HO
GENT

Faculteit Bedrijf en Organisatie

Een vergelijking tussen Mithril.js, Angular en React

Jelle Callewaert

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Lotte Van Steenberghe
Co-promotor:
TBD

Instelling: —

Academiejaar: 2019-2020

Eerste examenperiode

Faculteit Bedrijf en Organisatie

Een vergelijking tussen Mithril.js, Angular en React

Jelle Callewaert

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Lotte Van Steenberghe
Co-promotor:
TBD

Instelling: —

Academiejaar: 2019-2020

Eerste examenperiode

Woord vooraf

Samenvatting

Inhoudsopgave

Lijst van figuren

Lijst van tabellen

1. Inleiding

De inleiding moet de lezer net genoeg informatie verschaffen om het onderwerp te begrijpen en in te zien waarom de onderzoeksvraag de moeite waard is om te onderzoeken. In de inleiding ga je literatuurverwijzingen beperken, zodat de tekst vlot leesbaar blijft. Je kan de inleiding verder onderverdelen in secties als dit de tekst verduidelijkt. Zaken die aan bod kunnen komen in de inleiding (**Pollefliet2011**):

- context, achtergrond
- afbakenen van het onderwerp
- verantwoording van het onderwerp, methodologie
- probleemstelling
- onderzoeksdoelstelling
- onderzoeksvraag
- ...

1.1 Probleemstelling

Uit je probleemstelling moet duidelijk zijn dat je onderzoek een meerwaarde heeft voor een concrete doelgroep. De doelgroep moet goed gedefinieerd en afgelijnd zijn. Doelgroepen als "bedrijven," "KMO's," systeembeheerders, enz. zijn nog te vaag. Als je een lijstje kan maken van de personen/organisaties die een meerwaarde zullen vinden in deze bachelorproef (dit is eigenlijk je steekproefkader), dan is dat een indicatie dat de doelgroep goed gedefinieerd is. Dit kan een enkel bedrijf zijn of zelfs één persoon (je co-promotor/opdrachtgever).

1.2 Onderzoeksvraag

Wees zo concreet mogelijk bij het formuleren van je onderzoeksvraag. Een onderzoeksvraag is trouwens iets waar nog niemand op dit moment een antwoord heeft (voor zover je kan nagaan). Het opzoeken van bestaande informatie (bv. “welke tools bestaan er voor deze toepassing?”) is dus geen onderzoeksvraag. Je kan de onderzoeksvraag verder specifiëren in deelvragen. Bv. als je onderzoek gaat over performantiemetingen, dan

1.3 Onderzoeksdoelstelling

Wat is het beoogde resultaat van je bachelorproef? Wat zijn de criteria voor succes? Beschrijf die zo concreet mogelijk. Gaat het bv. om een proof-of-concept, een prototype, een verslag met aanbevelingen, een vergelijkende studie, enz.

1.4 Opzet van deze bachelorproef

De rest van deze bachelorproef is als volgt opgebouwd:

In Hoofdstuk 2 wordt een overzicht gegeven van de stand van zaken binnen het onderzoeksdomein, op basis van een literatuurstudie.

In Hoofdstuk 3 wordt de methodologie toegelicht en worden de gebruikte onderzoekstechnieken besproken om een antwoord te kunnen formuleren op de onderzoeksvragen.

In Hoofdstuk 4, tenslotte, wordt de conclusie gegeven en een antwoord geformuleerd op de onderzoeksvragen. Daarbij wordt ook een aanzet gegeven voor toekomstig onderzoek binnen dit domein.

2. Stand van zaken

2.1 Mithril.js

Mithril is een modern client-side framework, gemaakt voor het bouwen van single page applications. Het is klein in download, snel in performance en voorziet routing en XHR tools. (Mithril, 2019c)

Dit framework is vergelijkbaar met React, maar het is eenvoudiger en meer capabel. Mithril vervangt de behoefte aan bibliotheken zoals jQuery. De kleine bestandsgrootte en gemakkelijke API zorgt ervoor dat Mithril ideaal is om embedded JavaScript-widgets of user interfaces die hoge performance vereisten hebben, te maken. (Gilbert, 2018)

Mithril was oorspronkelijk geschreven door Leo Horie. Het framework werd uitgebreid en verbeterd tot wat het tegenwoordig is dankzij het werk van de community. (Mithril, 2019b)

2.1.1 Virtuele DOM nodes

Virtueel DOM

Een virtuele DOM-structuur is een JavaScript-datastructuur die een `DOM tree` beschrijft. Het bestaat uit een aantal geneste virtuele DOM nodes, genaamd `vnodes` (Mithril, 2019d)

Meestal worden virtuele DOM's opnieuw opgebouwd bij een rendercyclus. Deze cyclus vindt plaats na een event of na data changes. Bij zo'n rendercyclus bekijkt Mithril de de vorige versie van de vnode-structuur en wijzigt enkel de DOM-elementen op plaatsen waar een aanpassing plaatsvond. Nieuwe vnodes maken is goedkoper dan het wijzigen van de

DOM. (Mithril, 2019d)

Basis

Vnodes zijn JavaScript objecten die delen van het DOM voorstellen. De virtuele DOM-machine van Mithril verbruikt de vnodes-structuur en stelt een DOM-structuur op. (Mithril, 2019d)

Vnodes kunnen aangemaakt worden aan de hand van het `m()` hyperschrift. Hyperschrift kan ook `componenten` verbruiken. (Mithril, 2019d)

Structuur

Vnodes zijn JavaScript objecten die een element voorstellen. Ze kunnen volgende eigenschappen hebben:

tag	De `nodeName` van een DOM element. De tag kan ook een string zijn als de vnode een fragment, een text node of vertrouwd HTML node is.
key	De waarde die gebruikt wordt om een DOM element te mappen naar zijn respectievelijke item in een array van data.
attrs	Een hashmap van DOM attributen, events, eigenschappen en lifecycle methodes.
children	Een array van de kinderen van de vnode. Meestal zijn dit ook vnodes, maar in sommige gevallen kan dit een string, een number of een boolean zijn.
text	Dit wordt enkel gebruikt als een vnode enkel een text node heeft als kind. Dit wordt enkel gedaan om performance redenen.[1]
dom	Verwijst naar het element dat correspondeert met de vnode.
domSize	Deze eigenschap wordt enkel gebruikt in fragmenten en vertrouwde HTML nodes. Het stelt het aantal DOM elementen voor die de vnode vertegenwoordigt.
state	Een object dat voorzien wordt door de `core engine` en bewaard wordt doorheen cyclussen. Bij een POJO component vnode erft het state object de eigenschappen van de component klasse/object over.
events	Een object dat de event handlers opslaat zodat deze later verwijderd kunnen worden door gebruik te maken van de DOM API. Net zoals het state-object wordt deze ook bewaard doorheen de rendercyclussen. Deze eigenschap wordt intern gebruikt door Mithril en zou nooit gebruikt of aangepast mogen worden.
instance	Een object voor componenten. Het is een opslaglocatie voor de waarde die de view methode teruggeeft. Deze eigenschap wordt ook intern gebruikt door Mithril en zou nooit gebruikt of aangepast mogen worden.

(Mithril, 2019d)

Soorten vnodes

In Mithril bestaan er 5 verschillende soorten vnodes. Deze wordt bepaald door de `tag` attribuut.

Element	Vertegenwoordigt een DOM element.
Fragment	Vertegenwoordigt een lijst van DOM elementen waarvan de parent ook andere DOM elementen kan bevatten die zicht niet in de lijst bevinden.
Text	Vertegenwoordigt een DOM text node
Trusted HTML	Vertegenwoordigt een lijst van DOM elementen van een HTML string
Component	Vertegenwoordigt het DOM dat gegenereerd wordt bij het renderen van de component. Dit kan enkel als de `tag` een component is die een `view` methode bevat.

(Mithril, 2019d)

In een virtuele DOM structuur moet alles een vnode zijn, dus ook text. De `m()` utility normaliseert zijn `children` argument en verandert text in text vnodes en geneste arrays in fragment vnodes. (Mithril, 2019d)

Het eerste argument van de `m()` functie kan enkel een element tag naam of een component zijn. Trusted HTML vnodes kunnen gemaakt worden aan de hand van de methode `m.trust()` (Mithril, 2019d)

Monomorphische klasse

De `mithril/render/vnode` wordt gebruikt door Mithril om alle vnodes aan te maken. Dit zorgt ervoor dat moderne JavaScript machines optimaal virtuele DOM structuren kunnen vergelijken door altijd vnodes te compileren met deze ene verborgen klasse. (Mithril, 2019d)

Anti-patterns vermijden

Vnodes vertegenwoordigen de staat van het DOM op een gegeven tijdstip. De rendermachine van Mithril gaat ervan uit dat een vnode die opnieuw gebruikt wordt onaangepast is.Het aanpassen van een vnode die gebruikt werd in een vorige rendering zal resulteren in een `undefined` gedrag. (Mithril, 2019d)

Het is mogelijk om vnodes te hergebruiken om een render cycle te vermijden, maar het is beter om gebruik te maken van de `onbeforeupdate` hook te gebruiken. (Mithril, 2019d)

2.1.2 Componenten

Structuur

Componenten zijn een mechanisme die gebruikt worden om delen van de view in te kapselen. Hierdoor wordt de code gemakkelijker om te structureren of om opnieuw te gebruiken. Elk JavaScript object dat een `view` methode heeft is een Mithril component. Deze componenten kunnen gebruikt worden door gebruik te maken van de `m()` functie. (Mithril, 2019a)

Lifecycle methodes

Componenten hebben dezelfde lifecycle methodes als vnodes. Bij elke lifecycle methode wordt een `vnode` als argument meegegeven, net zoals bij de `view` methode. Zoals de andere types vnodes kunnen componenten extra lifecycle methodes hebben wanneer ze gebruikt worden als vnode type. (Mithril, 2019a)

Lifecycle methodes in vnodes overschrijven de component methodes niet, ook niet omgekeerd. De lifecycle methodes van componenten worden altijd opgeroepen na de methodes van de vnode. Dezelfde naam kiezen voor callback functies als de lifecycle methode namen zou dus altijd vermeden moeten worden. (Mithril, 2019a)

Data meegeven aan componenten

Om data mee te geven aan componenten kan door een `attrs` object mee te geven als tweede parameter in de hyperschrift functie. Deze data kan worden geraadpleegd in de view van de component of in de lifecycle methodes via `vnode.attrs` (Mithril, 2019a)

State

Zoals alle virtuele DOM nodes kunnen component vnodes een state hebben. Op deze manier kunnen zaken als object georiënteerde structuren, inkapseling en separation of concerns ondersteund worden. (Mithril, 2019a)

Een component aanpassen zorgt niet voor een DOM update. Dit gebeurt wanneer een event handler getriggerd wordt, wanneer een HTTP request (gemaakt via `m.request`) vervuld is of wanneer de browser navigeert naar een andere route. Om een redraw geforceerd te laten doorgaan kan gebruik gemaakt worden van `m.redraw()`. (Mithril, 2019a)

Closure component state

Het is mogelijk om component state te gebruiken bij POJO componenten, maar dat is in het algemeen niet de beste aanpak. Een betere manier is door gebruik te maken van een closure component. Dat is simpelweg een wrapper functie die een POJO component instantie returnt die op zich zijn eigen closed-over scope meedraagt.

Met deze closure componenten kan state bewaard worden door variabelen die gedeclareerd staan in de buitenste functie. Ook functies die in de closure component gedeclareerd staan hebben toegang tot de state variabelen.

Deze closure componenten worden op dezelfde manier gebruikt als POJO componenten. Het voordeel hiervan is dat er geen gebruik gemaakt moet worden van `this`.

POJO component state

Het is altijd aangeraden om closure componenten te gebruiken wanneer state moet bijgehouden worden. Er kan echter toch state bijgehouden worden in POJO componenten en die kan dan op drie manieren opgevraagd worden, namelijk als blueprint bij initialisatie, via `vnode.state` of via het `this` keyword in component methodes.

Classes

Componenten kunnen ook geschreven worden aan de hand van het `class` sleutelwoord. Deze class componenten moeten een `view()` methode definiëren. Deze methode wordt gedetecteerd via `.prototype.view` om in de DOM structuur gerenderd te worden. Ook klasse componenten worden op dezelfde manier gebruikt als normale componenten. Ook in klasse componenten kan state bijgehouden worden en deze kan beheerd worden door zijn attributen en methodes. State wordt aangeroepen via het `this` sleutelwoord, maar om de juiste referentie te verkrijgen moeten arrow functions gebruikt worden voor de event handler callbacks.

Gemixte soorten componenten

Componenten kunnen gelijk welke soort component als kind hebben. Er zijn geen restricties op de soort kinderen qua component.

Speciale attributen

Mithril gebruikt enkele property keys. Het is aangeraden om deze te vermijden in/als attributen van normale componenten. Deze keys zijn

- Lifecycle methodes, nl. `oninit`, `oncreate`, `onbeforeupdate`, `onupdate`, `onbeforeremove` en `onremove`
- `key`
- `tag`

Anti-patterns vermijden

Hoewel Mithril heel flexibel is, is het toch afgeraden om bepaalde patterns te vermijden.

'Fat' componenten vermijden In het algemeen is een 'fat' component een component met eigen instance methodes. Functies in `vnode.state` of in `this` zijn af te raden omdat deze niet hergebruikt kan worden in andere componenten. Het is ook eenvoudiger om de code te refactoren wanneer deze in de data laag staat in plaats van in state. Code die hergebruikt moet worden, wordt best in een module geplaatst.

vnode.attrs niet meegeven andere componenten Om de interface flexibel te maken en een simpele implementatie te garanderen lijkt het logisch om alle attributen (`vnode.attrs`) door te geven aan de child componenten. In plaats daarvan zouden slechts enkele attributen meegegeven mogen worden aan child componenten.

Vermijd het maken van component definities in views Als er een component gecreëerd word binnen een view functie, dan zal elke redraw een andere clone hebben van de component. Dit betekent dat componenten die dynamisch aangemaakt werden via een factory ook altijd opnieuw van nul worden aangemaakt.

Vermijd het maken van component instanties buiten views Als een component instantie buiten een view functie gemaakt wordt, dan zullen toekomstige redraws de vnode vergelijken, maar omdat deze dezelfde referentie krijgt, zal deze vnode overgeslagen worden en er zal geen update gebeuren.

2.2 Angular

Angular is een framework dat het gemakkelijk maakt om webapplicaties te bouwen. Angular combineert templates, dependency injectie, end to end tooling en geïntegreerde best practices om hindernissen in de ontwikkeling op te lossen. Angular stelt ontwikkelaars in staat om applicaties te maken voor op het web, mobiel of desktop. (Lotanna, 2019)

Angular kan gezien worden als een volwaardig MVC[2] framework. Het wordt zo beschouwd omdat Angular de structuur van de applicatie in ontwikkeling sterk wilt regulieren. Angular voorziet standaard veel functionaliteit. Aan de andere kant zorgt dit wel voor minder flexibiliteit. Angular verstrekt onderstaande functionaliteit. (Hamedani, 2018)

- Templates
- XSS[3] bescherming
- Dependency Injectie
- Component CSS encapsulatie
- Hulpmiddelen voor het unit-testen van componenten
- Ajax requests
- Routering

[2]Model View Controller
[3]Cross-Site Scripting

- Formulieren

2.2.1 Geschiedenis van angular

Google, de ontwikkelaar van AngularJS, kondigde eind 2014 aan dat Angular 2 een volledige herschrijving van AngularJS zou zijn. Ze zouden zelf een nieuwe programmeertaal `AtScript` ontwikkelen die speciaal voor Angular 2 bedoeld was. Toen begon Microsoft decorators te ondersteunen in hun taal TypeScript, waardoor deze de aanbevolen taal werd voor de ontwikkeling van Angular 2 applicaties. Het is echter ook mogelijk om applicaties te ontwikkelen in JavaScript (ECMAScript 5 en ECMAScript 6) en in Dart. (Fain, 2016)

Redenen voor herschrijving

Het team van Google besliste om geen grote updates meer te doen aan AngularJS omdat ze, naast het verbeteren van features en prestaties, Angular meer future-proof wilden maken door gebruik te maken van web componenten en ECMAScript 6. Het resultaat was Angular 2, verrijkt met enorm veel features. De voornaamste nieuwe features waren niet mogelijk in AngularJS waardoor een herschrijving nodig was. Als eerste was de ideologie veranderd. Bij AngularJS lag de focus op databinding en templates. Het belangrijkste doel was om af te komen van het traditionele proces voor DOM-handling via JavaScript's jQuery. Daarnaast moest het framework ook klaar voor de toekomst zijn. Sinds de meeste browsers ECMAScript 6 gebruiken, zullen applicaties dus ook ontwikkeld moeten worden in ES6. Angular 2 biedt die kans aan, met 100% browser achterwaartse compatibiliteit. Ook maakt Angular 2 gebruik van web componenten. Dit zijn JavaScript en HTML modules die gemaakt worden voor een specifieke taak in de applicatie. Als laatste werden routers, dependency injection, dynamic loading en async templating verbeterd of toegevoegd. (Shan, 2015)

2.2.2 Angular architectuur en werking

De basis-bouwstenen van Angular applicaties zijn NgModules. Deze voorzien compilatie context voor componenten en verzamelen gerelateerde code in functionele sets. Een Angular applicatie wordt bepaald door een set van NgModules. Elke applicatie heeft een root module dat bootstrapping mogelijk maakt. Daarnaast kunnen nog meerdere feature modules toegevoegd worden, maar deze zijn niet verplicht. (Angular, 2019a)

Componenten definiëren views, dit zijn sets van schermelementen die Angular kan gebruiken en aanpassen aan de hand van de logica in de applicatie. Daarnaast kunnen componenten services gebruiken die specifieke functionaliteiten voorziet die niet verbonden zijn met views. Deze services kunnen geïnjecteerd worden in componenten met behulp van dependency injection. Hierdoor wordt de code herbruikbaar en modulair. (Angular, 2019a)

Zowel componenten als services zijn gewone klassen met decorators. Deze kenmerken hun type en voorzien metadata. Angular gebruikt deze metadata om te weten hoe deze

klassen gebruikt kunnen worden. De metadata van een component verbindt de component met een template die een view definiëert. Een template combineert HTML met Angular directives en binding markup die Angular toestaan om de HTML aan te passen voordat het getoond wordt op het scherm. Daarnaast voorziet de metadata van een service de informatie die Angular nodig heeft om het beschikbaar te maken voor componenten via dependency injection. (Angular, 2019a)

Modules

Angular applicaties zijn `modulair`. NgModules zijn containers die samenhangende code van een bepaald domein of bepaalde workflow bevat. Ze kunnen componenten, services en andere code-bestanden die behoren tot de scope van de module bevatten. Ook kan functionaliteit geïmporteerd worden van een andere NgModule. Daarnaast kan een NgModule bepaalde functionaliteiten exporteren die andere NgModules kunnen gebruiken. Elke applicatie heeft minstens een NgModule klasse, de root module. Deze heet gewoonlijk AppModule en bevind zich in een bestand genaamd `app.module.ts`. De applicatie wordt gestart door deze root module te bootstrappen. Zoals eerder vermeld kunnen dus extra modules toegevoegd worden als feature modules. De AppModule kan deze extra modules omvatten in een hiërarchie met onbepaalde diepte. (Angular, 2019d)

Een NgModule wordt gedefinieerd door een klasse met de `@NgModule()` decorator. Deze decorator is een functie die een metadata object verwacht. De properties van dit object worden omschreven in de module. Enkele van de belangrijkste properties van een module zijn als volgt (Angular, 2019d):

- Declarations: set van componenten, directives en pipes die tot de module behoren.
- Exports: set van declarations die zichtbaar en bruikbaar moeten zijn in component templates van andere modules.
- Imports: set van modules die nodig zijn in component templates die gedefinieerd zijn in de huidige module.
- Providers: set van services (meer uitleg hier)
- Bootstrap: Dit is de view van de main applicatie, genaamd de root component. Deze host alle andere views van de applicatie. Dit kan enkel voor de root NgModule.

Modules voorzien compilatie context voor de componenten die ze declareren. De root module heeft altijd een root component die aangemaakt wordt tijdens de bootstrap. Elke module kan extra componenten omvatten die met behulp van de router geladen kunnen worden of aangemaakt kunnen worden door de template. (Angular, 2019d)

Een view wordt gedefinieerd door een component en zijn template. Een component kan een view hiërarchie bevatten die toestaat om een complexe samenstelling van een scherm te definiëren. Deze samenstelling kan als een eenheid aangemaakt, aangepast en verwijderd worden. De hiërarchie bestaat uit één enkele host view. Deze view kan de root van een hiërarchie zijn die `embedded views` kan bevatten, die op hun beurt host view zijn van een andere component. Deze componenten kunnen deel zijn van dezelfde module of geïmporteerd worden van een andere module. (Angular, 2019d)

De NgModules van Angular zijn totaal verschillend van de JavaScript modules. In JavaScript is elk bestand een module en elk object gedefinieerd in dat bestand behoort tot die module. Die objecten kunnen geëxporteerd worden door de module door gebruik te maken van het sleutelwoord `export`. Andere modules kunnen dan gebruik maken van dit publiek object door het sleutelwoord `import` te gebruiken. (Angular, 2019d)

Componenten

Elke applicatie heeft naast een module ook minstens één component, de root component. Die legt de verbinding tussen een component's view hiërarchy en het DOM. Elke component definiëert een klasse die applicatiedata en -logica bevat. Deze wordt ook verbonden met een HTML-template die een view definiëert. De `@Component()` decorator identificiëert de onderstaande klasse als een component en voorziet de template en alle specifieke metadata voor die component. Deze metadata is net zoals bij NgModules een functie die een object verwacht. (Angular, 2019a)

Een component is dus de meest basis bouwsteen van een Angular applicatie. Zo'n applicatie bestaat meestal uit een `tree` van Angular componenten. Componenten zijn een subset van `directives` die altijd geassociëerd worden met een template. Echter in tegenstelling tot `directives` kan slechts een component geïnstantiëerd worden per template-element. Daarnaast kan het gedrag van een component tijdens runtime ook geprogrammeerd worden door gebruik te maken van `lifecycle hooks` (**REFERENCE**)

De voornaamste properties van het metadata-object voor componenten zijn als volgt (Angular, 2019b)

- ChangeDetection: De change-detection strategie voor de component. Wanneer een component geïnstantiëerd wordt, maakt Angular een change detector aan die verantwoordelijk is voor het uitdragen van de bindingen van de component. Er zijn 2 mogelijkheden nl. ChangeDetection#OnPush: CheckOnce(on demand) en ChangeDetection#Default: CheckAlways
- ViewProviders: definiëert een set van injecteerbare objecten die zichtbaar zijn voor de DOM kinderen van de component's view.
- Template: Een inline template voor een Angular component.
- TemplateUrl: Het relatieve of absolute pad naar een template bestand voor een Angular component.
- Styles: Eén of meerdere inline CSS stylesheets die gebruikt kunnen worden in de component
- StyleUrls: Eén of meerdere relatieve of absolute paden naar bestanden die een CSS stylesheet bevatten om te gebruiken in de component.
- EntryComponents: een set van componenten die samen gecompileerd moeten worden met de component. Voor elke component hier gedefinieerd creëert Angular een `ComponentFactory` en slaat deze op in de `ComponentFactoryResolver`

Daarnaast erft het ook volgende properties over van de `@Directive()` decorator (Angular, 2019b)

- Selector: de CSS selector dat de directive identificiëert in een template en de instantiatie van de directive start.
- Inputs: set van datagebonden properties van de directive.
- Outputs: set van eventgebonden properties.
- Providers: configureert de injector van de directive met een token dat gemapt wordt naar een provider van een dependency

Angular componenten zijn een subset van directives die altijd verbonden zijn met een template. Anders dan andere directives kan maar een component geïnstantiëerd worden per element in een template. Een component moet tot een NgModule behoren zodat die beschikbaar zou zijn voor andere componenten. (Angular, 2019a)

Een template combineert HTML met Angular markup die HTML-elementen kan aanpassen nog voordat deze getoond worden op het scherm. (Angular, 2019a)

Template directives voorzien logica en binding markup verbindt applicatie data met het DOM. Er zijn twee soorten data bindings. (Angular, 2019a)

- `Event binding` staat toe om de applicatie te laten reageren op gebruikers invoer in de omgeving door het aanpassen van applicatie data.
- `Property binding` staat toe om berekende waarden uit de applicatie uit te schrijven in de HTML.

Voordat een view getoond wordt op een scherm zal Angular eerst de directives evalueren en de syntax van de binding in de template uitzoeken zodat de HTMLelementen en het DOM aangepast kunnen worden, afhankelijk van de applicatie data en logica. Angular staat ook `two-way-binding` toe. Dit wil zeggen dat zowel aanpassingen in het DOM zoals gebruikers invoer de applicatie data aanpassen en applicatieberekeningen ook weergegeven zullen worden in de HTML. (Angular, 2019a)

Een template kan gebruik maken van `pipes` om de UX[4] te verbeteren door waarden te transformeren voor weergave. Angular voorziet enkele voorgedefinieerde pipes voor standaard transformaties, maar pipes kunnen ook zelf gedefinieerd worden met een eigen transformatie. (Angular, 2019a)

Services en DI

Voor data of logica die niet tot een specifieke view behoort en over de hele applicatie voorzien moet worden, kan een service klasse gebruikt worden. Een service wordt gedefinieerd met de `@Injecatble()` decorator. (Angular, 2019a)

Deze decorator voorziet de metadata die de klasse beschikbaar maakt voor creatie aan een `Injector` zodat deze geïnjecteerd kan worden als een dependency. Voor elke dependency moet een provider voorzien worden zodat de injector van deze provider gebruik kan maken om een dependency op te halen of een nieuwe aan te maken. Voor een service is dit standaard de service klasse zelf. Angular creëert een globale injector tijdens het bootstrap

[4] User Experience: het gevoel en de ervaring dat een gebruiker opdoet

proces die gebruikt kan worden over de hele applicatie. Daarnaast kunnen extra injectors aangemaakt worden wanneer dat nodig is. (Angular, 2019e)

Dependency Injection (DI) is een belangrijk applicatie design pattern. Angular bevat een eigen DI framework dat kenmerkend gebruikt wordt in het design van Angular applicaties om op een efficiënte en modulaire manier om te gaan met het creëren van klassen. Een klasse verzoekt dependencies aan een externe bronnen in plaats van deze zelf aan te maken. In Angular voorziet het DI framework de dependencies aan een klasse zodra deze geïnstantieerd is. (Angular, 2019c)

Services die aangemaakt worden via het Angular CLI commando `ng generate service [servicenaam]` registreert een provider voor de service in de root injector door `'root'` toe te kennen aan het `providedIn` property van het metadata object. Angular creëert dan één enkele instantie die geïnjecteerd wordt in elke klasse die daar om vraagt. Het `providedIn` property kan ook een specifiek type zijn. Hierdoor kan Angular de applicatie optimaliseren door de service te verwijderen uit de gecompileerde app wanneer dat type niet gebruikt wordt. Services kunnen ook geregistreerd worden in een specifieke NgModule. De service zal dan beschikbaar zijn voor alle componenten van die module. Om dit te bekomen, moet de service aan de `providers` set in de module toegevoegd worden. Als laatste kan een provider voor een service ook op component level geregistreerd worden. Op deze manier wordt een nieuwe instantie van de service gemaakt bij elke nieuwe instantie van de component. Dit kan door deze toe te voegen aan de `providers` property van het metadata object van de component. (Angular, 2019e)

Een component kan gebruik maken van een service of andere dependencies door deze te injecteren in de component met behulp van de `Injector`. Angular creëert alle injectors zelf, dus er moeten geen injectors aangemaakt of geïmplementeerd worden. Een injector maakt de dependencies aan en houdt een container van deze instanties bij zodat deze hergebruikt kunnen worden als dat mogelijk is. Op deze manier krijgt de component toegang tot de service of kan deze de dependency gebruiken. (Angular, 2019e)

Wanneer een nieuwe instantie van een component aangemaakt wordt, bepaalt Angular welke dependencies de component nodig heeft door te kijken naar de types van de parameters in de constructor. Dan checkt de injector de container met bestaande instanties. Als deze de gevraagde instantie vindt, wordt die teruggegeven. Indien dat niet het geval is, wordt een nieuwe instantie aangemaakt van de dependency en wordt deze opgeslagen in de container en teruggegeven. (**REFERENCE**)

Samenwerking

Al deze items vormen de basis over de hoofdzakelijke building blocks van een Angular applicatie. De afbeelding xx toont hoe deze bouwstenen met elkaar verbonden zijn en samenwerken.

Een component en een template vormen samen een Angular view. Een `@Component()` decorator voegt de metadata toe, waarbij een pointer zit naar de geassocieerde template. Directives en binding markup in een template kunnen de view aanpassen afhankelijk van

applicatie data en logica. De dependency injector kan services in een component voorzien. Afhankelijk van de provider van de service wordt deze opnieuw aangemaakt. (Angular, 2019a)

2.3 React

React, aan de andere kant, is een library met als doel ontwikkelaars te helpen met het bouwen van User Interfaces. De structuur van deze UI's worden voorgesteld als een boom waarbij de knopen componenten zijn. Een component bestaat uit zowel HTML als JavaScript die de logica beschrijft om deze te tonen. (Baer, 2018)

2.3.1 Geschiedenis van React

Reden voor creatie

React is ontstaan binnen de Facebook Ads Org. Het team van Facebook bouwde standaard MVC client-side applicaties met two-way-binding en templates. In deze apps luisteren de views naar aanpassingen in de models en reageren op deze changes door zichzelf manueel up te daten. Deze applicaties waren in het begin heel simpel, maar naarmate het team van Facebook meer en meer features toevoegde, werden ze meer en meer complex. Om bij te blijven met deze complexe applicaties, werden ook meer en meer mensen bij het team toegevoegd, met als resulaat dat de applicaties moeilijk te onderhouden werden. Ze kwamen erachter dat in deze complexe apps een kleine aanpassing kon gebeuren, die ervoor zorgde dat de app moest uitzoeken welke views aangepast moesten worden en moest deze dan ook veranderen. Binnen het team noemden ze dit `cascading updates`. Deze updates gebeuren heel traag, want de app moest eerst en vooral bijhouden wat aangepast moest worden en wat niet en daarna de views zichzelf laten updaten. Het werd moeilijk om te voorspellen wat zorgde voor deze cascading updates. (Occhino, 2015)

Idee en prototype

De code die beschrijft hoe de applicatie eruit ziet, bestaat al. Het idee was dat telkens de data veranderde, enkel die code opnieuw uitgevoerd zou worden. Dit lijkt een goede oplossing, maar het probleem is dat op deze manier opgeslagen state verloren gaat bij de update en dus voor enkele glitches kon zorgen. Daarnaast was er ook nog een relatief groot verlies in tijd en een grote toename in CPU-gebruik op de client. Om dit op te lossen bedacht Jordan Walke een prototype die dit proces efficiënter maakt en een deftige `user experience` garandeert. Dit markeerde de geboorte van React. (Occhino, 2015)

Open sourcing

Facebook.com maakte in 2012 volop gebruik van deze nieuwe technologie. Op dat moment werd het gebruikt om onder andere reacties te plaatsen op berichten, berichten leuk te

vinden[5] en gesprekken te voeren in de chat. Toen Facebook Instagram overnam, kwam de vraag vanuit het Instagram team om React te gebruiken. Aangezien Facebook dit enkel intern gebruikte, vonden ze dat het lastig zou zijn om zaken los te koppelen van hun infrastructuur. Instagram wou absoluut deze technologie gebruiken en dat zorgde bij Facebook voor het initiatief om de versie van React aan te passen en opnieuw te bouwen zodat React `open sourced` kon worden. Op die manier zou Instagram dan ook gebruik kunnen maken van React. (Occhino, 2013)

2.3.2 React architectuur en werking

Componenten

Elke React component is eigenlijk een JavaScript functie, die code retourneert. Deze code stelt de visuele representatie van de component voor. React gebruikt een taal genaamd JSX. De functies worden in een bepaalde volgorde aangeroepen, waarop het resultaat dan samengevoegd wordt en getoond wordt aan de gebruiker. Op deze manier wordt een pagina gebouwd met React. (Domes, 2017)

JSX

JSX is een uitbreiding op de syntax van JavaScript. Het lijkt sterk op HTML, maar bevat de kracht van JavaScript. JSX maakt React elementen aan die dan uiteindelijk gerenderd worden in het DOM. Afbeelding xx toont een voorbeeld van hoe de JSX syntax gebruikt kan worden. (**React2019**)

Elements

Elementen zijn de kleinste bouwstenen van React applicaties. Een element beschrijft wat op het scherm moet komen. React elementen zijn gewone objecten. Deze zijn heel simpel om aan te maken. (React, 2019)

[5] Toen bestond de optie nog niet om met emoticons berichten leuk te vinden.

3. Methodologie

3.1 title

4. Conclusie

A. Onderzoeksvoorstel

Het onderwerp van deze bachelorproef is gebaseerd op een onderzoeksvoorstel dat vooraf werd beoordeeld door de promotor. Dat voorstel is opgenomen in deze bijlage.

A.1 Introductie

JavaScript is een van de slechtste programmeertalen. De object prototypen zijn een primitieve en slordige manier om aan OO-programmeren te doen. Die zorgen ervoor dat het niet goed schaalt met grote applicaties. (Eng, 2016)

Technologieën zoals React of Angular hebben de laatste tijd wat interesse gewonnen. React wordt meer en meer gebruikt, maar Angular blijft toch wel populair binnen de business. In full-stack developer vacatures wordt Angular in 59% van de gevallen genoemd, React slechts in 37% (Schlothauer, 2019)

Dit werk heeft als bedoeling de keuze van technologie voor webapplicaties gemakkelijker te maken. Dit zal vooral voordelig zijn voor bedrijven binnen webdevelopment. De keuze van technologie kan bepalend zijn of een project al dan niet succesvol opgeleverd wordt. We stellen hier de vraag: "Welke eigenschappen van een project bepalen de aan te raden keuze van technologie voor dit project?". Ook zal bepaald worden bij welke eigenschappen een technologie als Angular of React beter is dan puur JavaScript.

A.2 State-of-the-art

JavaScript is altijd al een van de meest gebruikte programmeertalen geweest. JavaScript is een scripttaal die gebruikt wordt voor het interactief maken van een webpagina. De kenmerken van JavaScript worden omschreven als:

- Snel
- Gemakkelijk
- Krachtig

(Garbadge, 2018)

React is een JavaScript library die het voor de gebruiker gemakkelijk maakt om User Interfaces te maken. Het werd door Facebook ontwikkeld in 2011 met als doel de code voor grote webapplicaties gemakkelijker beheersbaar te maken. (Chand, 2019)

Angular is een herschrijving van het AngularJS framework ontwikkeld door Google. De eerste publieke versie (Angular2) werd vrijgegeven in september 2016. AngularJS begon populariteit te verliezen in het voordeel van de nieuwe Angular2. Op 28 mei 2019 werd versie 8 uitgerold. (Bodrov-Krukowski, 2018)

Er werden onderzoeken en online artikels gevonden die de verschillen tussen React en Angular benaderen, zoals Ari (2018). Ook werd al onderzoek gedaan naar het verschil met Vue.js, een JavaScript framework, maar er zijn geen onderzoeken gevonden over het verschil met pure JavaScript.

A.3 Methodologie

Eerst zullen er applicaties ontwikkeld worden met verschillende eigenschappen, zoals grootte, soorten data en complexiteit in JavaScript, Angular en React. Daarna zullen deze applicaties vergeleken worden op basis van bepaalde eigenschappen.

Snelheid van compilatie

De snelheid van compilatie is de eerste factor die onderzocht zal worden. De tijd zal gemeten worden en daarna wordt die per technologie vergeleken met elkaar.

Prestatie in runtime

Voor gebruikers van de webapplicatie is de prestatie in runtime heel belangrijk. Tegenwoordig zijn mensen gewoon van een snelle internetverbinding te hebben, dus moet een webapplicatie ook snel geladen worden. Er zal een backend opgesteld worden waaruit data opgevraagd zal worden. Deze backend is voor alle technologieën dezelfde. Er zal o.a. nagegaan worden hoe snel de data opgehaald kan worden.

Eenvoud

Om grote applicaties te onderhouden, is eenvoud een van de meest belangrijke aspecten. In dit onderzoek zal bepaald worden hoe gemakkelijk het is om binnen de gebruikte technologie een aanpassing of uitbreiding te maken. Ook zal er nagegaan worden hoeveel lijnen code nodig waren en welke handelingen uitgevoerd moesten worden om tot het eindresultaat te komen.

De eigenschappen van de webapplicaties zullen gemeten worden binnen een virtuele machine. De gebruikte software voor de verschillende applicaties in alle drie de technologieën zal Visual Studio Code van Microsoft zijn.

Op basis van de eigenschappen zal dan een overzicht opgesteld worden waarin de technologieën vergeleken zullen worden bij de verschillende applicaties.

A.4 Verwachte resultaten

Vermoedens zijn dat aan de hand van de resultaten Angular en React heel snel de voorkeur zullen krijgen op JavaScript. Vermoedelijk zullen Angular of React de code overzichtelijker maken, hoewel dat voor een langere compilatietijd zal zorgen. JavaScript zal echter wel nog enkele voordelen metzich meebrengen bij enkele soorten webapplicaties tegenover Angular of React.

A.5 Verwachte conclusies

De verwachtingen voor dit onderzoek zijn:

- JavaScript is sneller bij kleine en gemiddelde applicaties met weinig functionaliteit
- Voor webapplicaties met veel functionaliteit en verschillende soorten data zullen Angular en React overzichtelijker zijn.
- JavaScript is trager dan Angular of React bij applicaties met veel klassen of interfaces
- React zal de snelste zijn wanneer het komt op het aanpassen van het DOM.
- Bij grote en complexe applicaties zal JavaScript moeilijker te programmeren zijn
- De performance van JavaScript zal in elk van de gevallen beter zijn.

Bibliografie

Angular. (2019a). Architecture overview. Verkregen 20 mei 2019, van https://angular.io/guide/architecture

Angular. (2019b). Component. Verkregen 20 mei 2019, van https://angular.io/api/core/Component

Angular. (2019c). Dependency Injection in Angular. Verkregen 20 mei 2019, van https://angular.io/guide/dependency-injection

Angular. (2019d). Introduction to modules. Verkregen 20 mei 2019, van https://angular.io/guide/architecture-modules

Angular. (2019e). Introduction to services and dependency injection. Verkregen 20 mei 2019, van https://angular.io/guide/architecture-services

Ari, D. (2018, september 12). A comparison between angular and react and their core languages. Verkregen 3 april 2019, van https://medium.freecodecamp.org/a-comparison-between-angular-and-react-and-their-core-languages-9de52f485a76

Baer, E. (2018). *What is React and why it matters*. Sebastopol, California, United States: O'Reilly Media, Inc.

Bodrov-Krukowski, I. (2018, maart 22). Angular Introduction: What It Is, and Why You Should Use It. Verkregen 13 oktober 2019, van https://www.sitepoint.com/angular-introduction/

Chand, S. (2019, oktober 13). What Is React? – Unveil The Magic Of Interactive UI With React. Verkregen 13 oktober 2019, van https://www.edureka.co/blog/what-is-react/#3

Domes, S. (2017, november 13). Everything You Should Know About React: The Basics You Need to Start Building. Verkregen 5 april 2019, van https://medium.freecodecamp.org/everything-you-need-to-know-about-react-eaedf53238c4

Eng, R. K. (2016, januari 26). The top 10 things wrong with JavaScript. Verkregen van https://medium.com/javascript-non-grata/the-top-10-things-wrong-with-javascript-58f440d6b3d8

Fain, Y. (2016, april 26). Angular 2 and TypeScript - A High Level Overview. Verkregen 20 mei 2019, van https://www.infoq.com/articles/Angular2-Typescript-High-Level-Overview

Garbadge, M. J. (2018, december 30). Top 3 most popular programming languages in 2018 (and their annual salaries). Verkregen van https://hackernoon.com/top-3-most-popular-programming-languages-in-2018-and-their-annual-salaries-51b4a7354e06

Gilbert. (2018, december 6). Mithril.js: A Tutorial Introduction (Part 1). Verkregen 2 december 2019, van https://gilbert.ghost.io/mithril-js-tutorial-1/

Hamedani, M. (2018, november 5). React vs. Angular: The Complete Comparison. Verkregen 3 april 2019, van https://programmingwithmosh.com/react/react-vs-angular/

Lotanna, N. (2019, maart 6). New Angular features you didn't know existed. Verkregen 1 december 2019, van https://blog.logrocket.com/new-angular-features-you-didnt-know-existed-7f292a6e7afc/

Mithril. (2019a). Components. Verkregen 6 december 2019, van https://mithril.js.org/components.html

Mithril. (2019b). Credits. Verkregen 6 december 2019, van https://mithril.js.org/credits.html

Mithril. (2019c). Introduction. Verkregen 2 december 2019, van https://mithril.js.org/

Mithril. (2019d). Virtual DOM nodes. Verkregen 4 december 2019, van https://mithril.js.org/vnodes.html

Occhino, T. (2013). JS Apps at Facebook (JSConfUS 2013). Verkregen 22 mei 2019, van https://www.youtube.com/watch?v=GW0j4sNH2w

Occhino, T. (2015, januari 28). Introducing React Native (React.js Config 2015). Verkregen 21 mei 2019, van https://www.youtube.com/watch?v=KVZ-PZI6W4

React. (2019). Introducing JSX. Verkregen van https://reactjs.org/docs/introducing-jsx.html

Schlothauer, S. (2019, maart 12). React steadily grows, while Angular maintains enterprise hold. Verkregen 18 september 2019, van https://jaxenter.com/react-angular-enterprise-156725.html

Shan, P. (2015, november 21). What's new in Angular 2.0? Why it's rewritten - addressing few confusions. Verkregen 20 mei 2019, van http://voidcanvas.com/angular-2-introduction