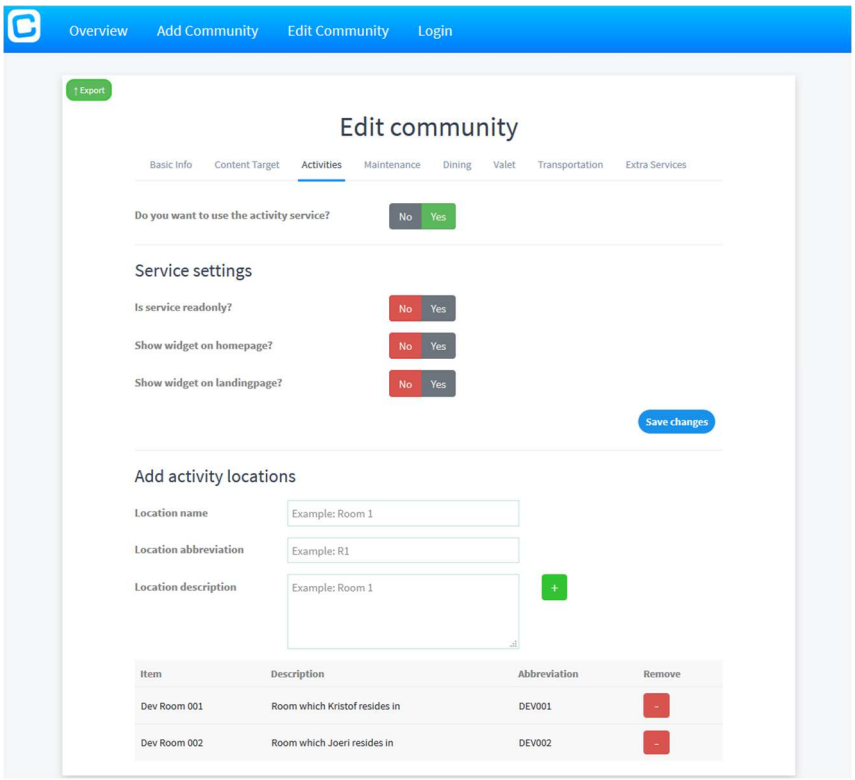
Via de "Edit" knop wordt er een omleiding gedaan naar de *edit*-pagina waarop je alle instellingen kan zien van een gemeenschap en deze kan bewerken. Het is dan ook mogelijk om de parent gemeenschap aan te passen zodat de structuur kan veranderen.

### 1.5.6 Bewerken van gemeenschappen

Voor het bewerken van gemeenschappen is er binnen Cubigo geen enkele functionaliteit voorzien. Voor alle aanpassingen moet een ontwikkelaar die objecten manueel in de database gaan zoeken, en zelf de waardes aanpassen. Dit neemt veel tijd in beslag indien er veel verschillende aanpassingen moeten gebeuren.

Het derde deel van de opdracht is een oplossing zoeken, om gemeenschappen aan te passen via een webpagina in de *configurator*. In deze pagina moeten dezelfde instellingen aanpasbaar zijn die ingesteld zijn bij het aanmaken van een gemeenschap. Om dit op een zo eenvoudig mogelijke manier op te bouwen zijn de *create* pagina's hergebruikt. Op deze manier zal er bij het toevoegen van een instelling bij de "create pagina" deze ook meteen bij het bewerken geïmplementeerd zijn.

Het grote verschil zit zich erin dat bij het aanmaken van een gemeenschap er gebruik gemaakt wordt van een wizard systeem en bij het bewerken van een gemeenschap kan er genavigeerd worden tussen de verschillende tabs.




*Figuur 21 - Bewerk voorbeeld*

Wanneer er aanpassingen gemaakt zijn kan er aan de hand van de "save changes" knop een update gestuurd worden naar de backend. Deze zal de data controleren en de data in de database aanpassen.

In het geval van lijsten is het niet mogelijk om met een updateknop te werken. Dit komt doordat er dan altijd nagegaan zou moeten worden of er iets veranderd is, in die lijst die doorgestuurd wordt. Dit zou veel bereken werk zijn voor de backend omdat die niet alleen moet kijken of er iets is toegevoegd maar ook of er iets verwijderd is. Om dit op te lossen wordt er bij het toevoegen van een item meteen een call naar de backend gestuurd om deze toe te voegen. Hierna wordt de lijst opnieuw binnengehaald om de juiste identificatie van de items te hebben. Aan de hand van deze identificatie kan een item ook verwijderd worden.

Er zijn een aantal zaken die bij het bewerk gedeelte niet zijn opgenomen, namelijk het administrator account, de vollegorde van de services en de “*content targets*”. Dit is zo omdat het niet mogelijk is om de gegevens van deze zaken te bewerken. Bij het aanmaken van een gemeenschap worden deze vast gelegd en veranderingen zouden de gemeenschap niet meer laten functioneren.

### 1.5.7 Import en Export

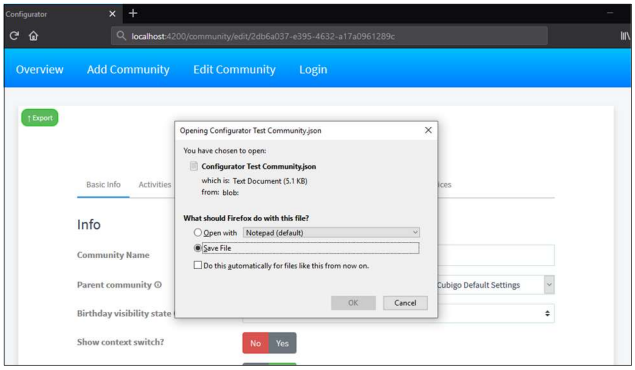
Naast de drie hoofdzaken die gemaakt worden, waren er een paar kleine extra’s. Twee hiervan zijn de import en export functionaliteit. Binnen Cubigo zijn er verschillende omgevingen waar hun applicatie op draait. Deze omgevingen hebben allemaal hun eigen doel, zo is er de *development* omgeving waarop ontwikkelaars de code die ze maken kunnen deployen en uittesten. Er is ook een acceptatie omgeving waarop de definitieve applicatie uitgevoerd wordt zodat deze getest kan worden door de tester. Als laatste is er de productie omgeving waar het eindproduct op uitgevoerd wordt, voor de gebruikers om te gebruiken.

De *configurator* gaat op deze verschillende omgevingen uitgevoerd worden om zowel test als echte gemeenschappen op te zetten. Een belangrijk aspect dat deze *configurator* mogelijk maakt is het overnemen van een gemeenschap binnen één omgeving naar een ander.

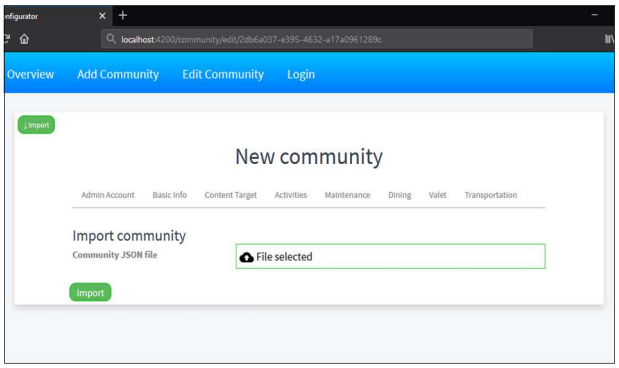
Omdat het niet mogelijk is om een verbinding te maken tussen deze verschillende omgevingen moet het importeren en exporteren op een andere manier gebeuren. De manier die gekozen was is om te werken met een “.json” bestand dat gegenereerd wordt met de data van een gemeenschap. Deze wordt dan gedownload en kan geïmporteerd worden op een andere omgeving.

Dankzij deze functionaliteit kan een gemeenschap aangemaakt worden op de development omgeving en gekeken worden of alles correct werkt. Als alles in orde is kan deze geëxporteerd worden en opnieuw aangemaakt in de productie omgeving. Hierdoor is er altijd een zekerheid dat er geen fouten gemaakt worden bij het aanmaken van een nieuwe gemeenschap.

Om een community te exporteren is er een knop voorzien in de *edit*-pagina. Wanneer er op de knop geduwd wordt zal een bestand gedownload worden met de nodige data om die gemeenschap opnieuw aan te maken. Dit bestand kan hierna geopend worden in de aanmaak pagina op een omgeving naar keuze.




*Figuur 23 - Exporteer gemeenschap*




*Figuur 22 - Importeer gemeenschap*

### 1.5.8 Functionaliteit

In dit deel wordt er dieper ingegaan in de technische kant van zowel, de frontend als de backend en hoe ze samen één groot geheel vormen.

#### 1.5.8.1 Connectie tussen frontend en backend

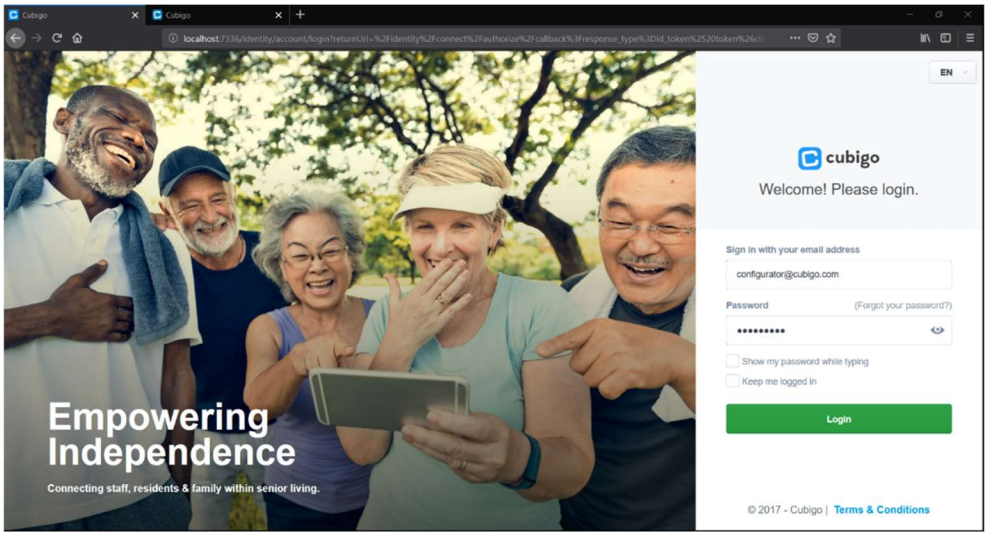
Het belangrijkste element van de applicatie is de connectie die bestaat, tussen de frontend en frontend. Het eerste deel van de opdracht bestond uit het toevoegen van een nieuwe gemeenschap. Bij het invullen van de wizard wordt er op het einde achterliggend de waardes naar de frontend gestuurd. De frontend gaat deze data verwerken, en omzetten naar een object waar het mee kan werken.

Om dit te doen wordt er in de frontend gebruik gemaakt van de *httpclient*, die ingebouwd zit in ASP.NET. Aan de hand van een aangegeven route kan er een *http-call* gedaan worden die de frontend kan ontvangen. Hieronder bevindt zich een voorbeeld van een *API-call in ASP.NET*.

```
[HttpPut]
[Route("communities/add")]
public async Task AddCommunity([FromBody] CreateCommunityModel communityModel)
    => await _communityCreateService.CreateCommunity(communityModel);
```

#### 1.5.8.2 Authenticatie

Voor de authenticatie in de webapplicatie is er gebruikt gemaakt van de bestaande *identityserver* van Cubigo. De *Identityserver* maakt gebruik van oAuth2 authenticatie. Dit wil zeggen dat initieel de gebruikersnaam en het wachtwoord wordt gestuurd naar de server. De server geeft hierna een *token* terug die gebruikt wordt voor verdere authenticatie. Wanneer de gebruiker de webpagina opent wordt er eerst gekeken of deze is ingelogd. Wanneer dit niet het geval is zal deze omgeleid worden naar de loginpagina van de *identityserver.*




*Figuur 24 – Loginpagina*

Na het inloggen wordt de *token* teruggegeven aan de frontend. De frontend slaat deze dan op in de lokale opslag van de browser. Deze *tokens* zijn maar voor een bepaalde duur geldig. Binnen de Cubigo applicatie is dit één uur. Wanneer de *token* niet meer geldig is wordt deze achterliggend vernieuwd.

De token wordt telkens mee gestuurd met elke REST call die er gedaan wordt. Om ervoor te zorgen dat dit gebeurd is er een header interceptor voorzien in de frontend. Deze vangt alle calls op die uitgestuurd worden of die binnen komen. Op de calls die naar buiten gestuurd worden, wordt er een authenticatie header toegevoegd die de token bevat.

```
@Injectable()
export class HeaderInterceptor implements HttpInterceptor, OnDestroy {
    private communityUuidSubscription: Subscription;
    private cultureCodeSubscription: Subscription;

    constructor(
        private errorHandler: OAuthResourceServerErrorHandler
    ) {
    }

    ngOnDestroy(): void {
        if (this.communityUuidSubscription) {
            this.communityUuidSubscription.unsubscribe();
        }
        if (this.cultureCodeSubscription) {
            this.cultureCodeSubscription.unsubscribe();
        }
    }

    public intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
        const url = request.url.toLowerCase();
        const token = localStorage.getItem('access_token');
        const header = 'Bearer ' + token;
        const headers = request.headers.set('Authorization', header);

        request = request.clone({ headers });
        request = request.clone({ headers: request.headers.set('Accept', 'application/json') });

        return next.handle(request).pipe(
            catchError((err, caught) => {
                this.errorHandler.handleError(err);
                return caught;
            })
        );
    }
}
```

*Figuur 25 - Header interceptor*

In de backend wordt er bij elke binnenkomende *call* de *token* uit de *header* gehaald. Deze wordt dan naar de *identityserver* gestuurd om te valideren of deze correct is. Wanneer de validatie gelukt is zal de backend deze *call* verwerken. Bij een incorrecte *token* zal de backend een 401 (*Unauthorized*) error terugsturen naar de frontend.

### 1.5.8.3 Ontwerp

De UX-designer binnen Cubigo had enkele schetsen gemaakt van hoe de webapplicatie eruit moest zien. Aan de hand van deze designs is de website opgebouwd.

Om de website op te bouwen is ervan bootstrap gebruik gemaakt. Deze *library* zorgt ervoor dat er op een eenvoudige manier een website dynamisch opgebouwd kan worden. Dankzij de bootstrap *grids* werkt de applicatie op zowel grote computerschermen als op kleine mobile schermen.

Voor een aantal elementen in de webapplicatie is er gebruik gemaakt van *Material design*. Uit deze *library* komt de *drag and drop* die nodig was om de service vollegorde te bepalen.

Voor de rest van de styling van de applicatie is er gebruik gemaakt van SCSS (*Sassy CSS*) wat een uitbreiding is op het traditionele CSS (*Cascading Style Sheet*). Het grote verschil tussen de twee zit zich in de ordelijkheid. Bij traditionele CSS moet er veel dubbel geschreven worden en is het moeilijk om zaken die samen horen te groeperen.

```css
.example {
    background: #dd0120;
    width: 10px;
}

.example:hover {
    border: 1px solid #dd0120;
    width: 15px;
}

.example:focus {
    border: 2px solid #dd0120;
    width: 20px;
}
```

*Figuur 27 - CSS*

```scss
$red: #dd0120;

.example {
    background: $red;
    width: 10px;
    &:hover {
        border: 1px solid $red;
        width: 15px;
    }
    &:focus {
        border: 2px solid $red;
        width: 20px;
    }
}
```

*Figuur 26 - SCSS*

Hierboven is meteen zichtbaar hoe dat bij SCSS het hergebruiken van waardes zoals bijvoorbeeld kleuren veel eenvoudiger is. Bij CSS moet de kleur altijd opnieuw gezocht en gekopieerd worden. Nog een groot voordeel aan SCSS is dat het gewone CSS-code kan bevatten, wat het overschakelen vereenvoudigd.

## 1.6 Besluit

### 1.6.1 Reflectie stageopdracht

Het realiseren van mijn stageopdracht is zeker gelukt. Ik heb alle voorziene doelstellingen gehaald en heb zelfs een aantal uitbreidingen kunnen implementeren. Zo was het voorzien van authenticatie geen “must” omdat het een intern te gebruiken tool was. Ook was het voorzien van de *import* en *export* functionaliteit een mooi extraatje.

Tijdens de stageperiode zijn er ook heel wat problemen naar boven gekomen, die ik niet altijd zelf kon oplossen. Hiervoor kon ik terecht bij één van de ontwikkelaars binnen Cubigo. Deze hielpen mij altijd verder en gaven dan ook extra uitleg over de oplossing van het probleem.

Door deze stage heb ik veel kunnen bijleren, zowel technische als persoonlijke kennis. Op technisch gebied heb ik uitgebreid leren werken met Angular en ASP.NET. Ook heb mijn kennis rond het versiebeheer met Git kunnen uitbreiden. Daarnaast heb ik ook beter leren werken met het ticketsysteem in Azure DevOps en het leren schrijven van deze verschillende tickets.

Ik had altijd mijn best gedaan om zo kwalitatief mogelijke code te schrijven. Ik was mij er namelijk bewust van dat, binnenkort, andere ontwikkelaars hieraan verder zouden werken. Het schrijven van kwalitatieve code ging niet altijd even gemakkelijk. In de lessen van school kregen we hier wel uitleg rond maar er werd nooit echt de focus op gelegd. Samen met mijn collega’s binnen Cubigo heb ik mijn kennis hierrond kunnen uitbreiden.

Samengevat, Cubigo heeft mij een super stageperiode bezorgd, waar ik veel heb kunnen bijleren en een visie naar de toekomst heb meegekregen. Deze ervaring kan ik ook mee nemen naar de bedrijfswereld wanneer ik volgend jaar ga beginnen met werken.

### 1.6.2 Reflectie werkhouding

Dankzij de stage zijn er een aantal zaken die verbeterd zijn in mijn houding en functioneren. Zo had ik altijd veel moeite met iets af te werken alvorens ik aan iets nieuw begon. Deze eigenschap heeft een aantal problemen naar boven laten komen in mijn stage. Zo had ik soms moeite met het volgen van de planning die ik opgesteld had. Ik heb hier extra aandacht aan besteed en zag al de nodige vooruitgang tegen het einde van de stageperiode. Terwijl ik in het begin nooit een ticket volledig af werkte voordat ik aan een ander begin, werkte ik op het einde van de stage eerst de ticket volledig af.

Door de stageperiode was een ander probleem dat naar boven kwam dat ik dacht dat ik mensen stoorde door vragen te stellen. Hierdoor bleef ik soms te lang vasthangen bij problemen die eigenlijk eenvoudig op te lossen waren. Deze stage heeft ervoor gezorgd dat er al veel verbetering is gekomen rond deze negatieve houding. Naar de toekomst toe ga ik sneller proberen vragen te stellen wanneer ik vast zit met iets en ga niet bang zijn om mensen te “storen”.

Een positieve eigenschap die naar boven is gekomen is dat ik goed zelfstandig kan werken. Deze eigenschap hangt ook samen met het niet tijdig stellen van vragen. Ik probeer altijd eerst problemen op te lossen en mijn eigen ding te doen vooraleer ik vragen begin te stellen. Hierdoor heb ik ook altijd met een kritische blik naar mezelf kunnen kijken, waardoor ik kon zien of ik wel correct bezig was en waarom ik kon verbeteren.

### 1.6.3 Reflectie opleiding

Toen ik aan mijn stage begon had ik niet het gevoel dat ik hier helemaal klaar voor was. Op school hebben wij echter alleen de basis geleerd rond programmeren.

Al snel had ik door dat de kennis, die wij uit de lessen die we op school gekregen hadden, goed van pas kwamen tijdens de stageperiode. Het waren niet enkel de programmeer vakken maar ook de business vakken waar ik aan kon terugdenken en waar ik mijn kennis kon uithalen. Het maken van user *stories* binnen Azure DevOps was immers iets wat wij tijdens de lessen gezien hadden.

De verschillende programmeer vakken kon ik ook toepassen binnen mijn stageopdracht. Zo moest ik een ASP.NET backend maken, dit was iets wat we geleerd hadden in het vak “Programming Advanced” en een Angular frontend. Dankzij de lessen web en Angular ben ik snel kunnen beginnen aan het maken van de frontend.

Zelfs de lessen *communication skills* hebben mij geholpen alvorens de stage begon. In deze lessen heb ik leren solliciteren en mijn CV leren opstellen. Dankzij deze lessen is het mij gelukt om te kunnen beginnen in Cubigo ondanks dat er toch wat concurrentie was.

# II. Onderzoekstopic

# 2 Onderzoek

## 2.1 Probleemstelling

Bij Cubigo gebruiken ze sinds het begin de nieuwste Angular versies. Zo zijn ze begonnen met AngularJS voor het maken van hun eerste product. Toen Angular 2 uitgekomen was, hadden ze binnen Cubigo de keuze gemaakt om hiernaar over te schakelen. Om deze keuze te maken had men beroep gedaan op een werknemer, die de verschillende opties onderzocht had, die er indertijd op de markt waren.

Uit dat klein onderzoek was gebleken dat ze er het meeste voordeel uithaalden om bij Angular te blijven. Hoofdzakelijk omdat de ontwikkelaars hier al de meeste ervaring mee hadden, veel zaken waren bij het maken van Angular 2 afgeleid van AngularJS. Angular 2 bood toen ook de meeste functionaliteit die ze nodig hadden voor de eerste versies van hun platform. Ook was Angular beter om grote applicaties op te maken en React was eigenlijk meer ontworpen om snel kleine applicaties te maken.

Dit onderzoek gebeurde al een aantal jaren geleden en ondertussen heeft React al verschillende grote updates gehad die dit resultaat kunnen veranderen. Ook is er ondertussen een nieuwe grote speler op de markt gekomen, namelijk “Vue”.

Angular is niet altijd even performant voor het inladen van elementen en dat is soms te merken in de Cubigo applicatie. Ook zijn er veel functies toegevoegd aan het platform waar vroeger nog geen rekening mee gehouden kon worden. Op dit moment is er geen echt probleem met het huidige *framework* maar er is wel de vraag naar verbetering. Ook kan er gekeken worden of de keuze die er vroeger gemaakt is, wel de juiste was.

## 2.2 Onderzoeksvraag

*Wat is het beste frontend framework in 2019?*

Om deze vraag te kunnen beantwoorden moeten er verschillende vragen gesteld worden, namelijk:

- Welk *framework* biedt de meeste functionaliteit?
- Welk *framework* is het makkelijkst aan te leren?
- Welk *framework* is het meest performant?
- Zijn de *frameworks* bruikbaar binnen grote webapplicaties?

Omdat er zoveel verschillende Javascript-*frameworks* bestaan gaat er een vergelijking gemaakt worden tussen de drie grootste spelers op dit moment, namelijk “Angular“, “React” en “Vue”.

## 2.3 Onderzoeksmethode

Om het resultaat van het onderzoek zo correct mogelijk te maken, werd er een uitgebreide literatuurstudie gedaan. Aan de hand van deze studie kon er gezien worden welke onderzoeken er allemaal al uitgevoerd waren en wat hun bevindingen zijn. Uit dit onderzoek werd er ook afgeleid welk van de drie *frameworks* het meeste biedt op gebied van functionaliteit en of de *frameworks* bruikbaar zijn binnen grote applicaties.

Na de literatuurstudie werden er verschillende prototypes gemaakt rond de drie gekozen *frameworks*. Aan de hand van deze prototypes zijn er een aantal testen gedaan om de verschillende deelvragen zo goed mogelijk te kunnen beantwoorden en zo een resultaat te bekomen. Om te beginnen werden per *framework* drie dezelfde webpagina's gemaakt en gemeten welke van de drie *frameworks* het snelst aangeleerd kon worden. Hierna werd er aan de hand van deze pagina's een aantal snelheidstesten gedaan om te kijken welke van de *frameworks* het meest performant is.

De gekende resultaten werden hierna besproken. Aan de hand hiervan werden de aanbevelingen geformuleerd.

## 2.4 Literatuurstudie

De literatuurstudie gaat dieper in op de technische kant van de verschillende *frameworks*. Eerst wordt er uitgelegd wat de drie gekozen *frameworks* juist zijn en daarna wordt er verder gezocht naar de voor-en nadelen van deze *frameworks* om zo een vergelijking op te kunnen stellen. Er gaat ook een vergelijking gemaakt worden tussen drie bronnen die zelf al een vergelijking gemaakt hebben tussen de drie *frameworks*. Aan de hand van dit onderzoek gaat er een antwoord gegeven worden op de volgende deelvragen: "Welk *framework* biedt de meeste functionaliteit?" en "Zijn de *frameworks* bruikbaar binnen grote webapplicaties".

### 2.4.1 Wat is het Angular *framework*?

Om exact te weten te komen wat Angular is moet er eerst een korte uitleg gegeven worden over zijn voorganger AngularJS. AngularJS is een *open source* frontend *framework* uitgebracht in Oktober 2010, dat gebaseerd is op *Javascript*. Dit *framework* is ontwikkeld door Google en wordt onderhouden door een gemeenschap van mensen en bedrijven. AngularJS is ontworpen, om het maken van *single-page* applicaties te vergemakkelijken op gebied van ontwikkelen en testen. [5]

Het schrijven van een AngularJS-applicatie is altijd onordelijk en vaak te omslachtig om iets, vrij simpel te laten functioneren. Om dit probleem en vele andere op te lossen, heeft het team binnen Google het Angular-*framework* ontwikkeld. Angular wordt ook vaak Angular2+ genoemd omdat het de opvolger is van AngularJS of Angular1.

Angular heeft als voordeel, dat het gebruik maakt van *Typescript*-taal [6] van Microsoft. *Typescript* wordt achterliggend omgevormd naar normale *Javascript* dus is even performant, en heeft veel voordelen [7]. Zo kan je bij *Typescript* object-georiënteerd programmeren en gebruikmaken van functies zoals *lambdas, iterators, for-loops* en meer [2]. *Typescript* is ook veel ordelijker om te schrijven, tegenover *Javascript*. Hieronder ziet u een verschil tussen dezelfde code geschreven in *Typescript* en *Javascript*. In deze foto is meteen duidelijk welk voordeel *Typescript* heeft op *Javascript,* op het gebied van leesbaarheid. Wat ook meteen zichtbaar wordt, is dat er veel zaken ongeveer overeenkomen tussen deze twee talen, hierdoor ziet men dat *Typescript* afgeleid is van *Javascript*.

***Code in Typescript:***

```typescript
class MessageSender {
    sender: string;
    constructor(message: string) {
        this.sender = message;
    }
    sendMessage() {
        return "Hello, " + this.sender;
    }
}

let greeter = new MessageSender("world");

let button = document.createElement('button');
button.textContent = "Say Hello";
button.onclick = function() {
    alert(greeter.sendMessage());
}

document.body.appendChild(button);

```

*Figuur 6 - Typescript voorbeeld*

***Code in Javascript:***

```javascript
var MessageSender = /** @class */ (function () {
    function MessageSender(message) {
        this.sender = message;
    }
    MessageSender.prototype.sendMessage = function () {
        return "Hello, " + this.sender;
    };
    return MessageSender;
}());
var greeter = new MessageSender("world");
var button = document.createElement('button');
button.textContent = "Say Hello";
button.onclick = function () {
    alert(greeter.sendMessage());
};
document.body.appendChild(button);

```

*Figuur 28 - Javascript voorbeeld*

Bij Angular wordt er gebruikgemaakt van componenten die overal binnen de applicatie toegankelijk zijn. Hierdoor kan je in één webpagina, verschillende van deze componenten aanroepen en tevens

laten zien op die pagina. Deze componenten werken dan allemaal afzonderlijk met hun eigen achterliggende *Typescript*-code, waardoor ze makkelijk te hergebruiken zijn op andere plaatsen [8].

Angular voorziet ook *loose coupling* aan de hand van *Dependency Injection*. Bij *Dependency Injection* worden afhankelijkheden van andere locaties geïnjecteerd in de applicatie. Hierdoor kunnen ze data uitwisselen zonder dat er een relatie in de code wordt vastgelegd [9].

### 2.4.2 Wat is het React *framework*?

React of beter gekend als React.Js of React JS is een *Javascript-framework* in Mei 2013 ontworpen door Facebook. React is een *open source* en wordt onderhouden door Facebook, en een gemeenschap van mensen en bedrijven die samen op een regelmatige basis nieuwe updates publiceren. Het is de opzet van React, om single-page webapplicaties zo eenvoudige mogelijk te creëren. [10].

React maakt gebruik van HTML en JSX op de pagina's op te bouwen [11]. JSX oftewel *Javascript* XML is een uitbreiding tot de Javascript-syntax. Bij JSX is het mogelijk om HTML-attributen te gebruiken in Javascript-code waardoor er op een eenvoudige manier elementen dynamisch gegenereerd kunnen worden. Hieronder bevindt zich een voorbeeld van een stukje JSX-code waarin meteen zichtbaar is hoe er een mengeling van HTML en *Javascript* samen staan in één klasse.

```
<div id="myReactApp"></div>

<script type="text/babel">
  class Greeter extends React.Component {
    render() {
      return <h1>{this.props.greeting}</h1>
    }
  }

  ReactDOM.render(<Greeter greeting="Hello World!" />, document.getElementById('myReactApp'));
</script>
```

*Figuur 29 - JSX-code*

Een andere nuttige functie van JSX, het wordt nu mogelijk, om expressies en conditionele uitdrukkingen te gebruiken in de HTML-code. Het is ook haalbaar om functies te gebruiken binnen HTML-code. Het gebruik van JSX is niet verplicht binnen React. Het is ook mogelijk om met normale Javascript de applicatie op te bouwen [12].

Een ander groot voordeel van React tegenover een klassieke HTML en Javascriptwebapplicatie is dat React gebruikmaakt van een "Virtual DOM" oftewel "Virtual Document Object Model". Dit wil zeggen dat React continu alle veranderingen bijhoudt in een *datacache* en zo in de browser alleen de elementen verandert die aangepast zijn. Op deze manier kan een ontwikkelaar zijn code schrijven alsof de hele pagina opnieuw geladen wordt, terwijl eigenlijk alleen maar de delen die veranderen geüpdatet worden. Een groot voordeel hiervan is dat de pagina's heel dynamisch gemaakt kunnen worden zonder snelheid te verliezen bij het veranderen van elementen [13].

### 2.4.3 Wat is het Vue *framework*?

Vue.js of gewoon Vue is een open source *Javascript-framework,* ontworpen om *single-page* webapplicaties te ontwikkelen. Even You heeft Vue gecreëerd na dat, hij bij Google in hun *Creative Labs* afdeling heeft gewerkt. Hier diende hij snel prototypes op te leveren, dus zocht hij naar een geschikt *framework*. Rond deze tijd was het enige bruikbare *framework* Angular, wat niet performant genoeg was voor de hoeveelheid prototypes die hij moest maken. Hierdoor begint hij zelf aan zijn eigen *framework*, namelijk "Vue.js", wat de nodige delen van Angular.JS gebruikt, maar dan veel lichter en performanter [14].

Vue maakt net zoals React gebruik van een *Virtual DOM*. Hierdoor kunnen elementen op een webpagina veranderen zonder dat de volledige webpagina opnieuw moet inladen. Dit zorgt ervoor dat Vue zo performant mogelijk is [15].

Wat vue nog heeft overgenomen uit React, is het gebruik van JSX, i.p.v. normale *Javascript*. Het voordeel hiervan is, dat het mogelijk wordt om HTML-code te mengen met *Javascript*-code. Zo kunnen er functies uitgevoerd worden in de HTML-code [16]. Hieronder ziet u een voorbeeld van JSX gebruikt binnen een Vue.js applicatie.

```
<script type="text/x-template" id="anchored-heading-template">
  <h1 v-if="level === 1">
    <slot></slot>
  </h1>
  <h2 v-else-if="level === 2">
    <slot></slot>
  </h2>
  <h3 v-else-if="level === 3">
    <slot></slot>
  </h3>
  <h4 v-else-if="level === 4">
    <slot></slot>
  </h4>
  <h5 v-else-if="level === 5">
    <slot></slot>
  </h5>
  <h6 v-else-if="level === 6">
    <slot></slot>
  </h6>
</script>
```

*Figuur 30 - Vue.js JSX*

Om de beloften van een *framework* te zijn voor *prototyping* waar te maken, heeft Vue recent een CLI uitgebracht. Dankzij deze CLI krijgt men in Windows terminal toegang tot het vue commando, en kan je projecten aanmaken, door middel van een simpel commando uit te voeren [17]. Hierdoor verliezen ontwikkelaars veel minder tijd aan projecten, en kunnen ze meteen beginnen met het ontwikkelen van prototypes.

### 2.4.4 Gelijkenissen en verschillen tussen de *frameworks*

React wordt door de meeste mensen als een *framework* gezien maar dit is niet correct. React is een view georiënteerde *library,* maar niet ontworpen om modellen en controllers in te maken. Het is wel mogelijk om React als een *framework* te gebruiken, maar dit is geen goede keuze. Angular en vue daarentegen zijn wel volledige MVC *frameworks*. Dit wil zeggen dat het mogelijk is om modellen, *views* en *controllers* te maken die met elkaar verbonden zijn [18].

De meest belangrijke gelijkenis tussen Vue, React en Angular is dat ze allemaal ontworpen zijn voor het maken van responsieve *Single Page Applications* [19] [20] [1] . Dit wil zeggen dat er maar één enkele HTML-pagina wordt ingeladen bij het binnenkomen van de website. In deze pagina worden dan dynamisch alleen de elementen ingeladen die op dat moment getoond moeten worden. Hierdoor moet de website niet helemaal opnieuw inladen, wanneer je naar een ander deel van de website navigeert [21]. Dit maakt de vergelijking tussen deze drie *frameworks* eerlijk, aangezien ze alle drie bedoeld zijn voor dezelfde soort applicaties te bouwen.

React en Vue maken allebei gebruik van een virtuele DOM, die bijhoudt welke aanpassingen er allemaal gebeuren op een webpagina. Aan de hand hiervan, kan deze er dan voor zorgen dat alleen deze elementen opnieuw geladen worden [15]. Een voorbeeld hiervan is bijvoorbeeld wanneer er een lijst is met 5 elementen. Als bij een klassiek DOM systeem één van die elementen zou veranderen moet die volledige lijst opnieuw geladen worden. Bij een virtuele DOM wordt alleen dat 1$^{e}$ element opnieuw geladen.

Angular maakt geen gebruik van een virtuele DOM, maar heeft zijn eigen manier ontwikkelt om websites performant te houden. Bij Angular wordt er gebruik gemaakt van, zowel de *browser thread,* als de *service worker thread*. Hierdoor kan de *browser thread* alleen gebruikt worden voor de DOM te laden, en de *service worker thread* daarentegen voor de rest. Door deze manier van renderen toe te passen, gaat het laden van de DOM heel snel bij Angular [3].

### 2.4.5 Wat is een DOM?

Er is ondertussen al een aantal keren gesproken over welke DOM de verschillende *frameworks* gebruiken, maar wat is een DOM juist? Veel mensen denken dat de DOM de HTML-code is en dit is niet het geval.

De DOM ofwel “Document Object Model” is omgevormde HTML of XML-code die je browser kan lezen en weergeven. Het is mogelijk dat er geen verschil is tussen de HTML die je schrijft en de HTML van de DOM. Het verschil zie je pas echt wanneer er een aantal zaken zijn in je HTML die missen. De DOM gaat deze zaken dan automatisch toevoegen zodat de browser dit kan weergeven [22]. Hieronder zie je een voorbeeld tussen geschreven HTML-code en de HTML die de DOM ervan gemaakt heeft.

```
<html>
<body>
<h1>HTML vs DOM</h1>
<table>
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>John</td>
    <td>Doe</td>
    <td>50</td>
  </tr>
</table>
</body>
</html>
```

*Figuur 32 - Geschreven HTML*

```
<html>
  <head></head>
▼<body>
    <h1>HTML vs DOM</h1>
  ▼<table>
    ▼<tbody>
      ▼<tr>
          <th>Firstname</th>
          <th>Lastname</th>
          <th>Age</th>
        </tr>
      ▼<tr>
          <td>John</td>
          <td>Doe</td>
          <td>50</td>
        </tr>
      </tbody>
    </table>
  </body>
</html>
```

*Figuur 31 - DOM HTML*

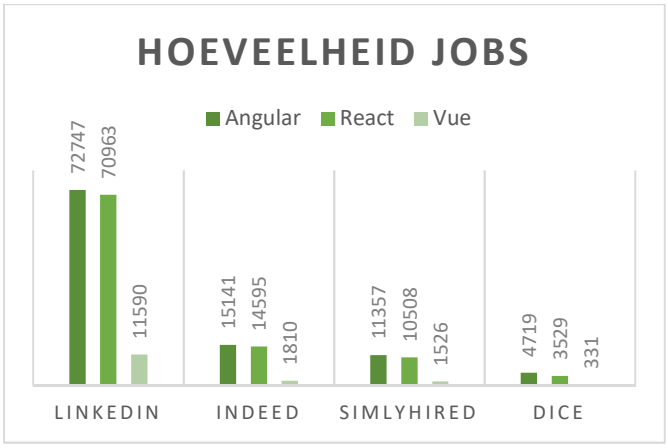
Zoals hierboven te zien is zijn er een aantal zaken bijgekomen in de DOM die nodig waren om de pagina juist in te kunnen laden. In dit voorbeeld is het alleen maar “<tbody>” en de “<head>” tag. Bij grotere stukken code gaat dit veel meer zijn. De DOM is ook verantwoordelijk om dynamisch elementen toe te voegen aan de webpagina. Dit kan bijvoorbeeld gebeuren wanneer er in een script extra elementen gegenereerd worden. Deze elementen worden dat niet in de html gegenereerd maar rechtstreeks in de DOM [22].

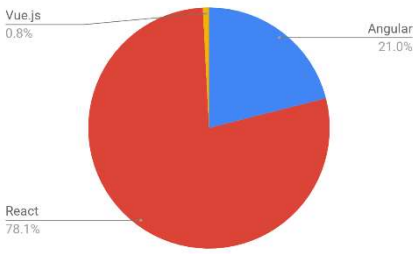
### 2.4.6 Welk *framework* wordt het meeste gebruikt?

Andrei Neagoie van medium.com heeft onderzoek gedaan naar de hoeveelheid vraag die er is naar elk van deze *frameworks* in de bedrijfswereld [23]. Om dit te onderzoeken is hij op verschillende plaatsen gaan kijken hoeveel vraag er is naar ontwikkelaars voor de *frameworks*. Uit zijn onderzoek is gebleken (zie grafiek) dat er op dit moment de meeste vraag is naar Angular ontwikkelaars. Het probleem met deze cijfers is dat hier niet alleen Angular bekeken is maar ook AngularJS.

In een ander artikel op medium.com had men een gelijkaardige test gedaan maar dan met zonder AngularJS mee te rekenen [24]. Hier zijn ze ook maar op één enkele plaats gaan kijken om de cijfers te halen. Hier is duidelijk zichtbaar dat React het meest populaire is in de job markt. In Vue daarentegen is het veel moeilijker om een werkplek te vinden.

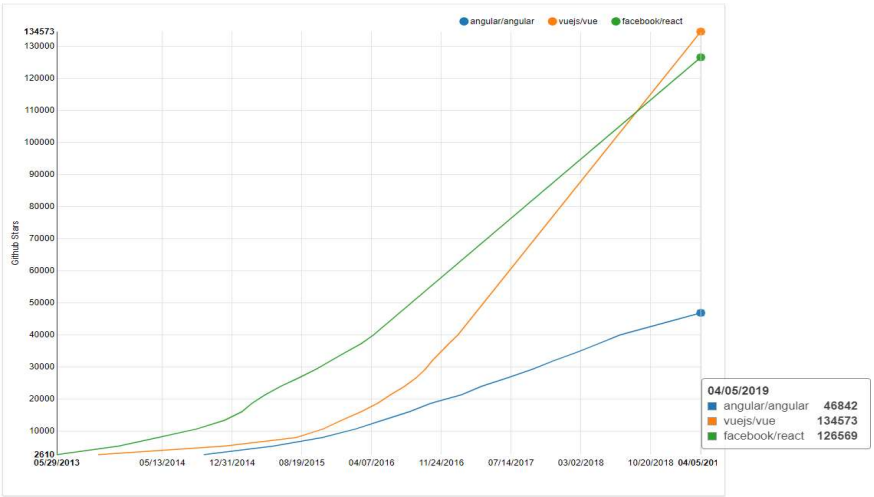
*Tabel 1 - Hoeveelheid Jobs per framework*



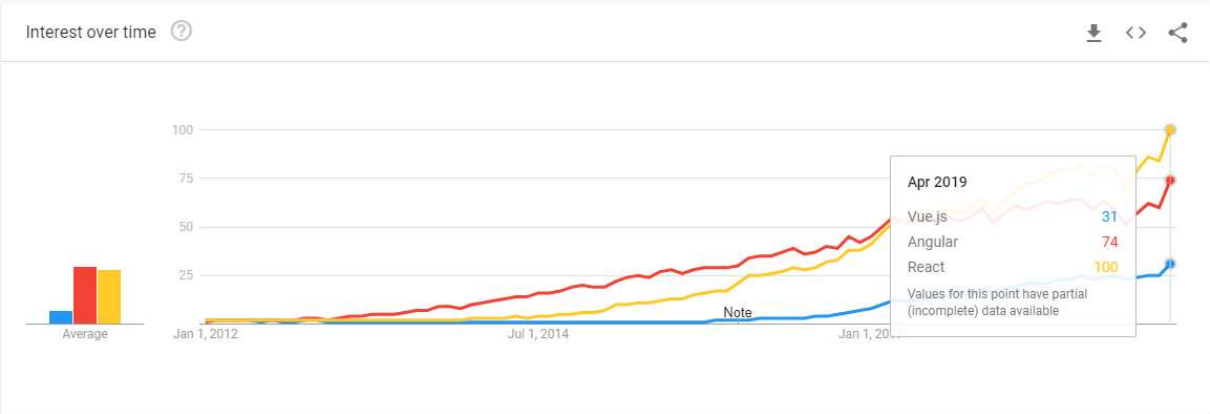

*Figuur 33 - Jobmarkt*

Wanneer er gekeken wordt naar de populariteit van deze *frameworks* buiten het bedrijfsleven krijg je een heel ander beeld. Hieronder zie je een grafiek van de hoeveelheid sterren die elk van deze *frameworks* heeft op GitHub [25]. Op deze grafiek is zichtbaar dat Vue recent populairder is geworden op GitHub als React en Angular.



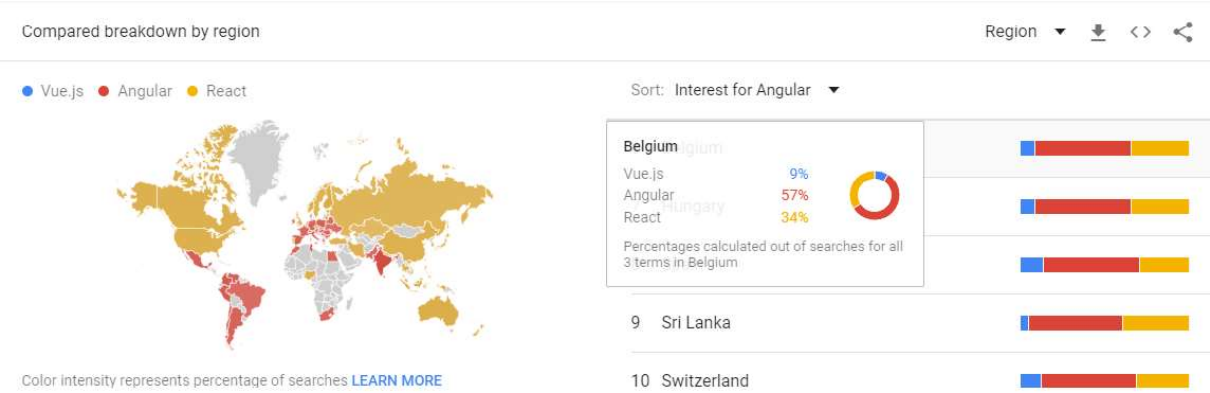

*Figuur 34 - Git Sterren*

0p Google Trends is dan weer zichtbaar dat er vooral gezocht wordt naar React en Angular. Dit zal te maken hebben met de hoeveelheid ontwikkelaars die werkt aan projecten met deze *frameworks*. Omdat Angular en React langer bestaan gaan er meer ontwikkelaars hier zaken over moeten opzoeken.




*Figuur 35 - Google Trends*

Wanneer er specifiek naar België gekeken wordt, ziet men dat hier vooral gezocht wordt naar Angular en React. Vue wordt 8% zoveel opgezocht als Angular wat weinig is.




*Figuur 36 - Google Trends België*

Een besluit dat hieruit getrokken kan worden, is dat momenteel Vue nog achterloopt in het bedrijfsleven. Waartegen Angular en React minder bekend beginnen te worden buiten werk uit. Samengevat, momenteel is React het meest gebruikte *framework*.

### 2.4.7 Wat is het beste *framework*?

Om deze vraag te beantwoorden, wordt er een vergelijking gemaakt tussen drie bronnen, die allemaal hun eigen vergelijking gemaakt hebben tussen deze drie *frameworks*. De drie bronnen die gebruikt worden in deze vergelijking komen van: “medium.com” [26], “codeinwp.com” [27] en “spec-india.com” [28].

#### 2.4.7.1 Scripting taal

Om te beginnen worden alle zaken die de websites met elkaar overeenkomen naast elkaar gezet. Een zaak die ze alle drie aanhalen is dat het *native* gebruik van *Typescript* binnen Angular een groot voordeel heeft tegenover de twee andere *frameworks* die gebruik maken van *Javascript* of JSX. Wat er wel gezegd wordt is dat JSX het vaak wel makkelijker maakt om de HTML dynamisch aan te maken, iets wat bij Angular alleen maar gaat via de meegeleverde functies.

#### 2.4.7.2 Populariteit

De drie websites halen ook aan hoe dat Vue het afgelopen jaar React en Angular is voorbijgestoken op gebied van hoeveelheid *Git Stars* en andere *Git* statistieken. Alle drie zeggen dat Angular, de moeilijkste is om aan te leren voor een nieuwe ontwikkelaar. Maar van het ogenblik dat je met Angular kan werken, wordt het leren van de andere twee makkelijker.

#### 2.4.7.3 Schaalbaarheid

Bij codeinwp.com wordt ook aangehaald dat Vue nog jong is, en hierdoor niet altijd geschikt is om grote applicaties in te bouwen. De hoeveelheid support in zo een nieuw *framework* kan soms voor problemen zorgen, doordat deze al langer bestaan, is dit bij Angular en React niet het geval. Wat hier nog bij gezegd moet worden, is dat Vue het laatste jaar, het heel goed doet. En hierdoor een sterke tegenstander is voor Angular en React.

#### 2.4.7.4 Conclusie

In de conclusie van codeinwp.com en spec-india.com is er geen duidelijke winnaar gevonden, omdat ze zeggen dat deze drie *frameworks* allemaal hun voor-en nadelen hebben ten opzichte van elkaar. Er wordt gezegd dat er bij elke nieuwe opdracht gekeken moet worden welk van de drie *frameworks* met meest geschikt is. Als je bijvoorbeeld houdt van Object georiënteerd programmeren is het best om te kiezen voor Angular. Als je flexibiliteit verkiest, is het beter om React te gebruiken. Wanneer je gebruik wilt maken van een licht *framework* dat heel simpel is om aan te leren, kan je beter Vue gebruiken.

Bij medium.com zegt men bij de conclusie, als je aan een nieuw project begint, en je geen idee hebt welk van de drie *frameworks* je moet gebruiken, je beter kan kiezen voor React. Omdat React het meest gebruikte *framework* is van de drie. Op deze manier vind je meer informatie over problemen die je onderweg kan tegenkomen, en *tutorials* die je op weg helpen. Als tweede *framework* kiest men voor Vue omdat het snel is aangeleerd en de nodige functionaliteit bevat, om de meeste webapplicaties te bouwen. Angular tenslotte wordt als laatste geplaatst wegens de moeilijke leercurve, en het gebruik *Typescript* tegenover JSX.

### 2.4.8 Persoonlijke reflectie literatuurstudie

Voordat ik aan de literatuurstudie begon had ik geen idee wat ik ervan moest verwachten. Ik had immers zelf nog nooit uitgebreid moeten zoeken naar informatie rond een bepaald onderwerp. Het

was ook de eerste keer dat ik verschillende bronnen moest raadplegen en hier de vergelijkingen en verschillen uit moest zoeken.

Ik heb hierdoor bijgeleerd dat opzoeken en vergelijken niet altijd saai moet zijn. Wanneer een onderwerp mij interesseert gaat vergelijken vanzelf veel vlotter. Ik heb veel kunnen bijleren over de drie *frameworks* waardoor mijn kennis rond *web frameworks* enorm is gegroeid.

Als ik naar de toekomst toe meer onderzoek moet gaan doen kan ik zeker van de kennis die ik hier heb opgedaan gebruik maken. Zo kan ik later altijd correcte informatie bezorgen aan de hand van verschillende bronnen.

## 2.5 Experiment

In dit onderzoek gaat er een vergelijking gemaakt worden tussen de drie gekozen *frameworks*. Om deze vergelijking te maken gaat er bekeken worden hoe goed de *frameworks* werken onder een aantal testcases. Aan de hand van deze verschillende testcases gaat er een antwoord gegeven worden op de volgende deelvragen: “Welk *framework* is het makkelijkst aan te leren?” en “Welk *framework* is het meest performant?”.

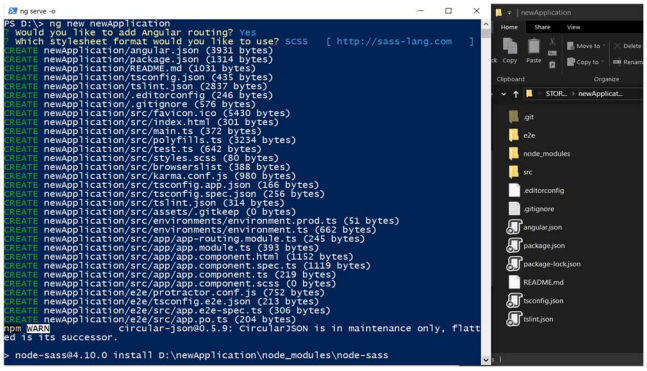
### 2.5.1 Aanmaken & beheren van een project

Bij het aanmaken van een nieuw project is er een verschil te merken bij deze verschillende *frameworks* op gebied van eenvoudigheid. Bij alle drie de *frameworks* kan er gebruik gemaakt worden van een CLI-tool om eenvoudig een nieuw project aan te maken. Dit wil zeggen dat er door het installeren van deze tools, de mogelijkheid bestaat een *command-line* venster te openen en hierin de verschillende commando’s uit te voeren.

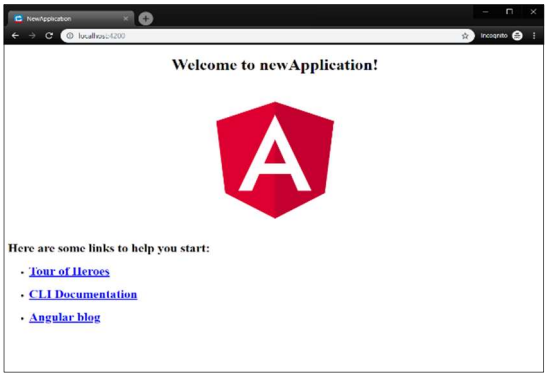
Het belang van deze vraag zit zich bij leren van een nieuw *framework*. Wanneer een nieuwe ontwikkelaar begint met het leren van een nieuw *framework*, zal deze zo weinig mogelijk zaken manueel willen instellen. Hoe makkelijker het aanmaken en beheren van een project gaat hoe sneller een ontwikkelaar kan beginnen aan het schrijven van code.

### 2.5.2 Angular

Bij Angular wordt een nieuw project aangemaakt aan de hand van het commando “ng new <naam>”. Hier kan er eerst gekozen worden of er gebruik moet gemaakt worden van Angular routing om te navigeren tussen verschillende componenten. Hierna kan men kiezen welke soort van styling er gebruikt moet worden. Aan de hand van deze info gaat de CLI de nodige bestanden downloaden en samen in een folder plaatsen. Om het nieuwe project op te starten moet er genavigeerd worden naar deze folder en hierin het commando “ng serve” uitvoeren. Het resultaat is in onderstaande afbeeldingen zichtbaar.




*Figuur 38 - Nieuw project via Angular CLI*

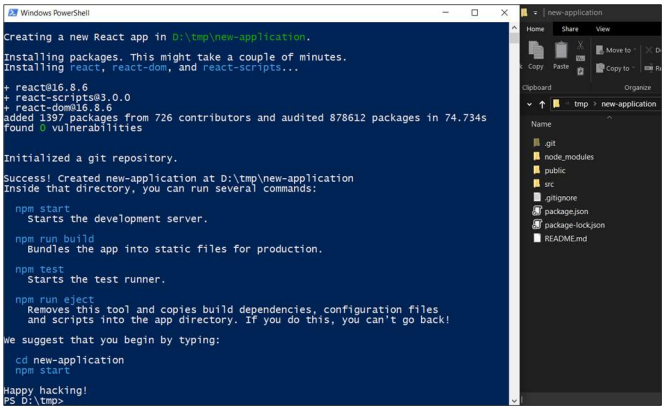



*Figuur 37 - Nieuw Angular project resultaat*

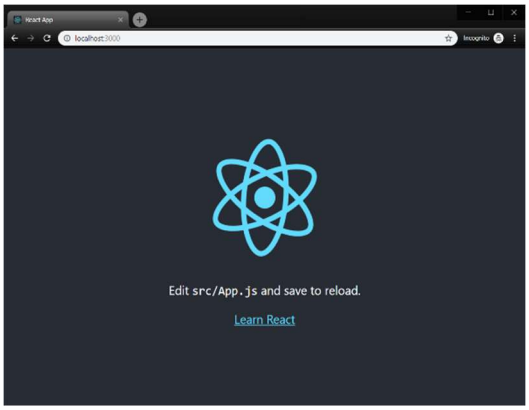
Bij Angular biedt deze CLI veel meer mogelijkheden dan alleen het aanmaken van een nieuw project. Zo kan ook aan de hand van het commando “ng generate component <naam>” een nieuw component aangemaakt worden waardoor er geen tijd verloren wordt met het aanmaken van zo een component. Deze commando’s bestaan er ook voor het aanmaken van services, pipes, classes en meer. Zonder deze commando’s moeten al deze zaken manueel toegevoegd worden bij het aanmaken van een nieuw project en tijdens het programmeren.

### 2.5.2.1 React

Bij React gebeurt het aanmaken van een nieuw project ongeveer hetzelfde als bij Angular. Met een package manager zoals bijvoorbeeld node.js wordt de CLI-tool geïnstalleerd. Waarna het mogelijk wordt om in een *command-line* venster, de React commando's uit te voeren. Door middel van het commando "create-react-app <naam>" worden de nodige bestanden gedownload en samen in een folder geplaatst. In deze folder kan hierna het commando "npm start" uitgevoerd worden, om de applicatie op te starten.



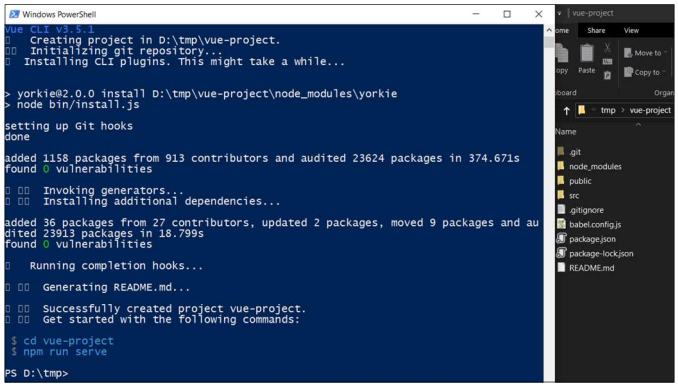

*Figuur 40 - Nieuw project via de React CLI*



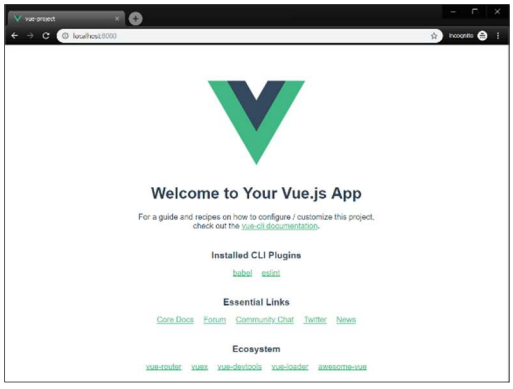

*Figuur 39 - Nieuw React project resultaat*

### 2.5.2.2 Vue

Bij Vue kan het aanmaken van een project gebeuren op dezelfde manier als bij Angular en React. De CLI-tool moet eerst geïnstalleerd worden in een package manager naar keuze en hierna kunnen de verschillende commando's gebruikt worden. Om een nieuw project aan te maken moet het commando "vue create <naam>" uitgevoerd worden waarna de nodige bestanden worden gedownload en geplaatst in een folder. In deze folder kan dan het commando "npm run serve" uitgevoerd worden om het project op te starten.



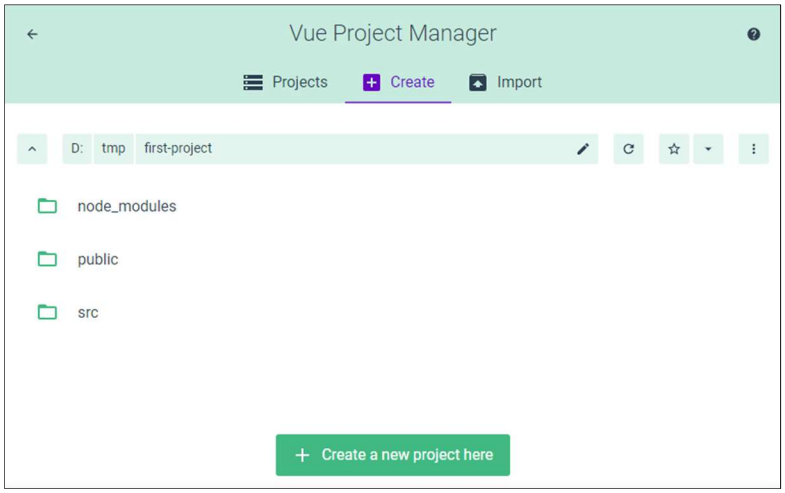

*Figuur 41 - Nieuw project via de Vue CLI*




*Figuur 20 - Nieuw Vue project resultaat*

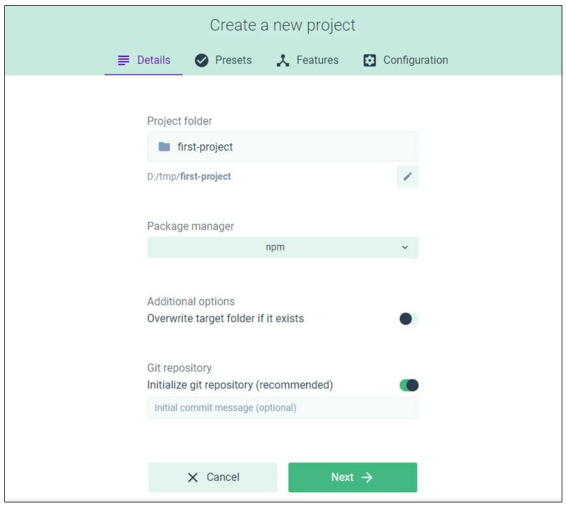
De CLI is een makkelijk te gebruiken tool, maar dit is niet het beste wat Vue te bieden heeft. Bij het installeren van de CLI, installeert Vue ook een grafische interface waarop nieuwe projecten aangemaakt en gemonitord kunnen worden. Om deze tool op te starten moet er een command-line venster geopend worden, om het commando "vue ui" uit te voeren.

Als de interface voor de eerste maal geopend wordt, zal er een scherm tevoorschijn komen met de vermelding, dat er nog geen projecten zijn aangemaakt. Om een project aan te maken, wordt er eerst op de tab “Create” geklikt en daarna op de knop “Create a new project here”. De locatie van waar het project aangemaakt wordt, kan gekozen worden in de bovenliggende balk.

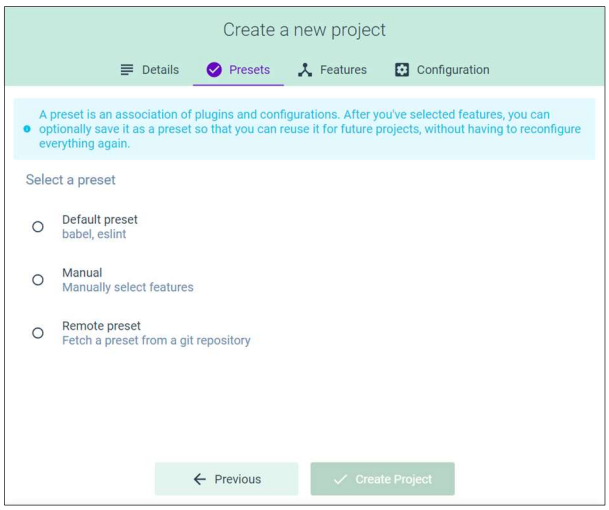



*Figuur 42 - Vue UI aanmaken nieuw project*

Bij het klikken op de knop wordt er naar de volgende pagina genavigeerd. Op deze pagina kan je de naam van het project instellen. Er is ook een mogelijkheid om een andere *package* manager te gebruiken om *packages* te downloaden, deze wordt hier aangeduid. In de volgende tab staan verschillende *presets* waaruit je kan kiezen. Klikt men op “Create Project”, worden vervolgens de nodige bestanden gedownload, en wordt er een folder gemaakt die nodige bestanden bevat.

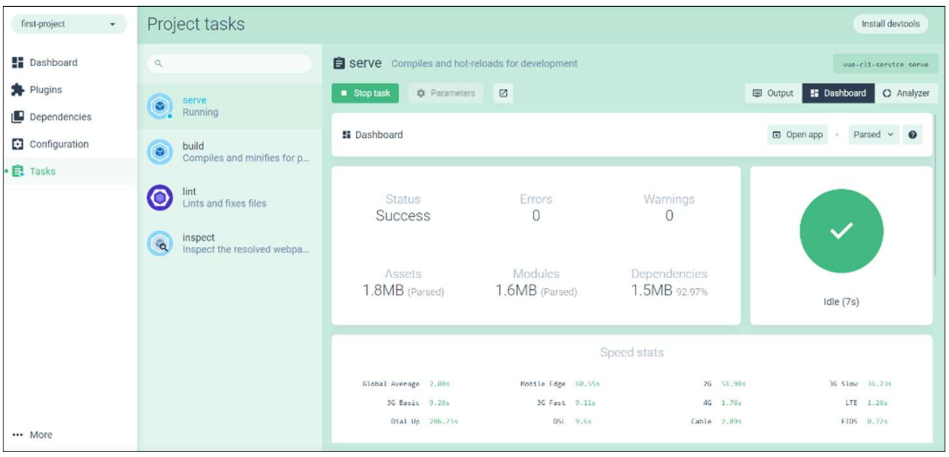



*Figuur 44 - Vue UI aanmaken nieuw project 2*




*Figuur 43 - Vue UI aanmaken nieuw project 3*

Wanneer het project aangemaakt is, wordt er een dashboard getoond. In dit dashboard zijn er verschillende zaken die je kan doen met het nieuwe project. De meest belangrijke hiervan is het project opstarten. Dit gebeurt door te klikken op “Tasks” en daarna op de tab “Serve”, hierna klik je op de knop “Run task”. De applicatie wordt hierdoor opgestart, door op de knop “Open app” te drukken, wordt deze geopend in een nieuw venster. In de “serve” tab kan de applicatie ook gemonitord worden om na te gaan of er geen foutmeldingen en nog andere handige statistieken weergegeven worden.



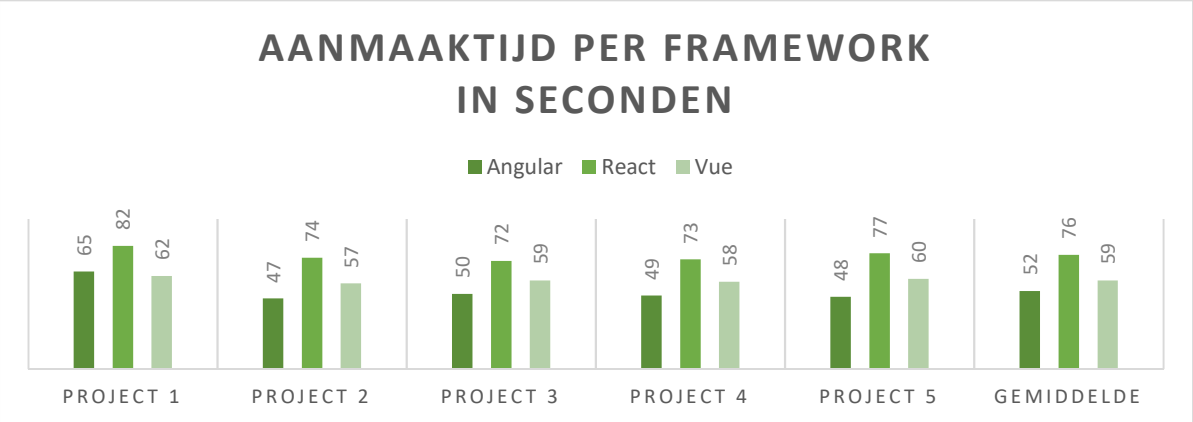

*Figuur 45 - Vue dashboard*

### 2.5.2.3 Conclusie aanmaken en beheren van project

Dankzij de grafische interface waarop alles ingesteld en gemonitord kan worden, heeft Vue bij het aanmaken van een nieuw project, een voorsprong op Angular en React. Het gebruik van een CLI-tool kan, indien men hier nog geen ervaring mee heeft, voor nieuwe ontwikkelaars intimiderend zijn. Voor deze ontwikkelaars is het gebruik van een UI veel makkelijker te begrijpen. Hierdoor kunnen ze sneller beginnen voor het ontwikkelen van webapplicaties.

Wanneer er alleen naar het *command-line* gedeelte gekeken wordt, is het winnende *framework* kiezen niet meer zo eenvoudig. De *frameworks* bieden nu alle drie functionaliteit, om een project aan te maken. En dit zonder alles manueel te moeten samenstellen in een folder. Hiervoor moet er rekening gehouden worden met de tijd, die nodig is om een project aan te maken.

*Tabel 2 - Aanmaaktijd per framework*

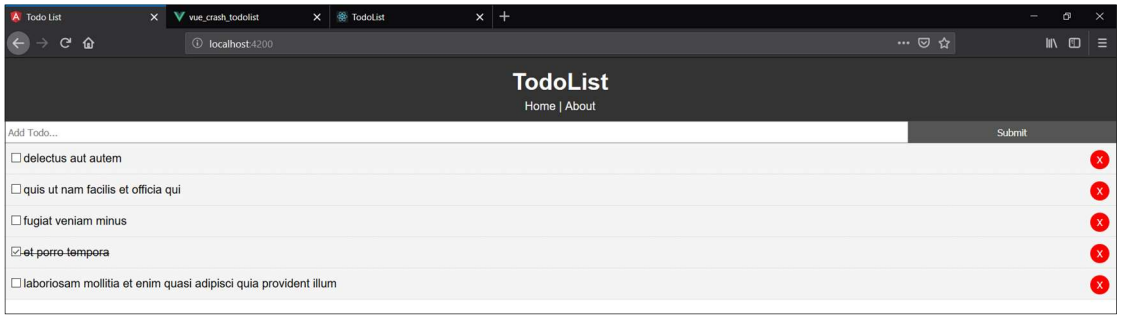



Uit het testen van het creëren van 5 projecten per *framework* blijkt dat Angular het snelste is om een nieuw project op te zetten via de *command-line* en dus het winnende *framework* is in deze categorie.

### 2.5.3 Leercurve

Een belangrijk punt bij het uitkiezen van een *framework* om te gebruiken is de leercurve. Een *framework* dat weken nodig heeft om aan te leren is het vaak niet waard om aan te beginnen. In het geval van dit onderzoek wordt er gekeken hoe moeilijk elk van de *frameworks* is en de beschikbaarheid van de nodige documentatie.

Omdat ik al met Angular heb leren werken is het moeilijk om precies te zien hoe lang het duurt om de verschillende *frameworks* aan te leren. Hierdoor ben ik gaan zoeken naar lessen van de drie *frameworks* die hetzelfde resultaat opbouwen. Zo ben ik uit gekomen op drie Crash Courses van Traversy Media [29]. In deze courses wordt drie keer een simpele webpagina gemaakt voor het bijhouden van todo items.




*Figuur 46 - Crash course resultaat*

Deze *course* haalt verschillende belangrijke zaken aan, namelijk:

- Werken met verschillende componenten.
- Data sturen tussen componenten.
- Functies aanroepen tussen componenten.
- Routing.
- Verbinding met een publieke API.

Dit zijn allemaal belangrijke zaken om na te lezen. Met deze kennis is het al mogelijk, om simpele basis applicaties te maken. Deze basiskennis maakt het ook makkelijker om meer andere zaken bij te leren.

#### 2.5.3.1 Angular

Wanneer je Angular voor de eerste keer gebruikt valt meteen het gebruik van *Typescript* op. Als je er nog geen kennis van hebt kan het moeilijk zijn om hiermee te leren werken. *Typescript* maakt het mogelijk om object georiënteerd te werken in combinatie met gewone Javascript functionaliteit.

Bij Angular gebruikt men het MVC-patroon. Dit wil zeggen dat de modellen, views en controllers van elkaar gesplitst zijn. Dit heeft als voordeel, dat er de codes niet door elkaar staan en dat de code makkelijker te hergebruiken is. Ook zorgt dit ervoor dat elke klasse maar één enkele responsabiliteit heeft, wat zorgt dat je kan werken met het SOLID-principe.

Het gebruik van services kan ook een nadeel zijn voor nieuwe ontwikkelaars. Bij Angular worden deze gebruikt voor het verwerken van data i.p.v. dit in de componenten zelf te doen. Dit neemt extra tijd in beslag wanneer je snel aanpassingen wil maken of iets aan de code wil toevoegen.

Bij Angular zijn er meteen heel veel zaken ingebouwd zoals een httpclient om data op te vragen uit een API, *forms* om data in te geven en meer. Hierdoor moeten er niet voor alles een dependentie gedownload worden en kan je sneller beginnen aan het schrijven van code.

Door de voorganger AngularJS kan het soms moeilijk zijn om documentatie te vinden van Angular. De meeste zoekresultaten geven antwoord op problemen met AngularJS wat het moeilijk maakt om een correct antwoord te vinden.

#### 2.5.3.2 React

Bij React wordt net zoals bij Angular gebruik gemaakt van componenten die dynamisch aangeroepen kunnen worden en hun eigen code bevatten. Dankzij met deze componenten te werken krijg je minder dubbele code en verlies je minder tijd met het schrijven van code. Voor mensen die van andere *frameworks* komen, kan deze moeilijk zijn om aan te leren. Er moet altijd nagedacht worden of er zaken zijn die je kan opsplitsen in aparte componenten.

Het gebruik van JSX zorgt ervoor dat er geen tijd verloren wordt met het opsplitsen van de html en de achterliggende code. Hierdoor is het maken van een nieuwe pagina of element heel snel en versnelt het ontwikkelproces.

JSX zorgt er ook voor dat mensen met weinig Javascript kennis sneller op weg geraken. Er moet minder tijd besteed worden met het dynamisch genereren van elementen omdat de code rechtsreeks in de HTML geplaatst kan worden. Met traditionele HTML moet je elementen toevoegen aan de *inner-html* van elementen door middel van namen of *ids*. Bij JSX zet je de code in het element waar de nieuwe elementen gegenereerd moeten worden.

Wanneer er gebruik wordt van speciale functies zoals het ophalen van data uit een API moet er bij React meteen een externe *dependency* opgehaald worden. Dit maakt het soms moeilijk om te weten welke er goed zijn om te gebruiken en welke niet.

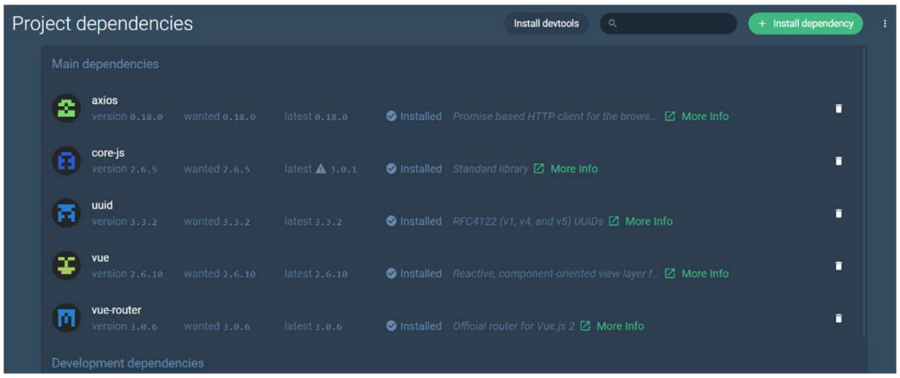
React biedt een uitgebreide documentatie voor het aanleren van dit *framework*. Ze hebben hun eigen *getting started guide* waarin de basis functionaliteit wordt aangeleerd. Ook is er veel informatie te vinden op het internet over eventuele fouten die je kan tegenkomen. Dit komt omdat het al een ouder *framework* is met een grote gemeenschap van mensen die het dagelijks gebruiken.

#### 2.5.3.3 Vue

Bij Vue wordt er net zoals bij Angular en React gebruik gemaakt van componenten, wat het makkelijk maakt als men al kennis heeft van Angular of React. Voor nieuwe ontwikkelaars kan dit wel verwarrend zijn.

Bij Vue kan men gebruik maken van, zowel Javascript als *Typescript*. *Typescript* is nog nieuw bij dit *framework,* en wordt ook nog niet aangeraden om te gebruiken voor grote projecten. De mogelijkheid om beide Scripting talen te gebruiken, door al kennis te hebben van één taal, maakt van Vue een makkelijk aan te leren *framework*.

Bij Vue moet net zoals bij React, voor de meeste zaken een externe dependentie gedownload worden, om veel functionaliteit te laten werken. Het voordeel wat Vue hier heeft, is de meegeleverde UI, waarop er overzichtelijk dependenties beheerd kunnen worden.




*Figuur 47 - Vue dependenties*

Vue maakt het ook heel makkelijk om data en functies door te sturen naar onder en bovenliggende componenten. Door gebruik te maken van "$emit" kan data doorgestuurd worden naar onderliggende componenten. Om de data naar onder sturen bij Angular, moet men hiervoor *EventEmitters* gebruiken. Om dit bij React te doen moet er data gelinkt worden aan methodes, wat verwarrend kan zijn als er meerdere functies aangeroepen moeten worden.

#### 2.5.3.4 Conclusie leercurve

Uit de drie *frameworks* is Angular, voor nieuwe ontwikkelaars duidelijk de moeilijkste om aan te leren. Om Angular te kunnen gebruiken, moet men eerst leren werken met *Typescript*, iets wat voor mensen die van Javascript komen niet evident is.

Als er tijd ingestoken wordt om Angular goed te leren, heeft men wel meer kennis, over hoe de frontend in elkaar zit. Het gebruik van MVC, kan het aanleren vereenvoudigen, voor ontwikkelaars die uit de backend wereld komen.

React zit tussen Angular en Vue op gebied van moeilijkheid. React biedt veel documentatie die het aanleren vereenvoudigd. Er is ook veel meer informatie te vinden, over wat nuttig is voor nieuwe ontwikkelaars. Sommige zaken in React zijn zeer omslachtig, zoals het aanroepen van functies in onderliggende componenten.

Vue is het makkelijkste om aan te leren uit de drie *frameworks*. Dankzij het gebruik van JSX kan er binnen Vue zeer snel begonnen worden met code schrijven. Alles zit logisch in elkaar en de manier van data beheren is makkelijk te begrijpen.

Dankzij de meegeleverde UI bij Vue, kan iedereen eenvoudig een project aanmaken, en de dependenties beheren. Zelfs voor mensen die niet kunnen werken met een *command line interface*.

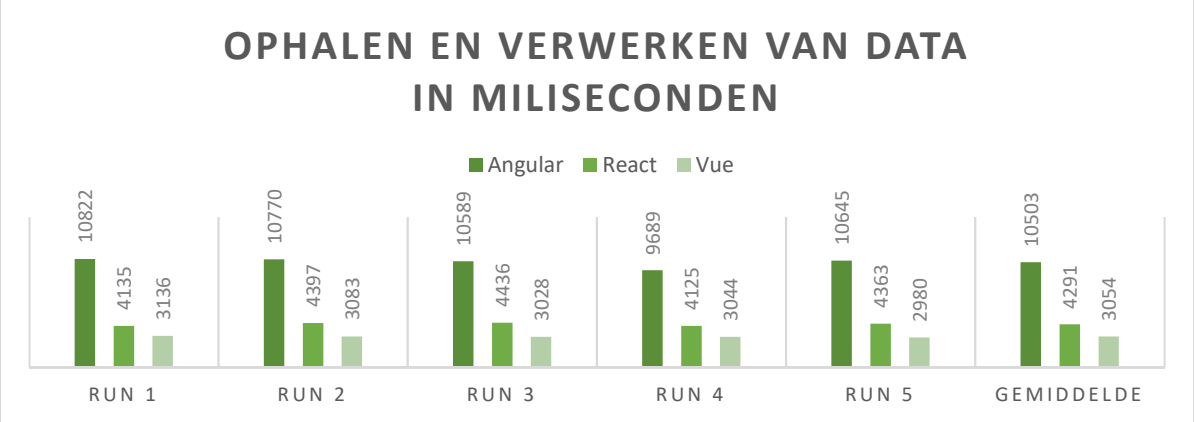
### 2.5.4 Snelheid

#### 2.5.4.1 REST calls & verwerken van data:

Bij bijna alle webapplicaties worden er REST calls gebruikt om data op te halen. Deze data worden dan verwerkt en vaak getoond in de webpagina. Omdat dit een essentieel deel is van een *framework* is er hier een snelheidstest rond gedaan. In deze test worden er 100 REST calls gedaan voor het ophalen van telkens 100 objecten. Deze objecten worden dan in een lijst geladen op de webapplicatie.

Deze test is vijf keer uitgevoerd per *framework*. De tijd werd gemeten vanaf het inladen van de pagina totdat het laatste object in de pagina geladen was.

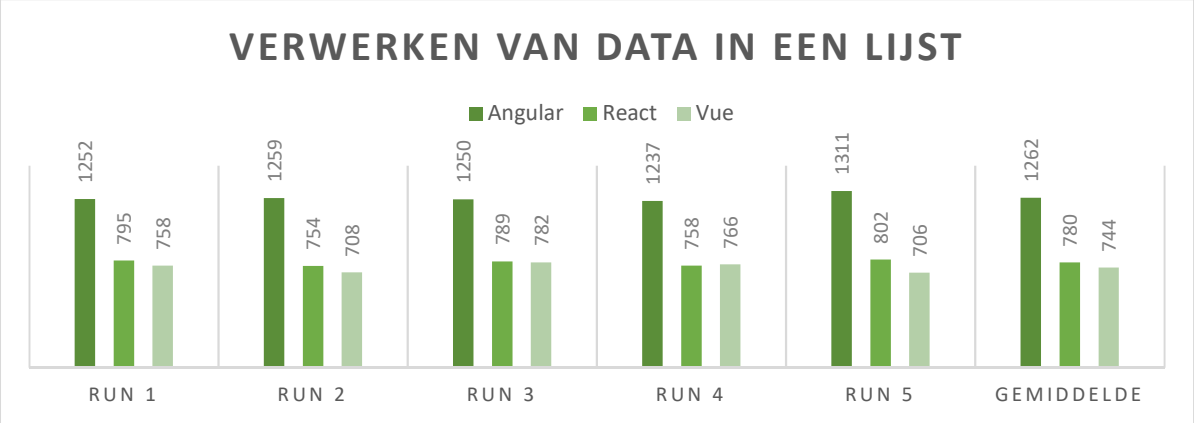
*Tabel 3 - Snelheid van de frameworks*




Uit de resultaten is meteen te zien hoe React en Vue een grote voorsprong hebben op Angular op gebied van snelheid. Het verschil van snelheid lag vooral aan de tijd die nodig was om de objecten in te laden. Bij alle drie de *frameworks* lag de tijd per *REST call* rond de 20ms. Waardoor de resterende tijd nodig was voor het inladen van de objecten in de lijst.

Om dit resultaat verder te bekrachtigen is er ook een test gedaan naar het inladen van data zonder het gebruiken van *REST calls.* In deze test werden er 1000 keer 10 objecten ingeladen in een lijst. De tijd die hiervoor nodig was wordt hieronder getoond in de grafiek.

*Tabel 4 - Snelheid van de frameworks 2*

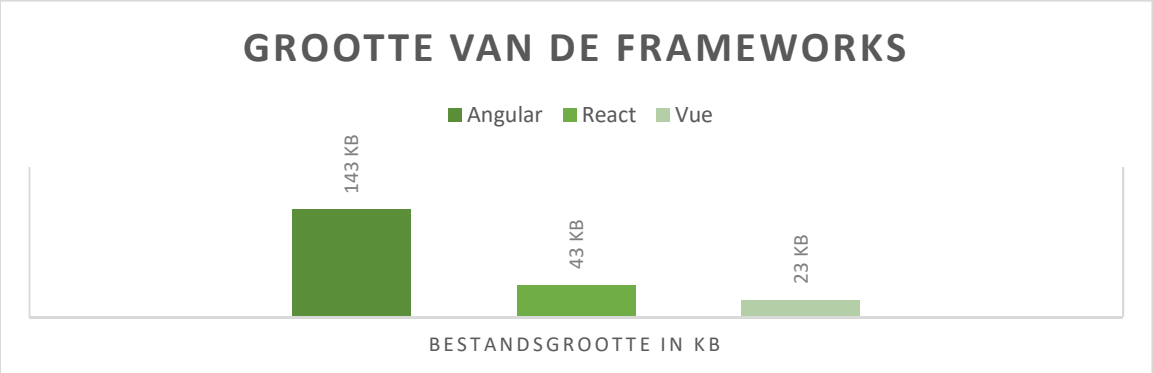



Uit deze test blijkt dat van de drie *framework* Angular de grootste achterstand heeft op gebied van snelheid. Vue is de winnaar met React dicht erachter.

### 2.5.4.2 Grootte van een project

De grootte van een *framework* heeft belang bij het inladen in de webbrowser. Wanneer iemand een website bezoekt wordt het volledige *framework* mee gekopieerd als dit nog niet aanwezig is. Als de grootte van een *framework* heel groot is zal het langer duren om een webpagina in te laden op een trage internetverbinding.

*Tabel 5 - Grootte van de frameworks*

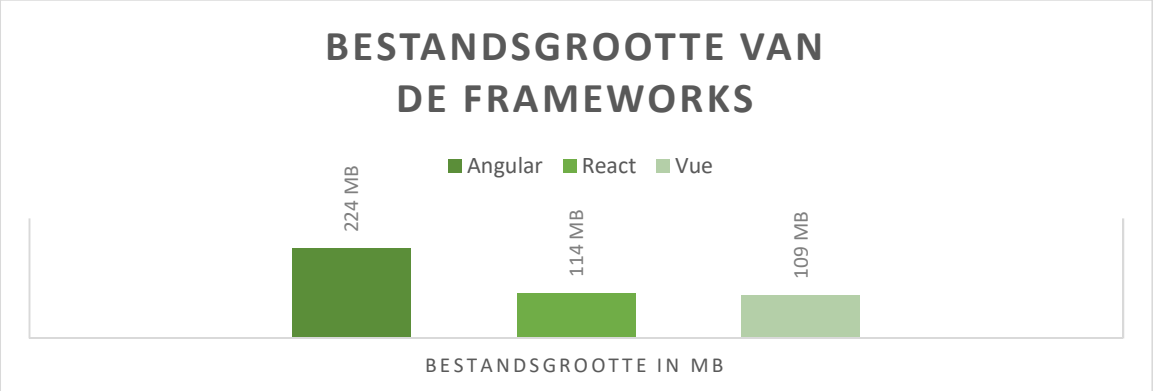



Uit deze kleine test is gebleken, dat Angular een grote bestandsgrootte heeft. Hierdoor zal het inladen van een Angular webpagina langer duren, als wanneer dezelfde pagina vanaf, bijvoorbeeld een Vue applicatie, zou binnengehaald worden. Op dit vlak zit React, tussen Vue en Angular in. Vue is duidelijk de winnaar in deze test.

Voor mensen die prototypes maken, kan het ook belangrijk zijn om te zien welke van de drie *frameworks* het meeste plaats innemen, van het geheugen in de computer. Dit is belangrijk om verschillende prototypes achter elkaar te kunnen maken, zonder de vorige te verwijderen. Wanneer de bestandsgrootte van een project te groot is, kan bij het maken van meerdere prototypes, het geheugen van een computer sneller vol raken.

Om dit te testen wordt de bestandsgrootte van drie nieuwe projecten bekeken, om na te gaan welke het meeste plaats inneemt. Uit deze test is gebleken, dat Angular ook hier dubbel zo groot is dan de andere twee *frameworks*. Hierdoor is het minder geschikt om veel kleine prototypes te maken. Vue en React zijn ongeveer even groot en zijn in deze categorie, beide de winnende *frameworks*.

*Tabel 6 - Bestandsgrootte van de frameworks*



#### 2.5.4.3 Conclusie snelheid

Wanneer er gekeken wordt naar de snelheid van *frameworks* is het meteen heel duidelijk dat Vue de winnaar is. Vue heeft niet alleen de kleinste bestandsgrootte maar is ook sneller in het verwerken van gegevens. Angular hiertegen is veel groter op gebied van bestandsgrootte en loopt hard achter bij het verwerken van gegevens. React staat ongeveer gelijk met Vue op gebied van snelheid en bestandsgrootte.

## 2.6 Besluit onderzoek

In het volgende besluit worden er concreet antwoorden gegeven op de verschillende onderzoeksvragen.

### 2.6.1 Welk *framework* biedt de beeste functionaliteit?

Het antwoord op deze vraag wordt gehaald uit de literatuurstudie. Hierin is er meer uitleg gegeven over de verschillende *frameworks* en welke functionaliteit deze allemaal bevatten.

Uit het onderzoek is gebleken dat van de drie *frameworks* React de minste functionaliteit bevat. Dit komt omdat het geen volledig *framework* is. Voor een simpele webapplicatie op te bouwen gaat er bij React al een aparte *library* binnengehaald moeten worden. Hiertegen zijn erbij zowel Angular als Vue al veel *libraries* meegegeven zoals bijvoorbeeld *Angular Routing* en *Vue Routing.*

Angular is uiteindelijk toch het winnende *framework* in deze categorie omdat deze meer functionaliteit biedt als Vue. Zo kunnen er bij Angular bijvoorbeeld services gebruikt worden om data te verwerken. Deze services kunnen aan de hand van *dependency injection* aangeroepen worden. Hierdoor zijn Angular applicaties veel eenvoudiger om te testen. Angular heeft ook de meest uitgebreide lijst van *libraries* die standaard meegeleverd worden.

### 2.6.2 Welk *framework* is het makkelijkst aan te leren?

Uit de drie *frameworks* is Angular duidelijk de moeilijkste om aan te leren voor nieuwe ontwikkelaars. Om Angular te kunnen gebruiken moet er eerst aangeleerd worden om te werken met *Typescript*, iets wat voor mensen die van Javascript komen niet evident is.

Als de tijd erin gestoken wordt om Angular goed te leren heb je wel veel meer kennis over hoe de frontend in elkaar zit. Het gebruik van MVC kan voor ontwikkelaars die uit de backend wereld komen het aanleren vereenvoudigen.

React zit tussen Angular en Vue op gebied van moeilijkheid. React biedt veel documentatie die het aanleren vereenvoudigd. Er is ook veel meer informatie rond te vinden, wat van pas komt voor nieuwe ontwikkelaars. Sommige zaken in React zijn zeer omslachtig zoals het aanroepen van functies in onderliggende componenten.

Vue is het makkelijkste om aan te leren uit de drie *frameworks*. Dankzij het gebruik van JSX kan er binnen Vue zeer snel begonnen worden met code schrijven. Alles zit logisch in elkaar en de manier van data beheren is makkelijk te begrijpen.

Bij het aanmaken van een nieuw project heeft Vue een voorsprong op Angular en React dankzij de grafische interface waarop alles ingesteld en gemonitord kan worden. Het gebruik van een CLI-tool kan voor nieuwe ontwikkelaars intimiderend zijn als men hier nog geen ervaring mee heeft. Voor deze ontwikkelaars is het gebruik van een UI veel makkelijker te begrijpen. Hierdoor kunnen ze sneller beginnen met het ontwikkelen van webapplicaties.

### 2.6.3 Welk *framework* is het meest performant?

Wanneer er gekeken wordt naar de snelheid van *frameworks* is het, dankzij de verschillende testen, meteen heel duidelijk dat Vue de winnaar is. Vue heeft niet enkel de kleinste bestandsgrootte maar is ook sneller in het verwerken van gegevens. Angular hiertegen is veel groter op gebied van bestandsgrootte en loopt hard achter bij het verwerken van gegevens.

Het verschil tussen Vue en React echter is heel klein maar Vue heeft toch altijd net de leiding op zowel bestandsgrootte als op het verwerken van gegevens.

### 2.6.4 Zijn de *frameworks* bruikbaar binnen grote webapplicaties?

Om een antwoord te geven op deze vraag wordt er weer teruggekeken naar de literatuurstudie. Terwijl React en Vue gebruik maken van JSX heeft Angular zich meer gefocust op het opsplitsen van code aan de hand van het *MVC pattern*. Dit heeft als voordeel dat de code opgesplitst is in verschillende bestanden die allemaal hu eigen doel hebben. Hierdoor is het eenvoudiger voor met grotere teams te werken aan één enkele applicatie. Er moet namelijk minder gezocht worden naar code en het begrijpen van bestaande code is eenvoudiger.

Door de *services* en *dependency injection* bij Angular is het ook eenvoudiger om je applicatie te testen, iets wat nodig is bij het maken van grote applicaties. Angular heeft ook al een grote community die dagelijks antwoorden geven op eventuele vragen, waardoor het oplossen van problemen eenvoudiger wordt.

Wanneer er tussen React en Vue gekeken wordt is React hier de winnaar. Vue is op dit moment nog te nieuw en de community errond is nog relatief klein. Wanneer er een grote applicatie bij dit *framework* gemaakt wordt is er geen zekerheid of eventuele problemen snel opgelost geraken. React hiertegen heeft de grootste community van de drie dankzij zijn ouderdom. Er zijn ook veel meer *libraries* speciaal ontworpen om te werken met React.

### 2.6.5 Wat is het beste frontend *framework* in 2019?

Door het antwoorden van de verschillende deelvragen, zijn er op deze vraag een aantal antwoorden mogelijk. Wanneer er een nieuwe ontwikkelaar zijn keuze doet om zijn eerste project in te maken is de beste keuze om te gaan voor Vue. Dit is omdat het een eenvoudig aan te leren *framework* is met veel meegeleverde functionaliteit.

Wanneer de snelheid van een *framework* het belangrijkste is, kan er gekozen worden tussen zowel Vue als React. Hier ligt de keuze er dan van af hoeveel ervaring de persoon al heeft. Met weinig ervaring is de beste keuze Vue met meer ervaring biedt React wat meer vrijheid op *frameworks* en libraries die je kan gebruiken.

Voor grote bedrijven die de focus leggen op een webapplicatie kan er best gekozen worden voor Angular. Dit omdat het ontworpen is om eenvoudig te kunnen testen en het gemakkelijker is om de SOLID-principes aan te houden.

## 2.7 Aanbevelingen

Een onderzoek is nooit volledig afgelopen, er zijn altijd nog zaken die extra onderzocht of vergeleken kunnen worden. In dit onderzoek is de focus vooral gelegd op de verschillen op gebied van functionaliteit en snelheid. Naar de toekomst toe kan dit onderzoekuitgebreid worden met meer criteria. Er is ook enkel de vergelijking gemaakt tussen Angular, React en Vue. Een mooie uitbreiding naar dit onderzoek zou zijn om meer *frameworks* te vergelijken. Zoals bijvoorbeeld Backbone.JS of Ember.js.

De keuze die Cubigo een aantal jaren geleden gemaakt heeft is uiteindelijk ook correct gebleken. Angular biedt de meeste functionaliteit en werkt het beste binnen grote applicaties. React zou ook een goede keuze geweest zijn door zowel de vrijheid als de snelheid dat het biedt.

## 2.8 Persoonlijke reflectie

Het kiezen van een onderzoeksvraag was niet zo vlot verlopen als ik gehoopt had. Mijn originele plan was om verschillende KPI-dashboards te vergelijken om zo het beste eruit te zoeken. Deze vraag had spijtig genoeg geen connectie met mijn stageopdracht. Hierdoor was mijn keuze al vrij beperkt. In school heb ik alleen leren werken met Angular dus om een vergelijking te maken tussen verschillende *frameworks* leek mij enorm leerrijk. Hierdoor is de vraag ontstaan om drie van de populairste *frameworks* te vergelijken.

Dit onderzoek was ook het eerste onderzoek dat ik ooit heb moeten doen, waar er een vergelijking gemaakt wordt tussen verschillende technologieën. Hiervoor had ik alleen nog maar onderzoek moeten doen rond één bepaald onderwerp. Het schrijven van een wetenschappelijke tekst is ook iets waar ik veel problemen mee had. Dankzij het maken van deze tekst is mijn kennis hierrond verbreed en gaat het schrijven van wetenschappelijke tekst steeds beter en sneller.

Het leren van de verschillende *frameworks* was iets wat mijn kennis rond *Javascript* heeft uitgebreid. In school was de kennis die we kregen hierrond heel beperkt. Het leren van deze *frameworks* heeft mij ook een andere kijk gegeven naar de toekomst. Voor ik aan dit onderzoek begon had ik geen idee welke kant ik op wou gaan met mijn verdere job. Ik vond het werken met frontend *frameworks* even interessant als het maken van een backend. Door dit onderzoek is mijn kijk hierop verandert. Het werken aan de frontend vond ik veel interessanter dan het werken aan de backend. Dit is omdat er een visuele representatie is die laat zien wat je allemaal gemaakt hebt. Dit is minder goed zichtbaar bij het schrijven van backend code.

Het resultaat dat dit onderzoek heeft gebracht is ook iets wat mij verraste. Ik had voor ik aan het onderzoek begon nog geen idee wat Vue was als *framework*. Uit het onderzoek is gebleken dat dit één van de beste *frameworks* is wanneer er gekeken wordt op snelheid en eenvoudigheid. Ik had verwacht dat dit *framework*, omdat het nog relatief nieuw is, het slechtste van de drie ging zijn.

De kennis die ik heb opgedaan in dit onderzoek ga ik zeker in de toekomst nog kunnen gebruiken. Als ik ook een project voor mezelf zou maken denk ik toch dat ik zou gaan voor Vue. Voor dit onderzoek zou dit niet eens een optie geweest zijn. De kennis van de verschillende *frameworks* ga ik sowieso ook kunnen gebruiken in de bedrijfswereld, waar er meestal gebruik gemaakt wordt van Angular of React.

# Conclusie

Binnen Cubigo wordt er een gebruiksvriendelijke, digitale webapplicatie gemaakt die het leven van oudere binnen gemeenschappen verbeterd. Dit doen ze door verschillende diensten aan te bieden die ze binnen zo een gemeenschap kunnen gebruiken. Dankzij dit platform is er een betere connectie tussen personeel, gebruikers en hun familie.

Aan het begin van de stageperiode was geen enkele manier voorzien om deze gemeenschappen aan te maken. Het was ook niet mogelijk om bestaande gemeenschappen te bewerken.

Een ander probleem dat er bestond binnen Cubigo was dat er geen mogelijkheid bestond om de gemeenschappen eenvoudig te beheren. Wanneer er veel gemeenschappen aangemaakt werden was er geen manier om te kijken welke er allemaal bestonden.

Aan het einde van de stageperiode was er de *community configurator* die een oplossing bood aan deze verschillende problemen. Aan de hand van deze online *configurator* kunnen er intern binnen Cubigo nieuwe gemeenschappen aangemaakt worden op een eenvoudige manier. Dit wordt gedaan door het doorlopen van een eenvoudige wizard die alle nodige instellingen bevat.

Deze tool zorgt er ook voor dat het mogelijk is om instellingen van bestaande gemeenschappen aan te passen. Het is nu ook mogelijk om te werken met een boomstructuur van gemeenschappen om het beheren te vereenvoudigen.

Het vinden van het meest geschikte frontend *framwork* passen volledig binnen deze opdracht. Er werd onderzoek gedaan om het beste frontend *framework* te vinden. In dit onderzoek werd er een vergelijking gemaakt tussen Angular, React en Vue. Deze *frameworks* werden getest op snelheid, functionaliteit, eenvoudigheid en meer.

Uit dit onderzoek is gebleken dat elk *framework* zijn eigen plaats heeft. Zo is een Angular applicatie meer geschrikt voor het maken van een grote applicatie of voor het gebruik binnen teams van ontwikkelaars. Een Vue applicatie is op zijn beurt dan weer meer geschikt voor nieuwe ontwikkelaars die een frontend *framework* willen aanleren. Op gebied van snelheid heeft Vue de leiding met React er vlak achter.

# Bibliografie


[1] Angular, „Angular Single Page Applications (SPA): What are the Benefits?," [Online]. Available: https://blog.angular-university.io/why-a-single-page-application-what-are-the-benefits-what-is-a-spa/.

[2] „TypeScript Language," [Online]. Available: https://www.typescriptlang.org/.

[3] M. Petrosyan, „Angular 5 vs. React vs. Vue," [Online]. Available: https://itnext.io/angular-5-vs-react-vs-vue-6b976a3f9172.

[4] Angular CLI, „CLI Overview and Command Reference," [Online]. Available: https://angular.io/cli.

[5] C. Wodehouse, „Upwork," [Online]. Available: https://www.upwork.com/hiring/development/angularjs-basics/.

[6] M. M, „Frameworks in AngularJS and Angular 2," [Online]. Available: https://www.sitepoint.com/angularjs-vs-angular/.

[7] B. Peter, „Microsoft TypeScript the Javascript we need, or a solution looking for a problem?," 3 10 2012. [Online]. Available: https://arstechnica.com/information-technology/2012/10/microsoft-typescript-the-javascript-we-need-or-a-solution-looking-for-a-problem/.

[8] Angular, „Introduction to components," [Online]. Available: https://angular.io/guide/architecture-components.

[9] Angular, „Injecting services," [Online]. Available: https://angular.io/guide/dependency-injection.

[10] P. Krill, „React: Making faster, smoother UIs for data-driven Web apps," 15 Mei 2014. [Online]. Available: https://www.infoworld.com/article/2608181/react--making-faster--smoother-uis-for-data-driven-web-apps.html.

[11] React.js, „Why JSX?," [Online]. Available: https://reactjs.org/docs/introducing-jsx.html.

[12] React.js, „Embedding Expressions in JSX," [Online]. Available: https://reactjs.org/docs/introducing-jsx.html.

[13] React.js, „Virtual DOM and Internals," [Online]. Available: https://reactjs.org/docs/faq-internals.html.

[14] O. Filipova, „Vue.js history," [Online]. Available: https://www.oreilly.com/library/view/learning-vuejs-2/9781786469946/ch01s02.html.

[15] A. Gore, „What does the virtual DOM do?," 26 September 2016. [Online]. Available: https://medium.com/js-dojo/whats-new-in-vue-js-2-0-virtual-dom-dc4b5b827f40.

[16] Vue.js, „Render Functions & JSX," [Online]. Available: https://vuejs.org/v2/guide/render-function.html.

[17] Vue.js, „Components of the System," 17 July 2018. [Online]. Available: https://cli.vuejs.org/guide/#components-of-the-system.

[18] Developer, „Is React library or a framework?,” [Online]. Available: https://developer.com/is-reactjs-library-or-a-framework-a14786f681a0.

[19] A. Dorol, „Building the Vue.js SPA,” [Online]. Available: https://snipcart.com/blog/building-a-vuejs-spa-to-sell-developer-tees.

[20] Kirupa, „Creating a Single-Page App in React using React Router,” [Online]. Available: https://www.kirupa.com/react/creating_single_page_app_react_using_react_router.htm.

[21] R. Carvalho, „SINGLE PAGE APPLICATIONS: WHEN AND WHY YOU SHOULD USE THEM,” 29 November 2017. [Online]. Available: https://www.scalablepath.com/blog/single-page-applications/.

[22] C. Coyier, „What is the DOM?,” [Online]. Available: https://css-tricks.com/dom/.

[23] A. Neagoie, „Tech Trends Showdown: React vs Angular vs Vue,” [Online]. Available: https://medium.com/zerotomastery/tech-trends-showdown-react-vs-angular-vs-vue-61ffaf1d8706.

[24] TechMagic, „React vs Angular vs Vue.js — What to choose in 2019?,” 16 Maart 2018. [Online]. Available: https://medium.com/@TechMagic/reactjs-vs-angular5-vs-vue-js-what-to-choose-in-2018-b91e028fa91d.

[25] „Star History,” 05 04 2019. [Online]. Available: https://star-history.t9t.io/#angular/angular&vuejs/vue&facebook/react&angular/angular&facebook/react&vuejs/vue.

[26] J. Neuhaus, „Angular vs. React vs. Vue: A 2017 comparison,” 28 Augustus 2017. [Online]. Available: https://medium.com/unicorn-supplies/angular-vs-react-vs-vue-a-2017-comparison-c5c52d620176.

[27] S. Daityari, „Angular vs Vue vs React: Which Framework to Choose in 2019,” 10 Januari 2019. [Online]. Available: https://www.codeinwp.com/blog/angular-vs-vue-vs-react/.

[28] S. INDIA, „React vs Angular vs Vue.js: A Complete Comparison Guide,” [Online]. Available: https://www.spec-india.com/blog/react-vs-angular-vs-vue-js-a-complete-comparison-guide/.

[29] T. Media, „Traversy Media,” 2019. [Online]. Available: https://www.traversymedia.com/.

[30] w3schools, „AngularJS Data Binding,” [Online]. Available: https://www.w3schools.com/angular/angular_databinding.asp.