Scriptie ingediend tot het behalen van de graad van
PROFESSIONELE BACHELOR IN DE ELEKTRONICA-ICT

Scriptietitel

naam student 1, naam student 2

academiejaar 2015-2016

AP Hogeschool Antwerpen
Wetenschap &Techniek
Elektronica-ICT

ARTESIS PLANTIJN
HOGESCHOOL ANTWERPEN

Inhoudsopgave

Titelblad 1.1
Woord Vooraf 1.2
Samenvatting 1.3
Inleiding 1.4
Project Management 1.5
- Agile Ontwikkeling 1.5.1
- Source Controle 1.5.2
- Projectstructuur 1.5.3
- Continue Integratie 1.5.4
- Continue Implementatie 1.5.5
- Persistence 1.5.6
- Lokale Ontwikkelingsomgeving 1.5.7

Ontwikkeling 1.6
- Gebruikservaring 1.6.1
- Authenticatie en Autorisatie 1.6.2
- Slack Integratie 1.6.3
- Beheerdersprofiel 1.6.4
- Medewerkers 1.6.5
- Technologieën 1.6.6
- Vragen en Antwoorden 1.6.7
- Heatmap 1.6.8

Testen 1.7
- Integratietesten 1.7.1
  - Integratietesten Client 1.7.1.1

Besluit 1.8
Bibliografie 1.9
Bijlages 1.10

Titelblad

Woord Vooraf

Samenvatting

Inleiding

Project Management

Agile Ontwikkeling

Source Controle

Projectstructuur

Client-side

Structuur

De client-side van het project is opgebouwd in Angular. Sinds oktober 2018 maakt het project gebruik van Angular versie 7.

De Angular CLI wordt gebruikt om de client en zijn componenten, services, ... te genereren. Dit zorgt ervoor dat de structuur van de client zich houdt aan de best practices, aangeraden door de officiële documentatie van Angular (Angular Docs, s.d.). De folder waarin het client-side project zich bevindt, is hernoemd naar *Involved.BrainChain.Client* om consequent te zijn met de naamgevingsregels van de backend projecten.

Om ervoor te zorgen dat de implementatie van features, zowel op korte termijn als op lange termijn, helder en overzichtelijk kan gebeuren, passen we in de client applicatie het volgende principe toe; Feature- , shared- en coremodules (Angular Docs, s.d.).

Featuremodules zijn modules gecreëerd voor een specifieke toepassing van de applicatie. Het is de bedoeling dat bestanden die gerelateerd zijn aan een zekere functionaliteit, in één folder terechtkomen. De totale inhoud van de folder wordt dan gehangen aan de featuremodule. Dit betekent concreet, dat de featuremodules afgestemd zijn op de businesslogica van onze applicatie. Bijvoorbeeld, alle client-side logica die een beheerder nodig heeft om medewerkers van het bedrijf te beheren, wordt gebundeld in de medewerkers featuremodule. Elke featuremodule heeft een routingmodule. Op deze manier heeft elke featuremodule een relatie met een specifieke link in de client applicatie. Featuremodules zijn lazy-loaded. Wanneer de gebruiker surft naar de link van een featuremodule Dit zorgt ervoor dat de modules enkel geladen worden wanneer ze beschikbaar moeten zijn voor de gebruiker. In dit project bevatten de featuremodules voornamelijk slimme- en domme componenten, en services.

De **sharedmodule** bevat delen van de applicatie die in meerdere featuremodules gebruikt worden. De sharedmodule bevat voornamelijk directives, pipes, componenten,.... Componenten die gebruikt worden in meer dan twee featuremodules, worden als vuistregel gedeclareerd in de sharedmodule. Deze componenten zijn altijd domme componenten. Services horen niet thuis in de sharedmodule. Dit voorkomt dat er meerdere instanties van een singleton service worden gecreëerd. De sharedmodule wordt geïmporteerd in de featuremodules die toegang nodig hebben tot de shared componenten, directives, enzovoorts.

De **coremodule** bevat alle services en componenten die de client moet laden bij het opstarten van de applicatie. In het algemeen zijn het meestal services die gedeclareerd worden in de coremodule. De coremodule wordt geïmporteerd in de root AppModule, en enkel daar.

Alle services in de client zijn singelton services. De coremodule zorgt ervoor dat de services geprovided worden in de root van de applicatie. Dit maakt dat de client meer *tree shakable*[1] is. Services die gebruikt worden in slechts één featuremodule, worden mee in de folder van die featuremodule geplaatst. Ook deze services worden geprovided in de root. De services die gegeneerd worden met behulp van Swagger, zijn voorbeelden van services die in meerdere featuremodules gebruikt kunnen worden.[2] Deze services horen dus in de folder van de coremodule.

De componenten die aanwezig zijn in de featuremodules, zijn opgedeeld in domme en slimme componenten. De relatie tussen de slimme en domme componenten is meestal een parent-child relatie. Slimme componenten staan in voor de werking van de client en het verwerken van data. Domme componenten tonen de data aan de gebruiker, en zorgt ervoor dat de gebruiker kan interageren met de data (Arnold, 2017). Bijvoorbeeld, data die verwerkt werd in de slimme component en getoond moet worden aan de gebruiker, wordt doorgespeeld naar een domme component. Het is dan de taak van die domme component om de data te tonen. Als een gebruiker interactie moet kunnen hebben met een domme component, wordt dit niet afgehandeld door de domme component. De domme component stuurt de interactie van de gebruiker naar zijn ouder, de slimme component. De slimme component zal dan de interactie van de gebruiker afhandelen. Als gevolg zullen services nooit geïnjecteerd worden in domme componenten, enkel in de slimme ouder. In de featuremodules wordt het aantal slimme componenten zo klein mogelijk gehouden. Concreet zullen de verschillende routes van de client altijd verwijzen naar de slimme componenten. De DOM-elementen die getoond moeten worden zijn dan opgebouwd met de HTML-selectors van de domme componenten.

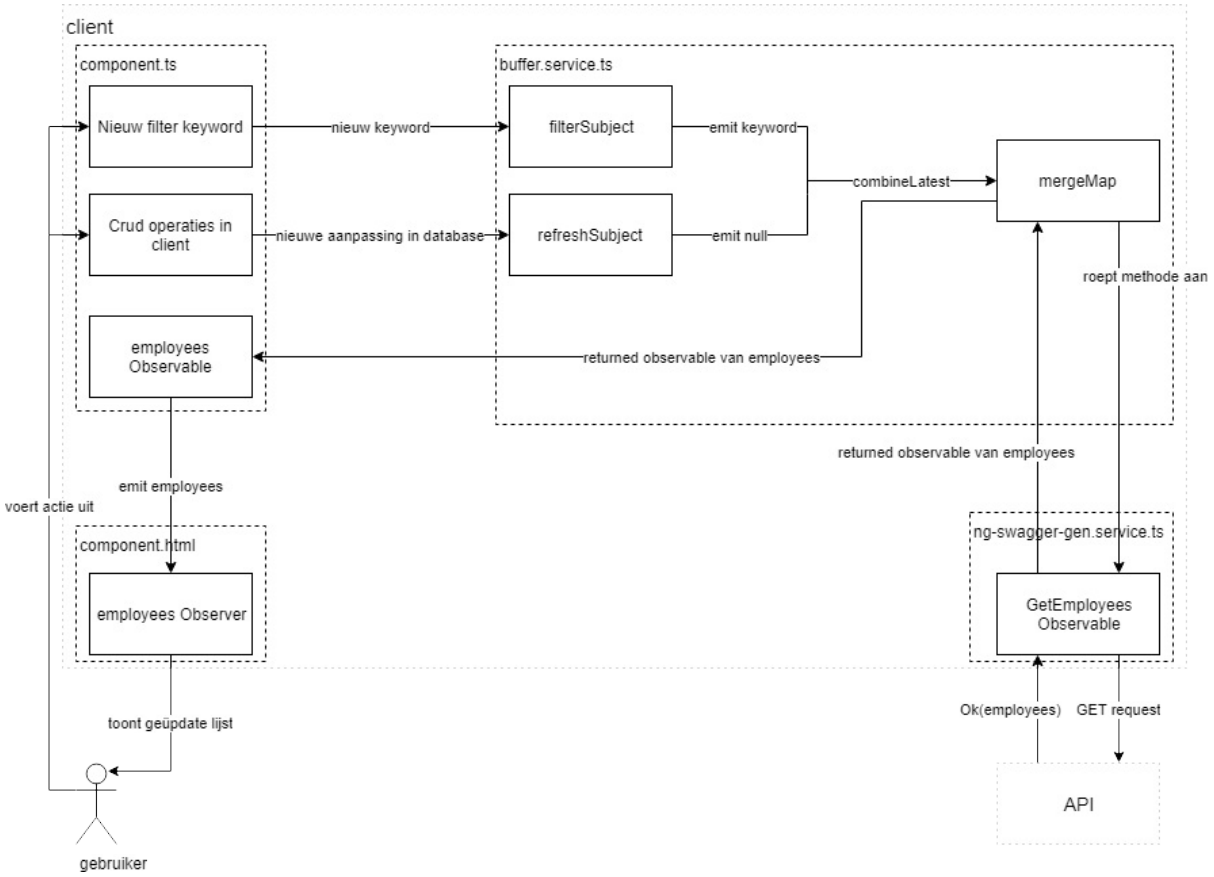
Programmeerconcepten

Reactive programming

Reactive programmeren is een ontwikkelingsmodel gestructureerd rond asynchrone datastreams. Reactive Extensions, ofte Rx, is de API die het meeste gebruikt wordt om reactive te programmeren. In de client wordt er gebruik gemaakt van de RxJS, de Rx library voor javascript.

Rx is opgebouwd rond het gedachtegoed van het Observable Pattern, het Iterator Pattern en Functional Programming. De belangrijkste van Observables en Observers. Observers kunnen zich abonneren (*subscriben*) op een Observable. Wanneer Observables datastreams de wereld insturen, zullen de geabonneerde Observers luisteren naar en reageren op de datastreams. De kracht van Rx zit in het feit dat de API de client toegang geeft tot verschillende Operators. Deze operators zorgen ervoor dat we datastreams kunnen transformeren, combineren, manipuleren, enzovoort… (Bhuvan, 2018).

Rx wordt in het project gebruikt om op een vlotte en asynchrone manier dataflow tussen de verschillende componenten, services, directives, enzovoort… te voorzien. De belangrijkste plek waar Rx gebruikt wordt, is bij de communicatie tussen de client en de backend. De web API maakt gebruik van Swashbuckle om een swagger file te generen. In de client wordt er gebruik gemaakt van de ng-swagger-gen library om services aan te maken die de effectieve REST calls naar de web API gaan uitvoeren. Alle methodes in de gegenereerde services retourneren Observables. De data die de client toont moet altijd de meest recente zijn die in de databank aanwezig is. Daarom steekt er een service als buffer tussen de slimme component die de data nodig heeft, en de service die de calls naar de web API maakt. De werking van zo een bufferservice wordt in de volgende paragraaf uitgelegd, met een voorbeeld uit dit project.



Op het beheerscherm voor de medewerkers kan de beheerder een lijst van medewerkers raadplegen en de details van elke medewerker te zien krijgen. Op deze lijst van medewerkers kunnen er CRUD-operaties worden uitgevoerd. De buffer service heeft een Observable property genaamd employees. Employees heeft als type een lijst van medewerkers. Employees wordt gecreëerd door combineLatest, een van de methodes van Rx. CombineLatest verwacht als parameters meerdere Observables, in dit geval refreshSubject en filterSubject. Subjects zijn objecten die zowel Observable als Observer zijn. Zodra een van de twee Subjects een nieuwe waarde emit, zal employees door combineLatest zijn ook zijn waarde *emitten* uitzenden. De Rx operator, mergeMap, wordt toegepast op combineLatest om ervoor te zorgen dat de waarde die employees zal *emitten* de lijst van medewerkers is. MergeMap spreekt de methode uit de gegenereerde service aan, die de GET-request naar de web API doet voor de lijst van medewerkers. MergeMap zorgt er ook voor dat employees de Observable, die geretourneerd wordt door de GET methode van de gegenereerde service, wordt toegewezen. Wanneer de client een CRUD-operatie wil uitvoeren op de lijst van medewerkers, wordt ervoor gezorgd dat refreshSubject een waarde uitzendt (deze waarde mag altijd null zijn). Aangezien refreshSubject een nieuwe waarde emit, wordt de functie van mergeMap terug opgeroepen. Zo bevat employees altijd de meeste recente lijst van medewerkers.

Continue Integratie

Azure pipelines yml files Continue integratie en continue deployment Azure pipelines is een tool dat gebruikt wordt om te helpen bij continue integratie en continue deployment. Voor continue integratie maken we gebruik van de build pipeline. De build pipeline detecteert wanneer er veranderingen zijn aangebracht op de branches van onze repository. De configuratie van de build pipeline wordt beschreven in yml files. De yml file die zich bevindt in de root folder van de repository vertelt de build pipeline wanneer de build pipeline getriggerd moet worden. Het project heeft dus één build pipeline die automatisch wordt uitgevoerd zodra er een commit wordt gepusht naar een van de branches, die beschreven staat in de yml file, in de repository. Ook bevat deze een

verwijzing naar andere yml files die dan weer beschrijven welke projecten moeten worden gebuild en getest. De bestanden, die gegenereerd worden door het buildproces, worden gekopieerd naar een zogenaamd artifact. Deze artifacts worden gebruikt door de release pipeline. In de build pipeline worden ook de unittests automatisch uitgevoerd. Wanneer zowel het builden als de test geslaagd zijn, gaat de release pipeline van start. De release pipeline staat in voor de continue deployment van de applicatie. Het project heeft twee release pipelines. Artifacts die gegenereerd worden door alle branches, buiten de master branch, triggeren de eerste pipeline. Artifacts die gegeneerd worde door de master branch triggeren de tweede pipeline. Aan een release pipeline kunnen variabelen worden vastgehangen. Omdat deze variabelen niet in de source controle terechtkomen, kunnen de release variabelen gebruikt worden om gevoelige informatie te bewaren. In het geval van het Angular project zorgen we ervoor dat de environment variabelen, die applicatie gebruikt in de productie omgeving, vervangen worden door de juiste release variabelen. Dit gebeurt met behulp van een powershell script. In het geval van .NET Core, kan de release pipeline automatisch de variabelen die zich bevinden in de appsettings.json files vervangen. In dat geval moet de naam van de release variabele voldoen aan de correcte benaming. De eerste release pipeline zorgt ervoor dat de applicatie automatisch gedeployed wordt naar de test servers. De tweede release pipeline zorgt ervoor dat de applicatie automatisch gedeployed wordt naar de acceptatie server. De tweede release pipeline kan de applicatie ook deployen naar de productieserver. Dit moet echter manueel getriggerd worden. Beide release pipelines zorgen ervoor dat de migraties die nodig zijn, worden uitgevoerd op de respectievelijke databanken. Nadat de applicatie gedeployed is op de testservers, worden er clientside integratietesten uitgevoerd

Continue Implementatie

Persistence

Fluent migrations De migraties van de databank worden uitgevoerd met behulp van Fluent Migrator. Fluent Migrator is een framework ontwikkeld voor .NET en .NET Core. Het staat toe om, op een gestructureerde en heldere manier, database schema’s aan te passen en te creëren. De migraties worden beschreven in klasses, geschreven in C#. Er zijn twee redenen waarom de voorkeur uit ging naar Fluent Migrator boven de Code First Migrations van Entity Framework zelf. In het project is er een .NET Core console applicatie aanwezig dat ervoor zorg dat de migraties worden uitgevoerd. De console applicatie aanvaard een connectionstring van een databank als argument. De console applicatie wordt ook gebruikt om de lokale databank en de databank op de testserver, te seeden met testdata. Fluent Migrator laat het toe om via de runner SQL-query’s uit te voeren op de databank. In de build pipeline wordt de console applicatie gebuild. De DLL-bestanden die het resultaat zijn van de build worden gekopieerd naar het build artifact. In de release pipeline worden de connectionstrings van de databanken opgeslagen. De migraties worden in de release planning als eerste stap uitgevoerd. Via een command line script in de wordt de console applicatie uitgevoerd, met de juist connectionstring verworven uit de release pipeline.

Lokale Ontwikkelingsomgeving

Local scripts Nadat wij gestopt zijn met werken aan de applicatie, wil Involved dat er nog verder ontwikkeld kan worden aan de applicatie. Daarom zijn er, op vraag van de opdrachtgever, enkele scripts gecreëerd die helpen bij het opzetten van de lokale ontwikkelingsomgeving. Iemand die de repository clonet, kan het bash script: build.sh uitvoeren. Dit script zorgt ervoor dat dat de API, scheduler en angular projecten worden gebuild met dotnet en npm. Er wordt een database gecreëerd op de lokale SQL-server en de console applicatie die de migraties van de database afhandelt wordt gebuild en gerund.

[1]. *Tree shaking* is een term die, in de context van JavaScript, gebruikt wordt om het elimineren van dode

code te beschrijven. Ongebruikte modules worden tijdens het buildproces niet gebundeld door *tree shaking* (Bachuk, 2017). ↩

2. De services die gegeneerd worden door Swagger zorgen ervoor dat de client calls kan doen naar de backend. ↩

Ontwikkeling

Authenticatie en Autorisatie

Autorisatie en authenticatie met auth0 Authenticatie en autorisatie in onze applicatie wordt afgehandeld door auth0. Auth0 is een management platform voor authenticatie en autorisatie. Via het online platform wordt de autorisatie van de applicatie en de gebruikers geconfigureerd. Vanuit de applicatie zelf kan de configuratie aangepast worden met behulp van de management API van auth0. Gebruikers worden geauthenticeerd met de universele login van auth0. Wanneer gebruikers een authenticatie request triggeren worden ze doorverwezen naar de universele login pagina van auth0. Deze pagina wordt enkel gebruikt voor inloggen en authenticatie, en toont het lock widget van auth0. Andere logica is niet beschikbaar op deze pagina. De hoofdreden waarom er gebruik gemaakt wordt van universele login, is veiligheid. De universele login zorgt ervoor dat de applicatie geen gebruik maakt van cross-origin authenticatie. Cross-origin authenticatie is in se gevaarlijker en heeft een grotere kans om het slachtoffer te zijn van, onder andere, een man-in-the-middle aanval. Wanneer de authenticatie geslaagd is, wordt het access token en de datum en tijd wanneer het token vervalt, ontvangen van auth0, opgeslagen in de browser cache via de localStorage tool van Angular. Deze tokens vervallen na tien uur. De routes in de client side van de applicatie worden beschermd door de authGuard service. De service kijkt of er zich een accestoken bevindt in de local storage, en of dit token nog niet vervallen is. Zo ja, wordt de gebruiker weer doorverwezen naar de login pagina. Aangezien er maar één soort gebruiker toegang heeft tot de client side van de applicatie, namelijk de beheerder, heeft een geauthenticeerde gebruiker toegang tot alle pagina’s van de website. In de backend van de applicatie worden alle controllers, buiten de Slack controller, in de API beschermd door autorisatie. Er kunnen dus geen ongeautoriseerde calls gemaakt worden vanuit de client. Calls die worden gemaakt vanuit de frontend worden in de client onderschept door een HttpInterceptor. Deze service zorgt ervoor dat het accestoken als bearer token wordt meegegeven aan de header van de call, alvorens ze naar de backend worden gestuurd. De backend checkt dan met behulp van de authorization API van auth0 of de gebruiker deze call mag maken. In de huidige iteratie van de applicatie, heeft een geauthenticeerde gebruiker toegang tot alle methodes van elke controller, buiten de Slack controller.

Slack Integratie

Slack integration Om data te verkrijgen van medewerkers van het bedrijf hebben we een Slack applicatie gebouwd. Slack laat zijn gebruikers toe om applicatie toe te voegen aan de workspace. Zo kan men de functionaliteit van een workspace uitbreiden en veranderen. In de Slack applicatie is er een bot user gebundeld. Deze bot zorgt ervoor dat medewerkers via conversaties kunnen interageren met de Slack app. De Slack maakt gebruik van twee Slack API’s om interacties met medewerkers af te handelen, de Events API en de Conversations API. Events API De Events API zorgt ervoor dat een Slack app zich kan abonneren op events die gebeuren in Slack. In de Slack app geven we een request URL mee. Om te verifiëren of deze URL geldig is stuurt Slack POST request met als type: url_verification en een willekeurige challenge string. Om de verificatie te voltooien moet de web API met een OK-response antwoorden, met een response body die de challenge string bevat. Wanneer er een event gebeurt waarop we geabonneerd zijn, stuurt Slack een POST request naar de URL. De requests zijn JSON requests. De request URL komt overeen met een methode in de Slack controller van de web API. Slack verwacht dat er binnen de drie seconden gereageerd wordt met een 2XX response code. Lukt dit niet, zal de Events API maximaal drie keer proberen om de request naar de request-URL te sturen. Het is dus belangrijk dat de web API zo snel mogelijk antwoord op een request. Hierom moet er zoveel mogelijk vermeden worden dat het verwerken van een event en het beantwoorden op een event in hetzelfde proces gebeuren. De

Slack app kan zich op twee verschillende soorten events abonneren., Workspace Events en Bot Events. Workspace Events zijn alle soorten events die kunnen plaatsnemen in een Slack Workspace. Deze events zijn echter niet relevant voor deze applicatie. Bot Events zijn alle events die gerelateerd zijn aan kanalen en conversaties waar de bot deel van uitmaakt. Het enige event waarop de Slack app geabonneerd is, is het Bot Event: message.im. Dit event ontstaat wanneer er een bericht wordt gestuurd in een direct message kanaal, waar de bot deel van uitmaakt. De web API checkt het type van de request die gestuurd is naar de request URL, om te weten of de request al dan niet verwerkt moet worden. De API is enkel in staat om requests met als type: url_verification (zoals hierboven geschreven wordt dit gebruikt om de request URL te verifiëren), en requests die een event hebben met als channel type: im (dit toont aan dat het event afkomstig is van een direct message channel), te verwerken. Als de controller een request ontvangt dat niet verwerkt kan worden, zal de methode een exception throwen. Conversations API

Beheerdersprofiel

Choose language Bij het injecteren van de translate service in de Appcomponent geven we mee voor welke talen we een json file hebben aangemaakt. Als standaardtaal gebruikt de client Engels. De client checkt of de taalcode in de browser overeenkomt met een van de talen die beschikbaar zijn (bijvoorbeeld: nl voor Nederlands). Zo ja wordt die taal gebruikt voor de lokalisatie, zo nee wordt Engels gebruikt. De translate service heeft een methode om de lijst van beschikbare talen te verkrijgen. Deze lijst wordt in de header weergegeven, zodat een gebruiker op die manier de taal kan aanpassen.

Medewerkers

CSV Upload De gebruiker van de client applicatie moet op een gemakkelijke manier alle medewerkers van zijn bedrijf kunnen toevoegen. Daarom is er een optie voorzien in het beheerscherm van de medewerkers om een CSV-bestand te uploaden. Dit bestand wordt met een request in een POST request doorgestuurd naar de web API. In de backend wordt het bestand met behulp van de .NET library CsvHelper omgezet naar een lijst van medewerkers. CsvHelper heeft een methode om data in een CSV-bestand lijn per lijn uit te lezen en te mappen naar een DTO, onder voorwaarde dat de titel van elke kolom gelijk is aan de property's van de DTO. Is dit niet het geval, wordt er een foutmelding gegenereerd dat de kolommen in het bestand niet voldoen aan de verwachte property's. De lijst van DTO's gegenereerd door de service wordt vervolgens gevalideerd. Ieder item in de lijst wordt gecontroleerd of het voldoet aan de voorwaarden die nodig zijn om een medewerker aan te maken (bijvoorbeeld de naam mag geen lege string zijn). Als er zich ergens in het bestand foute data bevindt, zal de gebruiker een foutmelding krijgen met het adres van de foute data (rijnummer en kolomnaam). Slechts als de lijst gevalideerd is zal er een request worden gemaakt om een lijst van medewerkers te creëren in de databank.

Filter De nieuwe filter moet ervoor zorgen dat er gefilterd kan worden op De filter is een domme component, die bestaat uit een enkel tekstveld. Wanneer de tekst in het tekstveld verandert, wordt het input event afgevuurd. De component bevat een subject, een speciaal type van observable. De waarde van het subject wordt de waarde van het tekstveld, telkens wanneer het input event wordt afgevuurd. De filtercomponent heeft als output een observable die de waarde van het subject publiceert, zolang deze waarde een halve seconde onveranderd is gebleven én de waarde verschillend is van de vorige gepubliceerde waarde. Dit zorgt ervoor dat, als een gebruiker in het tekstveld aan het typen is, de waarde niet verandert met elke toetsaanslag. Zo wordt er enkel een request voor gefilterde data gestuurd wanneer de gebruiker even niet meer getypt heeft. Een bovenliggende slimme component luistert naar het outputevent van de filtercomponent. De waarde die gebruiker intypte in het tekstveld is het keyword dat als parameter wordt gestuurd naar het filter subject in de dataservice. Aangezien een subject een soort observable is, kunnen we een functie creëren die een call doet naar de backend wanneer de waarde van het subject geüpdatet wordt.

Technologieën

Technologies module De client heeft een pagina voor het beheer van de technologieën en vaardigheden die de medewerkers kunnen bezitten. De pagina heeft een overzicht met alle technologieën die de beheerder relevant vindt voor het bedrijf. Vanuit deze pagina kan de beheerder nieuwe technologieën toevoegen, oude technologieën updaten en irrelevante technologieën verwijderen. Per technologie kan de beheerder bekijken welke werknemers er vaardig zijn in de technologie.

Vragen en Antwoorden

Open question qna rounds De Slackbot is in staat om open vragen te stellen via een direct message aan alle medewerkers. Dankzij de Events API kunnen we ervoor zorgen dat Slack de antwoorden doorstuurt naar de backend van de applicatie. Dit veroorzaakt een echter een probleem: Slack zal een request sturen naar de backend, elke keer iemand een direct message stuurt naar de bot. Er moet dus voor gezorgd worden de backend enkel de direct messages van de medewerkers verwerkt wanneer er effectief een vraag is gesteld door de bot. Ook moet ervoor gezorgd worden dat de bot reageert op een direct message van een medewerker wanneer er geen vraag gesteld is of wanneer de medewerker al een antwoord heeft gegeven. Het is beter voor de gebruikservaring dat de bot elke input van een medewerker erkent. Zo kan er geen verwarring ontstaan over het feit of de bot al dan niet beschikbaar is, en waartoe de bot in staat is. Als oplossing voor dit probleem is er een ronde systeem geïntroduceerd in de applicatie. QnaRound is een entiteit die informatie bevat over het onderwerp van de vraag die moet gesteld worden tijdens de ronde, de vraag die effectief gesteld is tijdens de ronde en of de ronde al dan niet nog actief is. De scheduler is verantwoordelijk voor het starten van een nieuwe ronde. Wanneer het tijd is om de medewerkers te bevragen over een nieuw onderwerp, stuurt de scheduler een request naar de business logica om een nieuwe ronde te creëren. Een ronde moet een onderwerp meekrijgen. Dit onderwerp kan bijvoorbeeld de werkplaats of een vaardigheid zijn. Aan de hand van het onderwerp van de ronde wordt dan bepaald welke vraag er overeenkomt met dat onderwerp. Bij het creëren van een nieuwe ronde, wordt de nieuwe ronde als actief beschouwd en wordt de vorige ronde op non-actief gezet. Zo zal er altijd slechts één ronde actief zijn. Zodra het creëren van een nieuwe ronde geslaagd is, stuurt de scheduler een request om de ronde-vraag te stellen. De API heeft nu een manier om te bepalen hoe een direct message event moet worden afgehandeld. De flow ziet eruit als volgt. De Slack Events API stuurt een request naar de Slack controller van de web API. De web API controleert het type van de request (de twee verwachte types zijn url_verification en event_callback). Als het gaat om een direct message request, wordt er een service aangesproken die de direct message kan afhandelen. De web API stuurt twee requests naar de business logica. Een om informatie te verschaffen over de actieve ronde, en een tweede om te bepalen of de medewerker die de request getriggerd heeft, al dan niet geantwoord heeft op de vraag van de huidige ronde. Als de medewerker al heeft geantwoord heeft, of er geen ronde actief is, wordt er via de bot een bericht gestuurd naar de medewerker dat er op dit moment geen vragen zijn die beantwoord kunnen worden. Wanneer er wel een ronde actief is, én de medewerker nog geen antwoord heeft gegeven op de vraag van die ronde, wordt de request afgehandeld naargelang het onderwerp van de ronde.

Heatmap

Heatmap De heatmap is een visuele weergave van het aantal vaardigheden die de medewerkers van een bedrijf hebben. De beheerder moet in een oogopslag kunnen zien met welke vaardigheden de medewerkers overweg kunnen. De heatmap wordt daarom weergegeven als een bubble chart. Elke cirkel komt overeen met een specifieke vaardigheid. Hoe groter de cirkel, hoe meer medewerkers aangeven dat ze ervaring hebben met die vaardigheid. De cirkels krijgen als titel de naam van de vaardigheid mee, en als ondertitel het aantal medewerkers die bekwaam zijn in die vaardigheid. Om ervoor te zorgen dat de cirkels op een intuïtieve manier te

vergelijken zijn, is er wiskunde nodig. Wanneer de ene vaardigheid dubbel zoveel medewerkers bevat als de andere, moet de oppervlakte van de ene cirkel dubbel zo groot zijn als de andere. Als de straal of de diameter van de ene cirkel dubbel zo groot zou zijn als de andere, merkt een menselijk oog gauw op dat de kleine cirkel niet juist twee keer in de grote past. Als er een één op één relatie zou zijn tussen de straal en het aantal vaardigheden, zou, bij een verdubbeling in vaardigheden, de cirkel vier keer zo groot lijken. De relatie tussen de vaardigheden en de medewerkers is een veel op veel relatie. In de backend kan er met een eenvoudige query het aantal medewerkers per vaardigheid worden opgehaald. De client ontvangt de naam en het aantal in een DTO. De opbouw van de heatmap gebeurt met behulp van de d3.js library. D3.js is een javascript library die documenten manipuleert op basis van data. We creëren een service, waarin d3 de heatmap gaat opbouwen. De service bouwt met de ontvangen data SVG-elementen op; voor elke vaardigheid een. Het DOM-element waar de heatmap moet in getoond worden krijgt als id: “charts” mee. Wanneer de we de methode van de heatmap service oproepen in een component, vult d3 het DOM-element met de juiste id op met de gecreëerde SVG-cirkels. De cirkels zijn gesorteerd op willekeurige volgorde en krijgen een willekeurige kleur.

Testen

Integratietesten

Integratie testen zijn testen die worden uitgevoerd om de samenwerking tussen verschillende modules te testen. Verschillende modules van de software worden samengenomen en in hun geheel getest. In het test proces worden deze testen uitgevoerd na de unit tests.

Integratietesten client

Client side worden er aan integratie tests gedaan om te verzekeren dat de functionaliteit van het softwaresysteem voldoet aan de vereisten

Client side testen besparen tijd aangezien de functionaliteit van het systeem niet meer handmatig moeten worden getest op de test server Client side integratie testen worden in de release planning uitgevoerd na het deployen van de software op de test server. Er wordt onder andere getest of werknemers die worden toegevoegd effectief verschijnen op de pagina. Het integratietest project draait op Node.js en is volledig in typescript geschreven. Client side integratie tests worden uitgevoerd met behulp van twee library's, jest en puppeteer. Jest is een open-source library ontwikkeld door werknemers van Facebook. Het framework wordt gebruikt om javascript te testen en ondersteund ook typescript. Puppeteer is een library die controle geeft aan google chrome en chromium via de Devtools Protocol. Puppeteer kan zowat alle handeling uitvoeren die iemand handmatig in een browser kan doen. Het is ontwikkeld door ontwikkelaars van google chrome. Met puppeteer kunnen google chrome en chromium "headless" worden gestart. Dit betekent dat er een browser wordt geopend zonder grafische gebruikersinterface. De testomgeving wordt als volgt opgezet. In jest zijn er drie configuratieopties waar gebruik van wordt gemaakt in ons project. GlobalSetup, globalTeardown en testEnvironment zorgen ervoor dat jest gebruik maakt van custom modules, zodat er vlot gebruik gemaakt kan worden van puppeteer voor het effectieve testen. De setup module exporteert een asynchrone functie die eenmalig wordt uitgevoerd voor alle testen. Hierin wordt er een een globale instantie van de headless browser gecreëerd met door puppeteer, er wordt ingelogd in de client side van de applicatie. Puppeteer creëert bij de opstart van de globale browser een websocket endpoint waar de browser met verbonden wordt. De verbindingsdetails van het websocket endpoint worden tijdelijk naar een lokaal bestand geschreven. De environment module exporteert een klasse. Deze klasse creëert een omgeving waarin de testen worden uitgevoerd. Puppeteer zorgt ervoor dat een nieuwe browserinstantie wordt aangemaakt voor de testomgeving, die verbinding maakt met het websocket endpoint, opgezet in globalSetup. De testen draaien dus in de browserinstantie aangemaakt in testEnvironment. De teardown module exporteert dan weer een asynchrone functie die eenmalig wordt uitgevoerd nadat alle testen zijn afgelopen, ofschoon deze geslaagd zijn of niet. De teardown module sluit de globale browser die werd aangemaakt in de setup module. Hierdoor sluit ook het websocket endpoint. Tussen enkele stappen van het testen door, worden er screenshots genomen van de browser. Deze screenshots worden geüpload naar een virtuele opslagruimte in de azure cloudomgeving. Op die manier kunnen de verschillende stappen worden opgevolgd en gecontroleerd wanneer de testen falen.

Besluit

Bibliografie

- Angular Docs. *NgModule FAQs. What kinds of modules should I have and how should I use them?*. Opgevraagd op 3 januari 2019 van https://angular.io/guide/ngmodule-faq#what-kinds-of-modules-should-i-have-and-how-should-i-use-them
- Angular Docs. *Style Guide*. Opgevraagd op 14 januari 2019 van https://angular.io/guide/styleguide
- Arnold, J. (2017). *Dumb Components and Smart Components*. Opgevraagd op 4 januari 2019 van https://medium.com/@thejasonfile/dumb-components-and-smart-components-e7b33a698d43
- Bachuk, A. (2017). *What is tree shaking?*. Opgevraagd op 3 januari 2019 van https://medium.com/@netxm/what-is-tree-shaking-de7c6be5cadd
- Bhuvan, M. (2018). *An introduction to functional Reactive programming in Redux*. Opgevraagd op 6 januari 2019 van https://medium.freecodecamp.org/an-introduction-to-functional-reactive-programming-in-redux-b0c14d097836

Bijlages