Professionele Bachelor Toegepaste Informatica

JIDOKA.
face the future

RECRUITMENT-TOOL

Toon Froeyen

Promotoren:

Paula Kozanecka
Bart Clijsner

JIDOKA
Hogeschool PXL Hasselt

Bachelorpaper Academiejaar 2017-2018

Professionele Bachelor Toegepaste Informatica

JIDOKA.
face the future

RECRUITMENT-TOOL

Toon Froeyen

Promotoren:

Paula Kozanecka
JIDOKA
Bart Clijsner
Hogeschool PXL Hasselt

Bachelorpaper Academiejaar 2017-2018

Dankwoord

Ik zou graag mijn collega's van mijn stagebedrijf JIDOKA bedanken voor de fijne samenwerking. Jullie waren een geweldige steun en waren altijd bereid om te helpen. In het bijzonder wil ik dhr. Bjorn Monnens bedanken om mij de kans te geven deze stageopdracht bij JIDOKA te doen. Ook wil ik dhr. Piet Vandeput en dhr. Jan Eerdekens bedanken voor het reviewen van de code.

Verder zou ik mijn hogeschoolpromotor Bart Clijsner en bedrijfspromotor Paula Kozanecka willen bedanken voor de opvolging en de feedback. Ook zou ik nog even Heidi Janssen, Lennard Snoeks en Davy Beyls willen bedanken voor de spellingscontrole.

Bedankt allemaal!

Abstract

Deze stage bestaat uit een project en een onderzoek.

Het project gaat over het vergemakkelijken van het recruitment-proces van JIDOKA. Dit is gebeurd in twee stappen. De eerste stap was het maken van een Spring Boot RESTful Web API om gemakkelijk de postgres-database te kunnen aanspreken en veilig te werk te kunnen gaan. De tweede stap was het maken van een webapplicatie gemaakt in Angular4. Dit is een Typescript framework om makkelijk een SPA te maken. In deze applicatie moest er een connectie gelegd worden zodat je via de frontend data kon ophalen en aanpassen.

JIDOKA is een bedrijf dat vooral webapplicaties maakt in React, een JavaScript framework. Voor deze stage werd bewust gekozen voor Angular4. In het onderzoek wordt gekeken naar beide frameworks op het vlak van setup, leercurve, performantie,… om tot een conclusie te komen welk framework voor JIDOKA het meest aangewezen is om te gebruiken.

Inhoudsopgave

Lijst van gebruikte figuren

Lijst van gebruikte tabellen

Lijst van gebruikte afkortingen

CV	Curriculum Vitae
KPI	Kritieke Prestatie-Indicatoren
HTML	Hyper Text Markup Language
CSS	Cascading Stylesheets
API	Application Programming Interface
TDD	Test Driven Development
NPM	Node Package Manager
CRUD	Create Read Update Delete
XSS	Cross Site Scripting
SQL	Structured Query Language
DTO	Data Transfer Object
NG	A**NG**ULAR
JSX	Java Serialization to XML
CLI	Command Line Interface
MVC	Model View Controller
DOM	Document Object Model
MIT	Massachusetts Institute of Technology

Tabel 1 Lijst van gebruikte afkortingen

Inleiding

Het rekruteren van nieuwe werknemers gebeurt vandaag de dag in veel bedrijven via het bijhouden van data in bestanden. Dit kan onoverzichtelijk worden en veel tijd in beslag nemen.

Tijdens de stage was het mijn opdracht om het recruitmentproces van JIDOKA overzichtelijker en minder tijdrovend te maken door het gehele proces in een webapplicatie te steken. De aanpak hiervoor was dat vacatures in het systeem worden vastgelegd en kandidaten hiervoor een kandidatuur kunnen indienen. De frontend is gemaakt in Anuglar4/5 en de backend in Spring Boot. Samen communiceren ze met een Postgres database waarin alle data wordt bijgehouden.

JIDOKA werkt vooral met React. Omdat deze stage in Angular4/5 moest gemaakt worden, heb ik verschillende aspecten van Angular en React vergeleken om te zien welk van de twee het meest aangewezen is voor JIDOKA.

Dit eindwerk behandelt alle hierboven vermelde elementen van mijn stage bij JIDOKA.

1 Bedrijfsvoorstelling

1.1 JIDOKA

Bij JIDOKA ligt de focus op end-to-end IT-projecten waarbij ze zich vooral richten op vier fases: business *envisioning*, *inception*, *construction* en *transition*. De uitdaging van JIDOKA is om lange termijnoplossingen te bieden voor al hun IT-projecten.

JIDOKA was opgericht met één doel in gedachten: succesvolle IT-projecten afleveren, VOOR mensen, DOOR mensen. Ze willen hierbij zeggen dat mensenkapitaal hun grootste vermogen is en ook de belangrijkste reden voor unieke klantenervaringen.

2 Stageopdracht

2.1 Het probleem

JIDOKA is een bedrijf dat snel wil groeien. Als er veel kandidaturen binnenkomen kan dit veel papierwerk leveren en bijgevolg veel tijd in beslag nemen. Hierdoor is een automatisatie van dit proces ten eerste overzichtelijker, omdat alles direct op de webapplicatie staat, en ten tweede ook veel minder tijdrovend, waardoor er op andere belangrijke dingen gefocust kan worden.

Tot nu werd het hele recruitment proces bijgehouden op spreadsheets op Google Drive. Dit is zoals eerder al vermeld niet overzichtelijk. Om dit efficiënter te maken gaan we dit gehele proces in een webapplicatie omzetten.

2.2 Doelstelling

Ik zal een link moeten leggen met de JIDOKA careerspagina en zorgen dat via de applicatie de recruitmentflow optimaal gevolgd kan worden. Kandidaten moeten kunnen solliciteren via de website met CV, LinkedIn profiel en een aantal standaard in te vullen velden. Hiernaast moeten CV's en feedback per gesprek bijgehouden kunnen worden.

Er wordt ook verwacht dat er een zoekfunctie met filters ingebouwd wordt. Hierbij zullen ook KPI's bijgehouden worden zoals het aantal sollicitanten per vacature.

2.3 Omgeving

2.3.1 PostgreSQL

De database die alle data bevat doorheen het project is toegankelijk met SQL: een query taal die gebruikt wordt voor de toegang tot en het aanpassen van informatie in een database. Om een visualisatie te verkrijgen van deze database wordt de databasetool van IntelliJ gebruikt. Verder wordt ook Flyway toegepast om te zorgen dat de databasetabellen overeen komen met de klassen van de backend.

2.3.2 Spring Boot

JIDOKA gebruikt voor de backend technologie vooral Spring Boot. Dit is een Java framework dat IOC of *dependency injection* toepast om tot goed overzichtelijke applicaties te komen. Spring boot zorgt ervoor dat er een webserver wordt opgestart waar verzoeken op kunnen gebeuren via verschillende paden in een URL.

2.3.3 Angular4/5

Angular is het framework dat gebruikt zal worden om de frontend te maken. Het is een op TypeScript gebaseerd framework dat gebruik maakt van *dependency injection* en het binden van objecten om zo tot een gestructureerde en overzichtelijke manier van werken te gaan.

2.3.4 TypeScript, HTML en CSS

Kennis van TypeScript is nodig om Angular te gebruiken. Dit is een sterk getypeerde taal en ook een superset van JavaScript. Verder gebruikt Angular ook HyperText Markup Language (HTML, een taal om webpagina's te ontwikkelen) en Cascading Style Sheet (CSS), style sheets om de lay-out van de webpagina's te formatteren).

2.3.5 React

React is het framework waarmee Angular zal vergeleken worden in het onderzoek. Dit is een op JavaScript gebaseerd frontend framework dat developers veel flexibiliteit van libraries toelaat op de manier van ontwikkelen.

2.3.6 IntelliJ IDEA/WebStorm

IntelliJ IDEA en WebStorm zijn de IDE's waarin gecodeerd zal worden. Ze zijn beide gemaakt door het softwarebedrijf JetBrains. Dit is een bedrijf waarvan de tools gericht zijn op softwareontwikkelaars en projectmanagers. WebStorm zal worden gebruikt voor de frontend en IntelliJ IDEA voor de backend.

2.3.7 Docker

Docker is een softwaretechnologie die containers levert. Deze containers bieden een extra laag van abstractie en automatisering. Docker is in de stage gebruikt om makkelijk een PostgreSQL-database op te zetten.

2.4 Implementatie van de stageopdracht

De stageopdracht duurde 12 weken waarbij er een tijdsplanning was opgesteld voor deze periode. Onderstaande tabel is een schatting van wat er per week verwacht werd. Omdat we op een agile manier te werk gingen, veranderden de vereisten continu en konden we deze planning niet strikt volgen.

Omschrijving	Startdatum	Geschatte dagen
Stageweek 1 2017-10-02		
Introductie tot JIDOKA + Toelichting technische aspecten opdracht	2017-10-02	1
Set-up van de opdracht: Opzetten van back-end, front-end en PostgreSQL	2017-10-02	5
Opdoen kennis van Angular4 en Spring Boot.	2017-10-02	5
Stageweek 2 2017-10-09		
Een overzicht krijgen van alle kandidaten in het systeem	2017-10-09	2
Een nieuwe kandidaat aanmaken in het systeem	2017-10-11	3
Stageweek 3 2017-10-16		
Een kandidaat beheren en aanpassen	2017-10-16	2
Loginsysteem met OAuth2	2017-10-18	3
Stageweek 4 2017-10-23		
Een overzicht van vacatures in het Systeem	2017-10-23	1
Een nieuwe vacature toevoegen	2017-10-24	1
Een vacature beheren en aanpassen	2017-10-25	1
Een overzicht van alle kandidaturen in het systeem	2017-10-26	1
Een kandidatuur beheren	2017-10-27	1
Stageweek 5 2017-11-06		
Via het detail van kandidaat een nieuwe kandidatuur registreren	2017-11-06	1
Op het detail van een kandidaat een overzicht zien van zijn kandidaturen	2017-11-07	1
Een overzicht krijgen van de contactmomenten voor een kandidatuur	2017-11-08	1

Een contactmoment met een kandidaat registreren	2017-11-09	1
Een contactmoment met een kandidaat beheren	2017-11-10	1
Stageweek 6 2017-11-13		
Een commentaar registreren op een contactmoment	2017-11-13	1
Een overzicht krijgen van de commentaren op een kandidaat	2017-11-14	1
Een commentaar registreren op een kandidaat	2017-11-15	1
Een verslag toevoegen aan een contactmoment.	2017-11-16	1
Stageweek 7 2017-11-20		
Kandidatuur voor een vacature stellen via de JIDOKA careers site	2017-11-20	3
CV opladen via de JIDOKA careers site	2017-11-23	2
Stageweek 8 2017-11-27		
Een overzicht krijgen van alle verslagtemplates	2017-11-27	3
Een verslagtemplate beheren.	2017-12-30	2
Stageweek 9 2017-12-04		
Een verslagtemplate verwijderen.	2017-12-04	2
Een verslag aanmaken op basis van een template	2017-12-06	2
Stageweek 10 2017-12-11		
Testing van applicatie	2017-12-11	5
Documenteren applicatie	2017-12-11	5
Stageweek 11 2017-12-18		
Verzorgen design en lay-out applicatie	2017-12-18	2
Tweaks en uitbreidingen op de applicatie – eventuele bugs oplossen	2017-12-20	3
Stageweek 12 2017-01-08		
Tweaks en uitbreidingen op de applicatie – eventuele bugs oplossen	2017-01-08	5

Voorbereiden eindpresentatie opdracht	2017-01-08	5

Tabel 2 Tijdsplanning stage

2.4.1 Opstart

De eerste weken van de stage dienden vooral voor de setup van het project: het ging voornamelijk over het installeren en het beter leren kennen van software die gebruikt ging worden tijdens de stage.

De eerste technologie die gebruikt werd was IntelliJ IDEA. Dit is een ontwikkelomgeving waarin gewerkt zal worden en die de talen die allemaal gebruikt zullen worden tijdens het project ondersteunt, waaronder Java, Typescript, HTML en CSS.

De volgende technologie die gebruikt werd was Spring Boot. Dit framework is op de PXL al aan bod gekomen, dus er was al enig inzicht op het maken van een RESTful API met Spring Boot. De manier om een Spring Boot API op te stellen was wel anders bij JIDOKA. Er werd verwacht dat we TDD gingen toepassen op het maken van het project. Om dit op een goede manier toe te passen zijn er tijdens het programmeren enkele *design patterns* toegepast, zoals het builder patroon. Dit zorgt ervoor dat je bv. voor een *constructor* maar 1 parameter moet meegeven. Onderstaande afbeelding is een voorbeeld van hoe het wordt gebruikt.

```
public Candidacy(CandidacyBuilder builder) {
    this.motivation = builder.motivation;
    this.candidate = builder.candidate;
    this.vacancy = builder.vacancy;
    this.candidacyStatus = builder.candidacyStatus;
}
```

Figuur 1 Builder principe

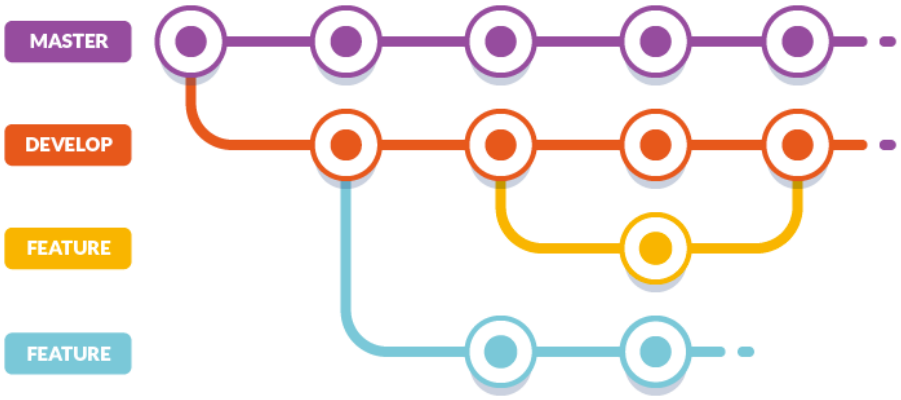
Postgres is de database-technologie waarvoor gekozen is. De backend zal met deze een connectie leggen zodat data vanuit hier kan gelezen en geschreven worden.

Voor de frontend moest ik Angular4 gebruiken. Voordat dit framework gebruikt kon worden, moesten eerst NPM en Node.js geïnstalleerd worden. NPM is de standaard JavaScript packagemanager die ervoor zorgde dat alle modules die nodig zijn om een Angular4 project op te starten aanwezig zijn. Verder gebruikt Angular4 Typescript, een sterk getypeerde programmeertaal die vandaag de dag nog niet door alle browsers begrepen wordt. Achterliggend komt hier Node.js bij kijken, wat ervoor gaat zorgen dat Typescript omgezet wordt naar JavaScript.

Angular4 was ook een nieuw concept. Op de PXL hebben we reeds de voorganger ervan bestudeerd, namelijk AngularJS, maar deze twee vallen totaal niet te vergelijken. Om dit framework een beetje onder de knie te krijgen is er een Pluralsight cursus gevolgd. Angular4 is een component-gebaseerd framework, dit wil o.a. zeggen dat je makkelijk je code kan hergebruiken.

Verder heb ik ook git-flow moeten aanleren. Dit is een principe van ontwikkelen met verschillende takken. Per functie die er wordt toegevoegd aan het project wordt er een tak toegevoegd vanuit de *develop*-tak. Hierop codeer je alleen wat nodig is voor de functie waar die tak voor staat. Wanneer de code klaar is om nagekeken te worden moet er op Bitbucket een *pull request* aangevraagd worden. Hierbij wordt de code nagelezen door een *code reviewer*. Wanneer de code getest en goedgekeurd is

kan de feature-tak terug gefuseerd mag worden met de *develop*-tak. Als dit allemaal correct is verlopen mag je vanuit de *develop*-tak een nieuwe tak aanmaken. Met deze manier van werken heb je altijd een stabiele versie staan op de *develop*-tak. Op het einde van het project wordt er een *merge* gedaan tussen de master en de *develop*, dit wordt dan de Release Build.



Figuur 2: GitFlow

2.4.2 Werking

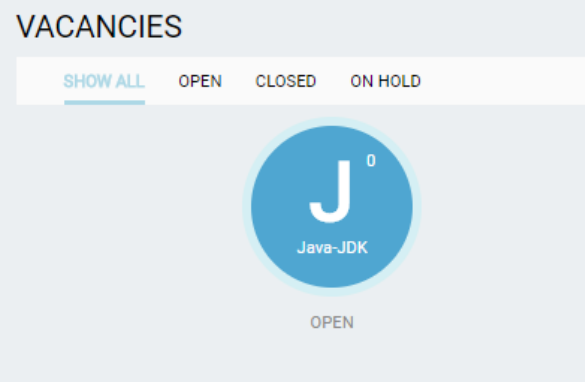
We zijn vanuit een domeinmodel en UCM beginnen werken (zie bijlage 2 en 3). Doorheen de applicatie werken we met kandidaten, vacatures en kandidaturen. In dit hoofdstuk wordt besproken waar ze globaal voor dienen en wat ze effectief met elkaar te maken hebben.

2.4.2.1 Vacatures

In onze stageopdracht is het de bedoeling dat alles van het recruitment-proces in deze applicatie wordt gestoken. Hierbij zijn we na de kandidaten begonnen met de vacatures.

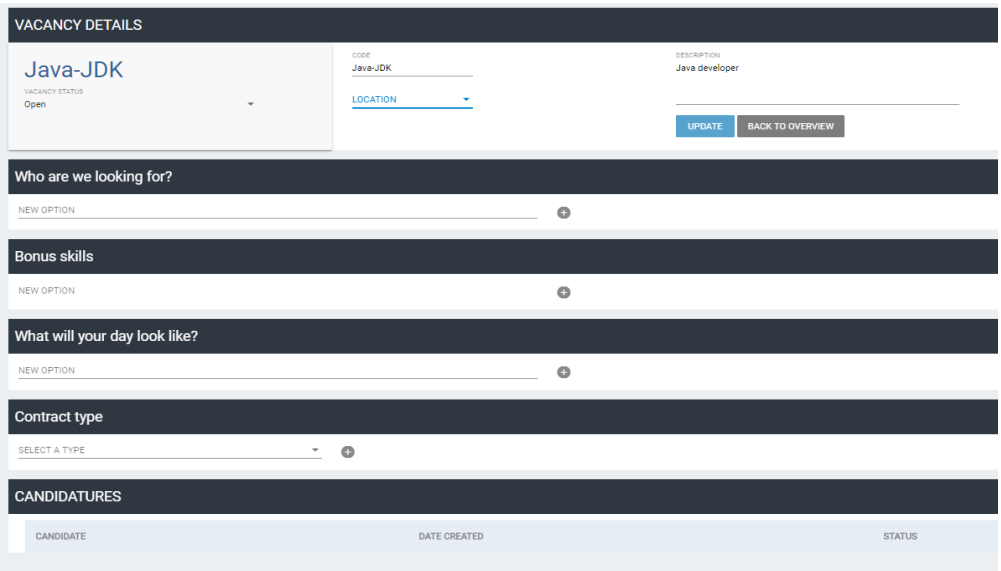
Vanaf het moment dat er een vacature openstaat bij JIDOKA kan deze worden toegevoegd aan het systeem. Bij het toevoegen van een vacature wordt er een code en een omschrijving meegegeven. Hierbij werd ook een status bijgehouden die aangepast kon worden. Deze status kon op open, gesloten of in de wacht staan.

Op het overzicht van vacatures kan onmiddellijk gezien worden hoeveel kandidaturen op de vacature geplaatst zijn.



Figuur 3: Overzicht vacatures

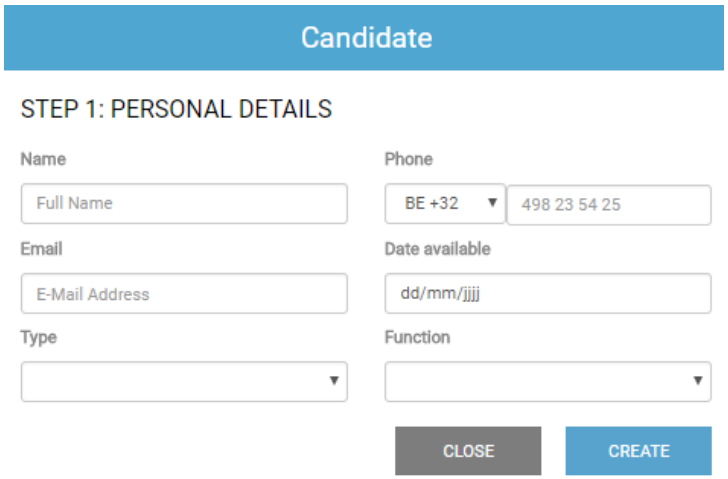
Na het toevoegen van een Vacature kan de gebruiker erop klikken en verdere details invullen. Details zoals de locatie, status, code,… kunnen ook allemaal aangepast worden. Dit kan gezien worden in afbeelding 4.



Figuur 4: Aanpassen vacature

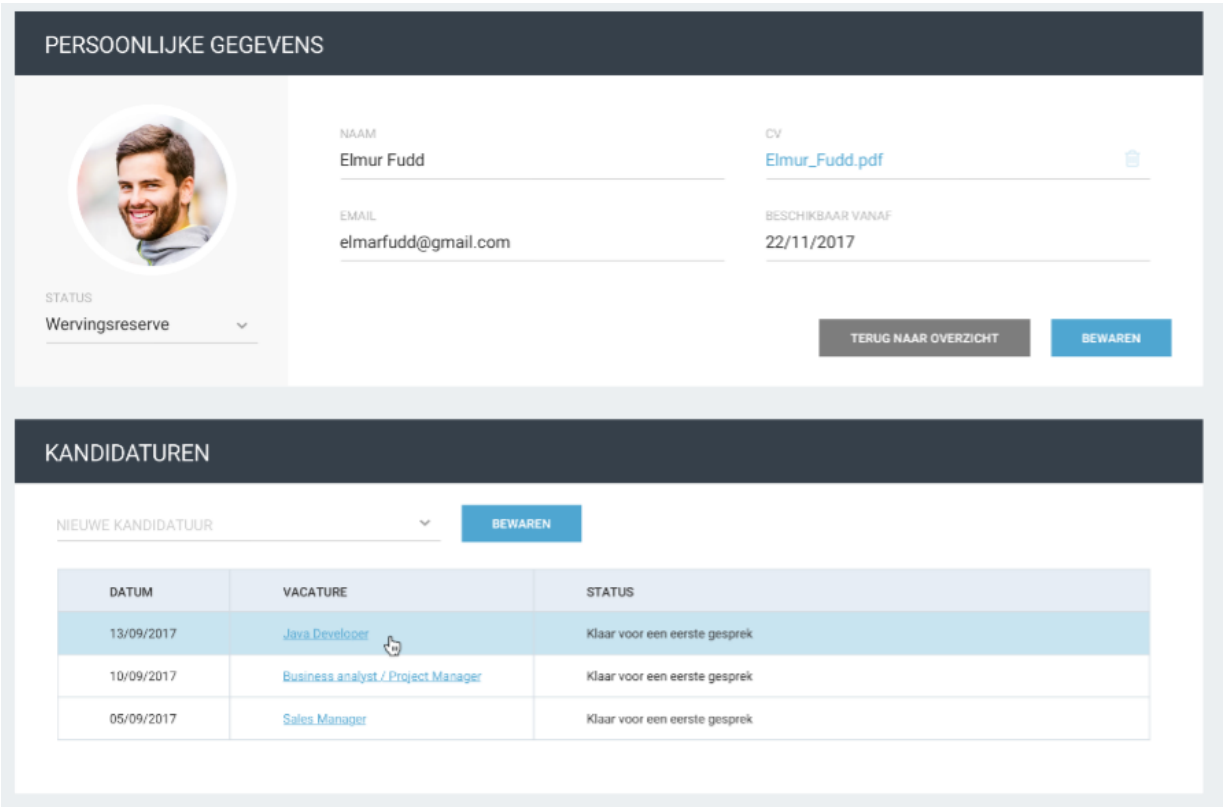
2.4.2.2 Kandidaten

Een kandidaat kan een kandidatuur indienen voor een vacature op de Careerswebsite van JIDOKA. Verder kan hij ook een mail sturen met zijn gegevens zodat een JIDOKA HR hem kan toevoegen aan het systeem. Er zijn vier types kandidaten: een student, een freelancer, een stagiair en een normale werknemer. Voor iedere soort kandidaat zijn er specifieke velden die moeten ingevuld worden. Verder bestaat ook de optie om een CV en een foto van de kandidaat toe te voegen aan het systeem.



Figuur 5: Toevoegen kandidaat

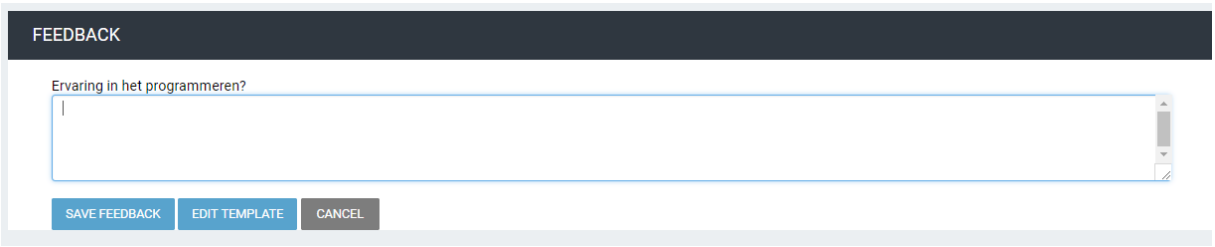
Als deze kandidaat is aangemaakt kan er verder op geklikt worden en kan een JIDOKA HR de kandidaat beheren. Er kunnen op de detailpagina kandidaturen en contactmomenten worden vastgelegd met de kandidaat. Verder kan er ook commentaar gegeven worden.



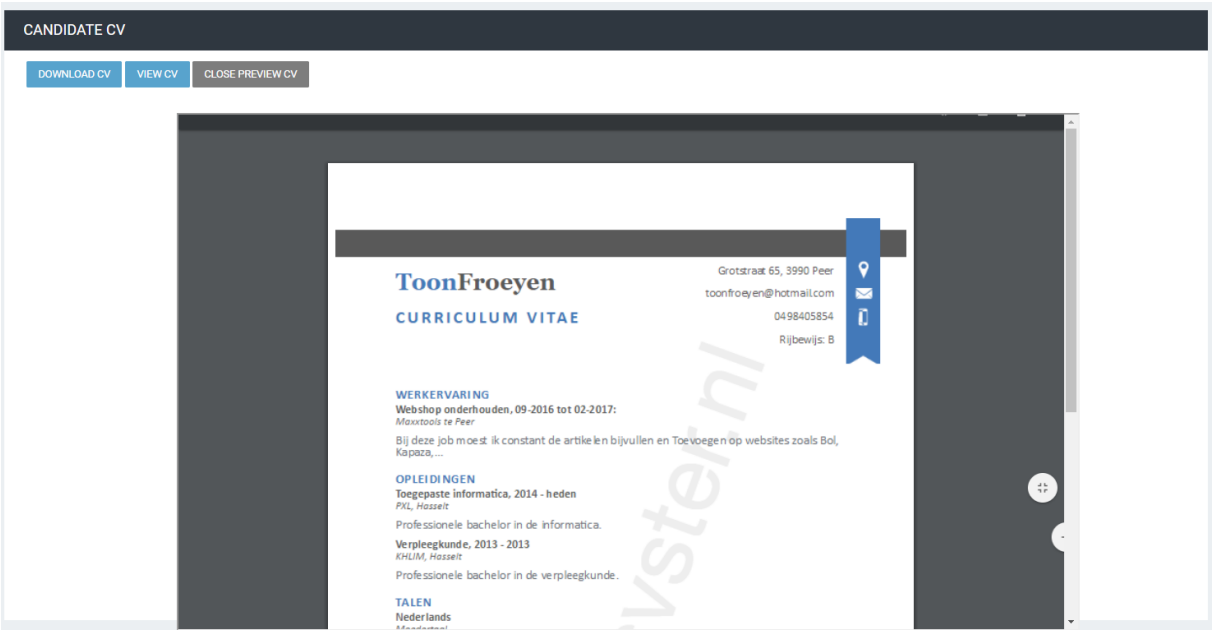
Figuur 6: Details kandidaat

2.4.2.3 Kandidaturen

Als een kandidaat een kandidatuur indient voor een vacature komt die in het overzicht te staan van de kandidaturen. In dit overzicht kan gefilterd worden op de code van vacature. Verder kan er ook op globale functie gefilterd worden. Als er verder wordt geklikt op een kandidatuur zie je hoever deze persoon staat op het vlak van zijn sollicitatie. Deze status kan aangepast worden. Zijn motivatie voor de job staat er ook bij, en er kan een contactmoment worden vastgelegd met de kandidaat. Zoals in figuur 7 en 8 te zien zijn kunnen verder er ook vragen beantwoord worden via een feedback formulier en kan ook de CV van de kandidaat bekeken worden.



Figuur 7: Feedbackformulier



Figuur 8: Show/hide CV

2.4.3 Functionaliteit

In dit hoofdstuk wordt er dieper ingegaan op hoe de functionaliteiten van de front- en backend te werk gaan en ook hoe deze samen tot één werkend geheel zijn gekomen. Verder gaan er ook nog enkele uitgelichte functionaliteiten aanbod komen.

2.4.3.1 Connectie

Onze eerste story's omvatten het toevoegen en beheren van een kandidaat. Dit wilt zeggen dat je in een formulier gegevens kon invullen en dat achterliggend de frontend deze waarden naar de backend ging versturen. De backend gaat dan deze gegevens omzetten naar een kandidaat object om deze op te slaan in de database. Bij het aanpassen van een kandidaat object worden er opnieuw gegevens doorgestuurd en wordt het gekende object aangepast met de nieuwe gegevens.

Dit wordt allemaal gebeurd via HTTP-calls. Zoals te zien is in onderstaande afbeelding wordt er een call gedaan naar een bepaalde URL die openstaat op de backend. Vervolgens vangt de backend deze call op en verstuurt hij deze naar de database om hem op te slaan.

```
addCandidate(candidate: Candidate): Promise<Candidate> {
  return this.post(this.candidateUrl, candidate);
}
```

Figuur 9: Post kandidaat naar backend

2.4.3.2 File-upload

Een kandidaat moet zijn CV in het systeem kunnen zetten. Hierbij moet er dus een bestand worden geüpload via de webapplicatie om deze dan vervolgens naar de backend sturen, waar het dan opgeslagen wordt in de database. Er moest gewerkt worden met Multipartfile in de backend, dit is je file die wordt omgezet naar een grote bytearray. Het voordeel hiervan is dat, als je de file gewoon als een base64(String) uploadt, deze wel eens tot 33% groter kan zijn omdat je een andere karakterset gebruikt.

Het bestand moest via Formdata doorgestuurd worden zodat dit via de backend geïnterpreteerd kon worden als Multipartfile.

```
fileEvent(event) {
  const files = event.target.files || event.srcElement.files;
  const file = files[0];
  if (file.name.split('.').pop() === 'pdf') {
    this.formData = new FormData();
    this.formData.append( name: 'file', file);
    this.fileFound = true;
  } else {
    alert('Enkel PDF bestanden toegelaten.');
  }
}

postFile() {
  this._candidateService.addDocument(this.formData, this.newCandidate.id)
    .then( onfulfilled: () => this.documentAdded = true);
}
```

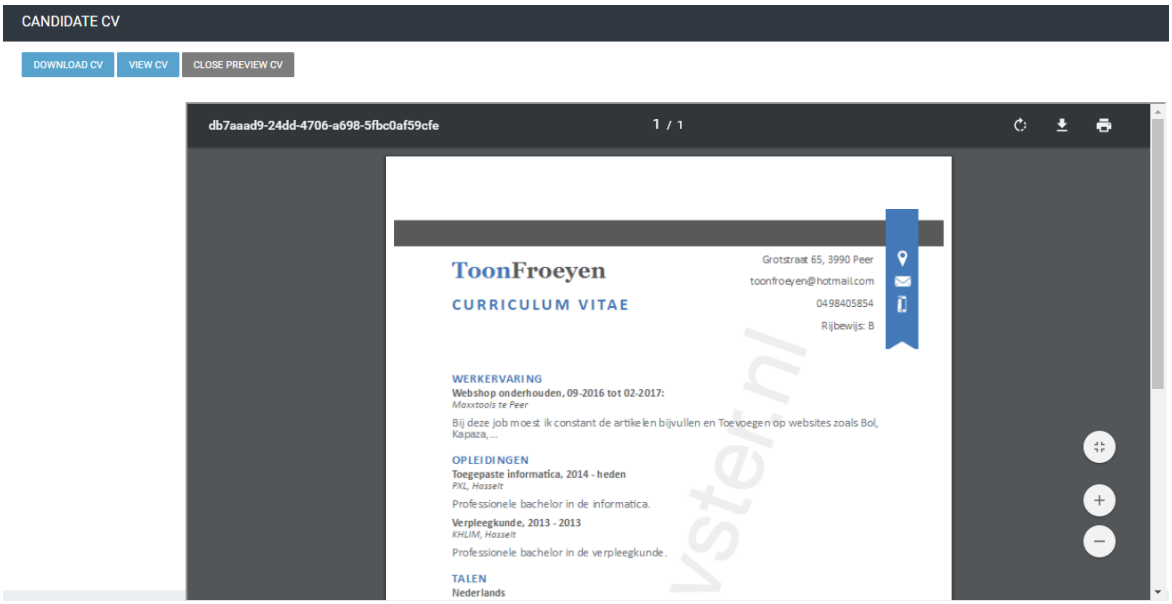
Figuur 10: Verstuur bestand naar Backend

De naam die gegeven werd aan het bestand kon dan als Multipartfile worden opgevangen in de backend.

```
@PostMapping("/{id}")
@PreAuthorize("hasRole('ADMIN')")
public DocumentDTO handleFileUpload(@PathVariable("id") Long id, @RequestParam("file") MultipartFile file)
    return documentService.save(file, id);
}
```

Figuur 11: Bestand wordt opgevangen in Backend

Zoals eerder al vermeld bestaat er in de applicatie bestaat de functie om de CV van een kandidaat ook terug op te halen. De base64 string die staat opgeslagen in de database wordt hierbij terug opgehaald. Die moet dan omgezet worden naar een BLOB om deze weer te geven op de webpagina. Hierna kan je de CV in een *IFrame*-tag steken om hem zo te bekijken. Verder kan de CV ook terug gedownload worden.



Figuur 12: Bestand in Iframe

2.4.3.3 Form-validatie

Form validatie is ook een vereiste bij CRUD-operaties om misbruik van kwaadwillende gebruikers te voorkomen. Onjuiste validatie van formuliergegevens is één van de belangrijkste oorzaken van beveiligingsrisico's. Het stelt de website bloot aan header-injecties, XSS en SQL-*injection*. Omdat Angular om het half jaar uitkomt met een nieuwe versie, is het nogal moeilijk om bij sommige aspecten zoals form validatie direct een relevant voorbeeld te vinden.

Verder zou er met form-validatie ook geen verkeerde data in de database terecht kunnen komen.

2.4.3.4 Master-detail

Voor kandidaten, vacatures en kandidaturen hebben we ieder een detailpagina gemaakt. Een master-detailinterface toont een hoofdlijst en de details voor het geselecteerde item. Mocht je op een kandidaat klikken dan zou je buiten zijn persoonlijke gegevens ook zijn kandidaturen en contactmomenten zien. Bij de vacatures zie je alle kandidaten met hun kandidatuur voor die vacature. Bij een kandidatuur zie je alle contactmomenten voor die specifieke kandidatuur.

2.4.3.5 DTO objecten

DTO Is een ontwerp patroon dat ervoor zorgt dat de backend alleen data gaat uitsturen die nodig is. Je kan verschillende DTO's maken om verschillende verzoeken voor andere data formaten door te sturen naar de frontend. Zoals je in onderstaande afbeeldingen ziet verschillen DTO's niet veel van gewone objecten. Hier wordt ook getoond dat de datum van aanmaken als string gezien wordt in backend verzoeken.

```
@JsonSerialize
@JsonIgnoreProperties
public class VacancyDTO implements Serializable {

    private Long id;
    private String code;
    private String description;
    @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "yyyy-MM-dd HH:mm")
    private LocalDateTime creationDate;
    private VacancyStatus vacancyStatus;

    private VacancyDTO(final VacancyDTOBuilder builder) {
        this.id = builder.id;
        this.code = builder.code;
        this.description = builder.description;
        this.creationDate = builder.creationDate;
        this.vacancyStatus = builder.vacancyStatus;
    }
```

Figuur 13 DTO principe

Door het gebruik van een mapper kunnen we het gewone object omzetten naar een DTO, zoals je ziet in de volgende afbeelding. Mocht je nu bijvoorbeeld de beschrijving niet willen meegeven, dan moet je deze in de builder weglaten. Dan zal hij ook niet worden opgehaald in de frontend.

```java
@Component
public class VacancyDTOMapper implements Mapper<Vacancy, VacancyDTO> {

    @Override
    public VacancyDTO map(Vacancy vacancy) {
        VacancyDTO.VacancyDTOBuilder vacancyDTOBuilder = VacancyDTO.VacancyDTOBuilder.aVacancyDTOBuilder();

        return vacancyDTOBuilder
                .id(vacancy.getId())
                .code(vacancy.getCode())
                .description(vacancy.getDescription())
                .creationDate(vacancy.getCreationDate())
                .vacancyStatus(vacancy.getVacancyStatus())
                .build();
    }
}
```

Figuur 14 Object naar DTO

2.4.3.6 Security

Om onze backend volledig veilig te maken is er gebruik gemaakt van Oauth2.0. Dit is een security-framework dat toegang-tokens en rollen gebruikt om acties die mogelijk zijn te kunnen uitvoeren.

Om dit framework toe te passen in Spring Boot moeten er eerst enkele tabellen toegevoegd worden aan de database. Deze zijn in bijlage 3 te vinden. Het principe is nu dat wanneer een gebruiker inlogt op de webapplicatie hij een token ontvangt. Samen met dit token worden er *requests* gestuurd naar de backend en wordt er gekeken of deze gebruiker voldoet aan de regels om deze *request* te mogen uitvoeren.

```java
    @GetMapping(value = "/all")
    @PreAuthorize("hasRole('ADMIN')")
    public Collection<CandidateDTO> findAll() { return candidateService.findAll(); }

    @PostMapping
    @PreAuthorize("hasRole('ADMIN')")
    public CandidateDTO save(@RequestBody T candidate) { return candidateService.save(candidate); }
```

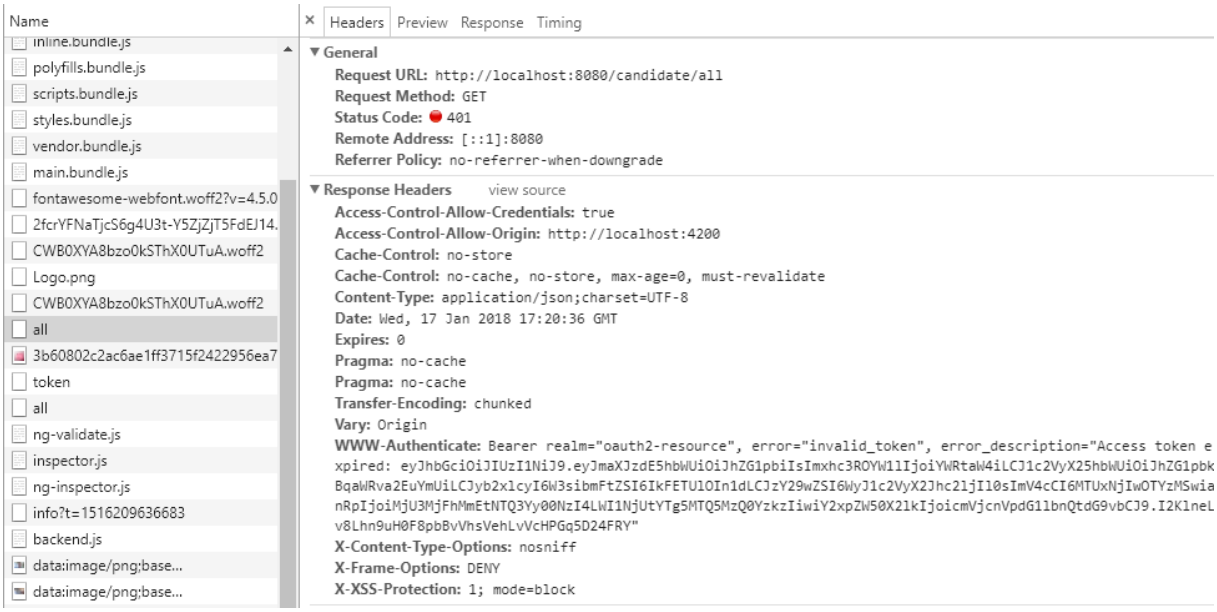
Figuur 15: Admin requests

Een token is een uur geldig. Als een gebruiker zich niet uitlogt of langer dan een uur actief is op de applicatie zal hij geen *requests* meer kunnen uitvoeren. Om dit probleem op te lossen moeten we de error opvangen die wordt gegooid als er een *request* niet kan worden uitgevoerd. Onderstaande afbeelding laat zien hoe we dit hebben opgelost.

```
post(url: string, object: any, params?: HttpParams): Promise<any> {
  const options = this.buildRequestOptions(params);
  const response = this.http.post(url, object, options);
  return response
    .toPromise()
    .then( onfulfilled: promise => {
      return promise ? promise : null;
    })
    .catch( onrejected: (error) => {
      if (error.status === 401) {
        return this.refreshToken()
          .then( onfulfilled: () => this.post(url, object, params), onrejected: () => this.logout());
      } else {
        Observable.throw(error);
      }
    });
}
```

Figuur 16: Http-call opnieuw proberen

Vanaf het moment dat er een 401 (*Unauthorized*) error wordt gegooid gaat de bestaande token vernieuwd worden. De oude refresh token wordt de nieuwe access token en er wordt een nieuwe refresh token aangevraagd. In afbeelding 17 kan je zien hoe dit proces in werkt.



Figuur 17: Refresh token aanvraag

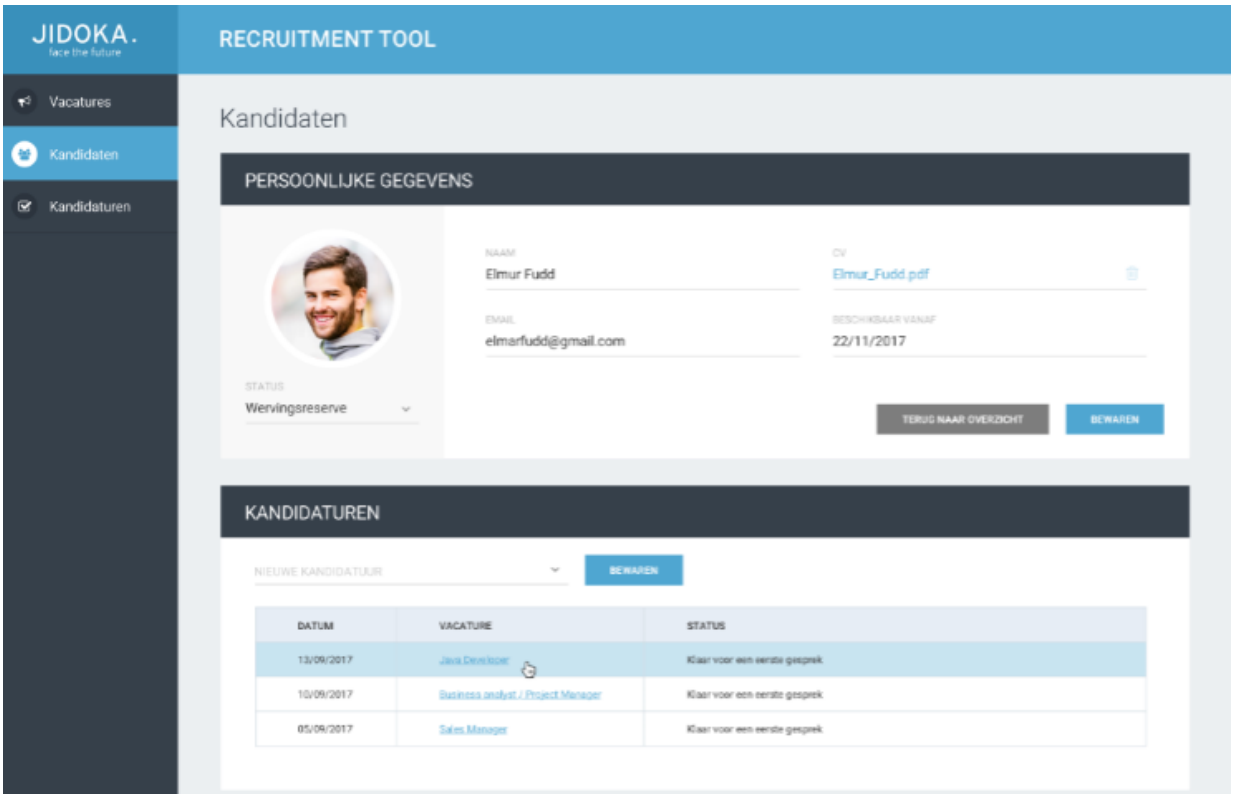
In de error-message staat dat de token niet meer geldig is. Wanneer je een 401 error krijgt wordt de refresh aangeroepen. Met deze nieuwe token wordt de request nog een keer aangeroepen. Mocht deze request nog een keer falen dan zal er uitgelogd worden.

2.4.4 Ontwerp

De *Creative designer* van JIDOKA heeft enkele designs gemaakt van hoe de website eruit ging zien. Deze waren a.d.h.v. *Material UI* gemaakt. Onderstaande afbeelding is een voorbeeld van hoe de stijl van onze website eruit moest gaan zien.

Om tot een responsieve webapplicatie te komen hebben is er voor de schaalbaarheid bootstrap gebruikt. Dit framework zorgt ervoor dat we de lay-out makkelijk kunnen aanpassen a.d.h.v de breedte van het scherm.

Tot op dit moment is er altijd Angular4 geïmplementeerd geweest in de applicatie. Omdat dit enkel *Alpha* en *Beta* versies van *Material UI* implementeerde is er een update naar Angular5 moeten gebeuren. Dit is zo goed als vlekkeloos gebeurd. De manier van het aanroepen van HTTP-calls is het enige verouderde kernonderdeel van Angular.



Figuur 18: Design Recruitment-Tool

2.5 Persoonlijke reflectie

Als ik nu terugkijk op de planning die gemaakt is en hoe het effectief is verlopen kan er geconcludeerd worden dat daar veel aan kan veranderen. In het begin werd deze planning goed gevolgd. Maar na verloop van tijd duurde het langer om bepaalde story's goed te keuren. Hierdoor lopen wachttijden op en werd de progressie vertraagd. Tijdens het uitvoeren van de opdracht veranderden de doelstellingen ook. Verder kwamen er knelpunten en problemen naar voren en kon het oplossen hiervan enige tijd duren.

De opdracht was leerrijk en interessant om te maken. Dit komt omdat er een nieuwe technologie was die ik moest gebruiken, namelijk Angular. De doelstelling van de opdracht was het recruitment-proces van JIDOKA in een Webapplicatie steken. Een opdracht van die grootte is iets wat ik nog niet gezien had. Een grote plus is dat er veel werknemers waren die hulp aanboden tijdens de opdracht.

De laatste weken van de opdracht waren de meest uitdagende, omdat dan de lay-out op punt moest gebracht worden. Persoonlijk ben ik niet echt goed op dit vlak. Dus het is zeker nuttig geweest om ook op dit aspect bij te leren. Hierbij was het werken met *Material UI* een leuke ervaring. Met dit framework heb ik geleerd dat er niet veel werk in lay-out moet gestoken worden om iets er relatief goed te laten uitzien.

Mijn kennis is zeker gegroeid. Ik heb niet alleen nieuwe technologieën gezien, maar ook de kans gehad om mijn *knowhow* van enkele technologieën die in de lessen werden onderwezen uit te breiden, zoals bijvoorbeeld Spring Boot. Ook het werken in een zakelijke omgeving, hulp krijgen van mensen die vaker met de technologieën werken en deel uitmaken van een ontwikkelteam was een geweldige ervaring.

3 Onderzoekstopic

3.1 Vraagstelling

JIDOKA is een IT-bedrijf dat veel webapplicaties maakt. Hierbij maken ze gebruik van enkele frameworks om dit te vergemakkelijken. Bij JIDOKA zeggen ze zelf dat ‚als je er wilt werken, je sowieso het React framework gaat moeten leren. Maar voor de stage die ik heb gekozen is bewust Angular4 opgelegd. Dus hierbij kwam ik vrij simpel uit bij mijn onderzoeksvraag: ‘Welk framework is het meest aangewezen voor JIDOKA, Angular4 of React?’. Hierdoor kan ik JIDOKA helpen om naar de toekomst toe tijd te besparen met de keuze van het juiste framework voor verschillende projecten.

Ik ga dit onderzoek voeren door bepaalde pagina’s die ik in mijn project heb gemaakt opnieuw te maken in React. Hierdoor kan ik vervolgens tot mijn antwoorden en conclusies komen.

Verder ga ik ook performantie testen kunnen doen, dit zullen de gewone testen zijn waaronder de laadtest. Met deze ga ik bijvoorbeeld kijken hoelang het voor ontwikkelaars duurt om een project op te zetten.

Ook kan ik de setup van projecten gaan vergelijken. Wat is de kost van het ene framework vergeleken met het andere.

3.2 Methode van onderzoek

3.2.1 Aanpak

In dit onderzoek wordt naar verschillende aspecten van beide frameworks gekeken. Eerst zal er voldoende informatie moeten opgedaan worden over beide frameworks. Daarna is het de intentie om te zien welk framework op welk vlak de betere is. Dus zullen deze getest worden om te zien wat de voor – en nadelen van beide opties zijn. Hierbij ga ik ten eerste zien wat het verschil is tussen de setup van beide projecten. Hierbij gaat het vooral over hoelang het duurt voor je kan beginnen met coderen. JIDOKA is ook een bedrijf dat schoolverlaters direct toelaat om te komen werken. Omdat niet elke schoolverlater Angular4/5 of React heeft gezien in zijn opleiding is het ook interessant om naar de leercurve van ieder framework te kijken. Voor de performantie zal ik Stefan Krause zijn framework benchmark gaan gebruiken.

3.2.2 Analyse

Om informatie op te doen worden er cursussen van verschillende bronnen bekeken of gelezen, zoals Pluralsight, of artikels gelezen op het internet. Ook kan er bij verschillende werknemers bij JIDOKA zelf gekeken worden, omdat ze ook al jaren met React werken, en aangezien deze stage in Angular4 wordt gemaakt, kunnen er zo conclusies getrokken worden.

3.2.3 Vergelijking

De vergelijking zal voor een groot deel al doende gebeuren. Zo wordt duidelijk hoe de setup van ieder project loopt of hoelang het duurt om deze op te zetten. Hierbij kan ik conclusies trekken en deze in woorden brengen voor de opdracht zelf. Verder zal er met de benchmark ook een vergelijking gemaakt kunnen worden op het vlak van performantie.

3.3 Onderzoek

3.3.1 Wat is het Angular Framework?

Angular (meestal gerefereerd met Angular2 of Angular2+) is een open-source Typescript library die wordt geleid door het Angular team bij Google. Net zoals React is Angular een populair framework om webapplicaties te maken. In plaats van puur Javascript gebruiken ze Typescript. Dit is een sterk getypeerde *superset* van JavaScript die achterliggend door de compiler wordt omgezet naar JavaScript. Hieronder worden enkele belangrijke functies besproken van Angular.

Zoals hierboven al vermeld gebruikt Angular de programmeertaal Typescript. Typescript is een sterk getypeerde taal zoals C#. Typescript heeft populariteit gewonnen door zijn simpliciteit en omdat het makkelijk te gebruiken is. Hieronder zie je een *class* die gemaakt is in Typescript.

```
class Greeter {
    greeting: string;
    constructor (message: string) {
        this.greeting = message;
    }
    greet() {
        return "Hello, " + this.greeting;
    }
}
```

Figuur 19 TypeScript class

Onderstaande afbeelding laat zien hoe dit er in JavaScript uit zou hebben gezien.

```
var Greeter = (function () {
    function Greeter(message) {
        this.greeting = message;
    }
    Greeter.prototype.greet = function () {
        return "Hello, " + this.greeting;
    };
    return Greeter;
})();
```

Figuur 20 JavaScript class

AngularJS (Angular 1.x) was bedoeld voor responsieve applicatieontwikkeling, er was nog geen mobiele ondersteuning. Angular heeft vandaag de dag ingebouwde ondersteuning om mobiele applicaties te maken.

Angular past ook Dependency Injection toe. Dit zorgt ervoor dat er afhankelijkheden van buitenaf geïnjecteerd worden in de applicatie. Hierdoor kan er loose coupling toegepast worden tussen klassen. Dit wil zeggen dat ze data uitwisselen zonder dat deze relatie in de code vastgelegd is.

Databinding is ook een groot deel van Angular. Als je een property aan een HTML tag wil binden, dan gebruik je de ngModel-functie. Als je de waarde van de *property* wilt gebruiken in je HTML-code, dan moet je deze in dubbele accolades plaatsen. Wil je aan two-way binding doen moet je gebruik maken van de ngModelChange-functie.

```
<input [ngModel]="username" (ngModelChange)="username = $event">

<p>Hello {{username}}!</p>
```

Figuur 21 Angular 2-way-binding

Omdat username 2 keer wordt getypt heeft Angular ook een korte notatie. Deze heet het Banana-in-a-box principe. Je gaat enkel nog ngModel gebruiken, maar in plaats van de vierkante haakjes gebruik je nu haakjes in deze vierkante haakjes: [(ngModel)].

3.3.2 Wat is het React Framework?

React is een open-source Javascript library die gemaakt werd door Jordan Walke; een software engineer die werkt bij Facebook. React zelf is als eerste in 2011 in de tijdlijn van Facebook gebruikt. React is heel populair omdat er grote webapplicaties mee gemaakt kunnen worden die data kunnen veranderen zonder de pagina te herladen. Hieronder worden in het kort enkele belangrijke functies aangehaald.

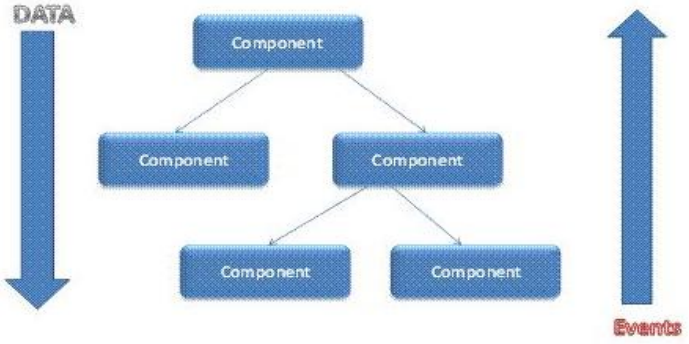
In React gebruiken ze voor het templaten JSX in plaats van standaard JavaScript. Dit is eenvoudige Javascript waarmee HTML quotering kan worden geïmplementeerd.

```
/** @jsx React.DOM */
var HelloMessage = React.createClass({
  render: function() {
    return <div>Hello {this.props.name}</div>;
  }
});
```

Figuur 22 JSX-voorbeeld

React heeft ook native libraries sinds 2015. Dit zorgt ervoor dat je mobiele applicaties kan maken met het framework.

De data wordt ook maar in 1 richting gestuurd. Er wordt een reeks van onveranderlijke waarden doorgegeven aan de componenten. Deze worden gebruikt als eigenschappen (properties) in de HTML-tags. De component kan eigenschappen niet direct aanpassen maar wel met behulp van een callback-functie wijzigingen doorbrengen. Dit hele proces heet '*properties flow down; actions flow up'.*



Figuur 23 Properties flow down Actions flow up

React maakt ook een in-memory datacache die de aangebrachte wijzigingen berekent en vervolgens in de browser bijwerkt. Dit betekent dat je de applicatie niet helemaal terug moet aanzetten, maar gewoon je pagina moet vernieuwen om de veranderingen die je in je code hebt gemaakt te zien.

3.3.3 Setup

Bij een setup wordt er gekeken naar de benodigdheden en naar de simpliciteit van het opzetten.

Setup Angular

Voordat je een Angular project kan opzetten zal NPM eerst geïnstalleerd moeten worden. Dit is de standaard packagemanager voor Javascript projecten. Hierna zal je dit commando in de opdrachtprompt kunnen gebruiken.

Node.js zal best ook geïnstalleerd worden voor je met de effectieve setup van het project begint. Angular gebruikt Typescript, een programmeertaal die de browsers van vandaag nog niet snappen, en dus zal het op een andere manier gecompileerd moeten worden. TypeScript zal achterliggend worden omgezet naar JavaScript, een taal die bijna alle browsers begrijpen.

Via NPM kan je nu ook de Angular CLI installeren. Deze zorgt ervoor dat je via de opdrachtprompt makkelijker Angular projecten kan opstarten.

```
1 npm install -g angular-cli
```

Figuur 24 Installeer Angular CLI

Nadat je naar je projectenmap bent veranderd kan je een nieuw project opstarten met de standaard mappen van Angular

```
1 ng new my_first_angular_app
```

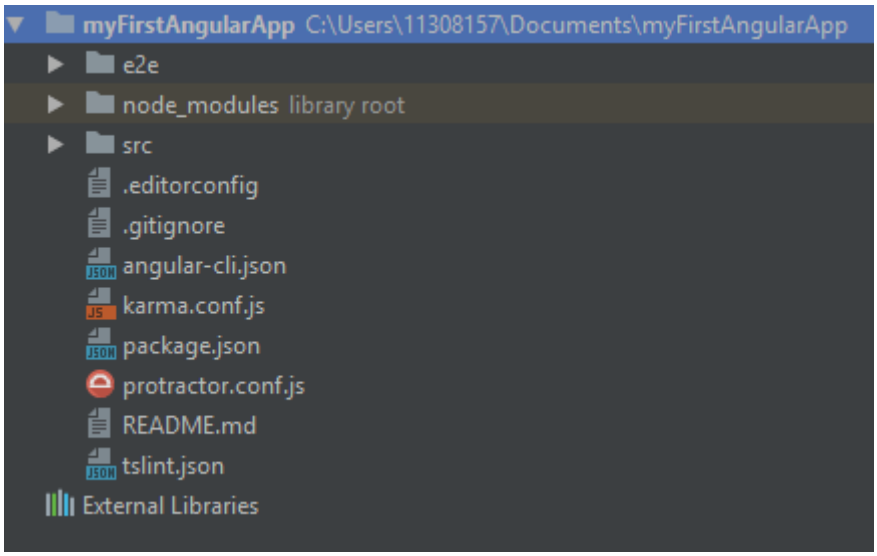
Figuur 25 Maak Angular applicatie

Verander nu naar de locatie je nieuw project en gebruik onderstaand commando om je applicatie op “localhost:4200” te zien.

```
1 ng serve
```

Figuur 26 Start Angular applicatie

Als je het project opent in een IDE zal de projectstructuur er als onderstaande afbeelding uitzien. Mocht je nog dependencies toevoegen aan het project, dan kan je deze toevoegen door npm install <naam dependency> uit te voeren in de map van het project.



Figuur 27 Angular mappenstructuur

Setup React

Net zoals bij het opzetten van Angular zul je eerst NPM en Node.js moeten installeren. NPM is meestal nodig omdat dit voor de meeste JavaScript frameworks de standaard package manager is.

Node.js wordt gebruikt om een webserver op te zetten en het project in de browser te zien.

React heeft een tijd gehad dat het echt wel een tijdje duurde om een project op te zetten. Hier hebben ze op ingespeeld door de volgende commando's te maken.

```
npm install -g create-react-app
create-react-app my-app
```

Figuur 28 Commando React project

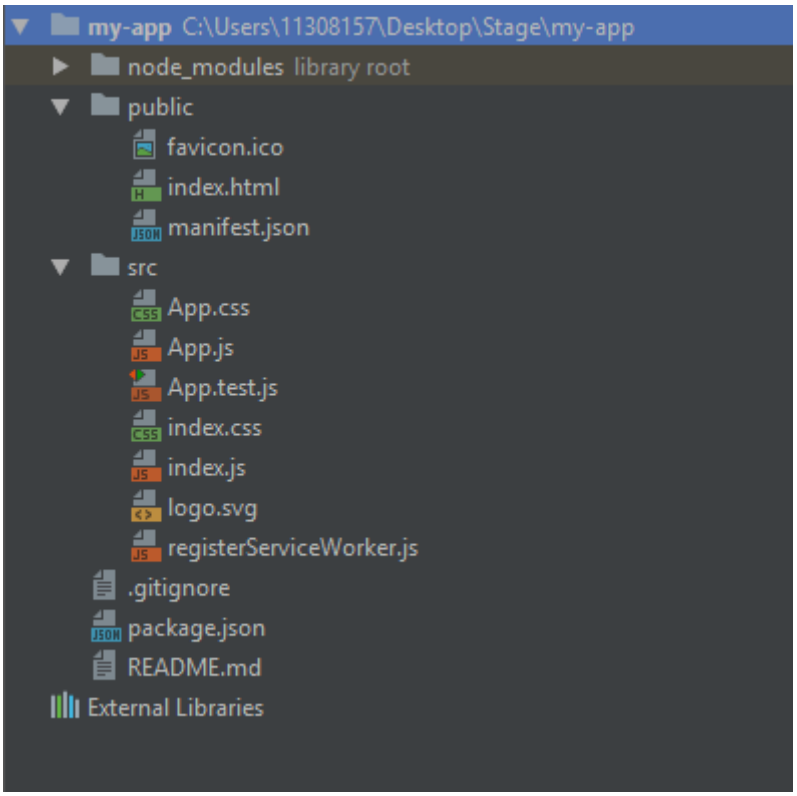
Verder als je het project wilt starten moet je enkel naar de project map veranderen en het volgende commando uitvoeren.

```
cd my-app
npm start
```

Figuur 29: Starten React applicatie

Nu zal je applicatie starten en te zien zijn op localhost:3000

De mappenstructuur van een React project ziet er als volgt uit.



Figuur 30 Standaard mappenstructuur React

3.3.4 Leercurve

Als een bedrijf nieuwe werknemers aanwerft, wil het natuurlijk dat ze zo snel mogelijk kunnen werken. Hierbij moet je ook kijken naar de complexiteit van het framework en hoe makkelijk of moeilijk het is om aan te leren.

Ondanks de uitgebreide documentatie is de leercurve voor Angular2+ niet zo steil. In dit framework moeten er verschillende aspecten geleerd worden voordat het fatsoenlijk begrepen wordt.

Typescript is de programmeertaal die achterliggend wordt gebruikt in Angular2+. Als je JavaScript als webdeveloper gewoon bent, dan zal dit in begin niet al te simpel zijn. Maar het lost het probleem op van een zwak-getypeerde programmeertaal zoals JavaScript zelf. Verder moet je het principe van MVC leren kennen, ook hierbij wordt er een iets steilere leercurve verwacht.

De leercurve van React is steiler, veel mensen raken er binnen de week mee weg. Hoewel je in het begin voor veel aspecten staat waar je nog nooit van gehoord hebt zoals: *actions*, *components*, *state*, *reducers*,… lijken deze enkel gecompliceerd op het eerste zicht.

Het is ingewikkeld omdat de kans groot is dat je nog nooit een framework gezien hebt dat werkt met deze structuur. Als backend developer is het logisch om sneller met Angular weg te zijn. Het lijkt op de structuur van een backend met services en controllers.

3.3.5 Support/documentatie

Om naar de problemen van het heden en de toekomst te kijken bij het maken van een webapplicatie met een bepaald framework zal er ook een goede documentatie en support moeten zijn.

Allereerst is er gekeken op de meest bekende website, namelijk stackoverflow. Hierop valt te zien dat van de 90 000+ vragen die gesteld zijn er nog 38 000+ niet beantwoord zijn. Ook heeft Angular op Github 563 personen die eraan meegewerkt hebben. Veder hebben ze ook nog een documentatie-website. Deze heeft een quickstart guide en ook een volledige uitleg van hoe het framework ineen zit. Verder vind je hier ook de api van alle functionaliteiten die het aanbiedt.

Op stackoverflow doet React het ongeveer hetzelfde als Angular, van de 69 000+ vragen heeft het er 26000+ nog niet beantwoord. Op GitHub hebben ze 1156 mensen die eraan hebben meegewerkt. Verder heeft het ook nog een heel uitgebreide documentatie met veel uitleg over hoe het framework te werk gaat.

3.3.6 Migratie naar vorige versies

Hoe makkelijk is het om een upgrade of downgrade te doen van je framework. Angular en React worden jaarlijks 2 keer geüpdatet. Om gebruik te maken van 3rd party libraries is het nodig om van versie te veranderen.

Angular heeft bij de upgrade van 1 naar 2 grote problemen gehad. Enkele bedrijven hebben destijds op Angular 1 ingezet, maar je kan de twee niet vergelijken met elkaar. Bijgevolg hebben sommige bedrijven hun angular 1 projecten buiten gesmeten en volledig terug opnieuw moeten beginnen.

Vanaf Angular 2 is dit probleem opgelost geraakt. Aanpassingen die moeten gebeuren staan in de notities van de update.

React heeft dit van begin af aan toegepast. Je moet maar met enkele CLI commando's de dependencies in de package.json naar de gewenste versie brengen en dan opnieuw installeren.

3.3.7 Licentiemodel

Angular en React hebben hetzelfde Licentiemodel, namelijk de MIT-licentie. MIT is een licentie voor opensourcesoftware. De software die onder deze licentie valt mag gratis gebruikt worden. Als een permissieve licentie legt het enkel een heel kleine beperking op het hergebruik van deze software.

3.3.8 Virtual DOM

De DOM is de uitkomst van HTML code en de manipulatie van JavaScript op deze HTML. Deze uitkomst is te zien als we de *developer* tools openen en op *Elements* klikken. React gebruikt zijn eigen soort DOM als er aanpassingen zijn. Dit principe heet de *virtual* DOM. Dit neemt een kopie van de bestaande DOM en vergelijkt wat er veranderd is. Vanuit dit resultaat worden enkel de HTML nodes vervangen waar er aanpassingen in zijn gebeurd. Angular daarentegen gebruikt dit niet en vervangt heel de pagina wanneer er veranderingen worden vastgesteld. Dit is kostelijk t.o.v. React zijn *virtual* DOM.

3.3.9 AOT

Angular heeft de kans om bij de compilatie gebruik te maken van AOT. De template wordt gecompileerd tijdens de *build time* door de compiler als onderdeel van het *build process* uit te voeren. De browser laadt vervolgens alleen het gegenereerde JavaScript voor een bepaald onderdeel. Als gevolg hiervan hoeft de Angular-compiler niet door de browser te worden geladen en hoeft de browser de template niet elke keer te compileren als de toepassing wordt geladen. Dit zou de opstarttijd van Angular twee of drie keer zo snel moeten laten gaan.

React gebruikt TypeScript of Babel om zijn JSX om te zetten naar JavaScript. React is al een stuk sneller d.m.v. de virtual dom die het implementeert. Hieruit kunnen we stellen dat React het AOT principe niet nodig heeft.

3.3.10 Flexibiliteit

Angular heeft zijn manier van werken. Ze hebben hun eigen MVC componenten die moeten gerespecteerd worden. React laat zijn gebruikers open tot meer flexibiliteit. Het is enkel de ‘V’ in MVC. De ‘M’ en ’C’ kunnen zelf door de gebruiker gekozen worden.

3.3.11 Performance testing

De basis van volgende testen zijn te vinden in de github repository 'js-framework-benchmark' van Stefan Krause. Deze is te vinden op volgende link:

https://github.com/krausest/js-framework-benchmark

Stefan Krause heeft tijd gestoken op het vlak van performance van de meeste JavaScript frameworks. Hij heeft een benchmark gemaakt die het mogelijk maakt om te zien hoeveel tijd er in wat wordt gestoken, verder wordt er ook gekeken op het vlak van geheugenallocatie. Deze benchmark maakt een grote tabel, plaatst hier willekeurige gegevens in en meet de tijd die het kost om verschillende operaties uit te voeren.

Voor dit onderzoek werden alle testen in Chrome versie 59 uitgevoerd.

Soort testen

De volgende testen worden voor beide JavaScript-frameworks uitgevoerd:

- **Creëren van rijen:** tijd die nodig is om duizend tabelrijen na het laden van de pagina te creëren.
- **Vervangen van alle rijen:** tijd die nodig is om alle 1000 tabelrijen te updaten.
- **Selecteer rij:** tijd die het kost om een rij te selecteren
- **Rijen wisselen:** tijd om steeds 2 rijen te wisselen in een tabel van 1000 rijen.
- **Rijen verwijderen:** tijd die het kost om een rij te verwijderen
- **Veel rijen toevoegen:** tijd die het kost om 1000 rijen toe te voegen bij een tabel van 10 000 rijen
- **Veel rijen wissen:** tijd die nodig is om gegevens te verwijderen in een tabel met 10 000 rijen
- **Beschikbaar geheugen:** Beschikbaar geheugen na het laden van de pagina
- **Run geheugen:** Geheugengebruik na het creëren van duizend rijen

Keyed vs non-keyed

De meeste frameworks vandaag de dag hebben een soort van *binding* of relatie tussen de data en de DOM. Dit is vooral interessant wanneer de data bestaat uit een lijst in plaats van een enkel item. Als er waarden of gegevens veranderen dan kan de vraag volgen waar er op de DOM iets veranderd en op welk element deze toepassing dan wordt uitgevoerd. Deze veranderingen kunnen updates, inserties of verwijderingen zijn in de DOM.

Frameworks zoals React of Angular laten het toe om met de DOM een soort relatie te maken. Deze relatie is dat er een data-item wordt verbonden met een DOM node. Bij beide frameworks gebeurt dit door het specificeren van *Trackyby*. Bij Angular is de directive **ngFor* een goed voorbeeld, het wordt gebruikt als een soort identicator. Vanaf dit moment kan er gesproken worden van *keyed*.

Bij non-keyed zal een verandering in de data de bijbehorende elementen in de DOM aanpassen die voorheen met andere data waren gekoppeld. In theorie zal dit een stuk betere presentaties geven omdat er minder op de DOM zal uitgevoerd worden. Deze acties zullen alle bestaande nodes bijwerken om nieuwe data weer te geven. Voor Angular en React geldt dat wanneer er gebruik wordt gemaakt van 'item index' er kan gesproken worden van *non-keyed*

Kort samengevat, mochten we een tabel van 1000 rijen aanpassen in de *keyed-mode*, dan zal de hele DOM verwijderd worden en terug opnieuw 1000 rijen toevoegen. Bij *non-keyed* zal enkel de tekst in de tabel aangepast worden en zullen er geen nieuwe objecten gemaakt worden.

Resultaten

Zoals hierboven al vermeld is er een onderscheid gemaakt tussen de *keyed* en *non-keyed*. Hierbij is het ook belangrijk te vermelden dat je deze niet met elkaar kan vergelijken. In onderstaande tabellen wordt er dan ook een onderscheid gemaakt tussen de twee.

Keyed resultaten: Tijd in milliseconden

Name	angular-v5.0.0-keyed	react-v16.1.0-keyed
create rows Duration for creating 1000 rows after the page loaded.	185.7 ± 7.8 (1.0)	201.2 ± 12.1 (1.1)
replace all rows Duration for updating all 1000 rows of the table (with 5 warmup iterations).	179.3 ± 6.5 (1.1)	169.0 ± 4.3 (1.0)
partial update Time to update the text of every 10th row (with 5 warmup iterations) for a table with 10k rows.	73.5 ± 4.9 (1.0)	90.9 ± 3.3 (1.2)
select row Duration to highlight a row in response to a click on the row. (with 5 warmup iterations).	7.6 ± 4.0 (1.0)	12.4 ± 4.1 (1.0)
swap rows Time to swap 2 rows on a 1K table. (with 5 warmup iterations).	118.5 ± 2.8 (1.0)	121.8 ± 4.2 (1.0)
remove row Duration to remove a row. (with 5 warmup iterations).	46.1 ± 2.6 (1.0)	51.5 ± 2.0 (1.1)
create many rows Duration to create 10,000 rows	1,682.0 ± 53.1 (1.0)	2,033.7 ± 32.0 (1.2)
append rows to large table Duration for adding 1000 rows on a table of 10,000 rows.	257.6 ± 11.1 (1.0)	271.8 ± 9.9 (1.1)
clear rows Duration to clear the table filled with 10.000 rows.	360.3 ± 16.4 (1.6)	224.4 ± 6.0 (1.0)
slowdown geometric mean	1.06	1.08

Figuur 31: Keyed resultaten

In bovenstaande figuur zien we de resultaten van de eerste benchmarks die zijn uitgevoerd. De resultaten zijn van de recentste versies van Angular en React. *De keyed mode* wordt in deze resultaten ondersteund.

Als we de resultaten even langs elkaar leggen dan kunnen we zien dat het verschil miniem is, maar dat Angular toch wel lichtjes de bovenhand heeft.

De gegenereerde tabel heeft de testnamen in het Engels gepresenteerd, daarom worden deze hieronder ook als zodanig gebruikt.

- Select row

  Hoewel beide het heel goed doen, doet Angular het toch een stukje beter. Het doet er 7,6 milliseconden over terwijl React er 12,4 milliseconden erover doet. Dit betekend dat Angular ongeveer 39% sneller is.

- Create many rows

  Hier is een duidelijk voordeel te zien bij Angular: dit duurt 1682 miliseconden terwijl dat bij react 2033,7 milliseconden duurt. Hieruit kunnen we concluderen dat Angular ongeveer 17% sneller is dat React.

- Clear rows

  De tijd die het kost om rijen te verwijderen uit een tabel is voor Angular 360,3 milliseconden terwijl dit voor React 224,4 milliseconden is. Dit is wel een significant verschil waarin React ongeveer 38% sneller is dan Angular.

De rest van de testen die zijn gebeurd zijn nagenoeg gelijk in snelheid.

Geheugenallocatie in MB's

Bij het geheugen gebruik van beide frameworks is duidelijk te zien dat React het een stuk beter doet dan Angular. Angular gebruikt 6,7 megabyte terwijl React er maar 3,7 gebruikt. Dit is omdat Angular veel groter is dan React. We kunnen hieruit concluderen dat Angular het wel beter doet op vlak van prestaties, maar dat het wel veel meer geheugen verbruikt dan React.

Als we kijken naar het geheugen dat in gebruik is na het toevoegen van 1000 rijen doet React het ook een stuk beter dan Angular: het gebruikt net geen 3 megabyte minder dan Angular.

Name	angular-v5.0.0-keyed	react-v16.1.0-keyed
ready memory Memory usage after page load.	6.7 ± 0.1 (1.8)	3.7 ± 0.1 (1.0)
run memory Memory usage after adding 1000 rows.	10.5 ± 0.0 (1.4)	7.6 ± 0.0 (1.0)

Figuur 32: Geheugenallocatie Keyed

Non-keyed resultaten in milliseconden

Als er nu wordt gekeken naar de *non-keyed* resultaten dan kan er worden afgeleid dat de resultaten voor beide frameworks een stuk beter zijn. Dit heeft zoals eerder vermeld te maken met het feit dat *non-keyed* implementaties bestaande 'DOM elementen' hergebruiken.

Name	angular-v5.0.0-non-keyed	react-v16.1.0-non-keyed
create rows Duration for creating 1000 rows after the page loaded.	188.6 ± 6.1 (1.0)	187.4 ± 4.2 (1.0)
replace all rows Duration for updating all 1000 rows of the table (with 5 warmup iterations).	60.1 ± 5.4 (1.0)	67.0 ± 2.4 (1.1)
partial update Time to update the text of every 10th row (with 5 warmup iterations) for a table with 10k rows.	75.1 ± 4.1 (1.0)	91.9 ± 5.9 (1.2)
select row Duration to highlight a row in response to a click on the row. (with 5 warmup iterations).	9.6 ± 3.1 (1.0)	10.1 ± 4.5 (1.0)
swap rows Time to swap 2 rows on a 1K table. (with 5 warmup iterations).	14.4 ± 5.0 (1.0)	12.7 ± 4.8 (1.0)
remove row Duration to remove a row. (with 5 warmup iterations).	34.3 ± 4.5 (1.0)	42.8 ± 1.9 (1.3)
create many rows Duration to create 10,000 rows	1,664.7 ± 72.3 (1.0)	2,039.9 ± 45.4 (1.2)
append rows to large table Duration for adding 1000 rows on a table of 10,000 rows.	262.6 ± 10.3 (1.0)	262.0 ± 6.7 (1.0)
clear rows Duration to clear the table filled with 10.000 rows.	355.4 ± 23.3 (1.5)	230.2 ± 7.6 (1.0)
slowdown geometric mean	1.05	1.09

Figuur 33: Non-keyed resultaten

Wat opvalt als we de *non-keyed* en *keyed* resultaten gaan vergelijken is dat bijna alle resultaten van *non-keyed mode* sneller zijn t.o.v. dezelfde testen in *keyed mode* van dezelfde frameworks. Uit deze gemiddelde prestatieverbetering kan je concluderen dat beide frameworks er goed door ondersteund worden.

Geheugenallocatie in MB's

Als er wordt gekeken naar het geheugen verbruik van beide frameworks is er geen verschil op te merken tussen de *keyed mode* en de *non-keyed mode*.

Name	angular-v5.0.0-non-keyed	react-v16.1.0-non-keyed
ready memory Memory usage after page load.	6.7 ± 0.1 (1.8)	3.7 ± 0.1 (1.0)
run memory Memory usage after adding 1000 rows.	10.5 ± 0.0 (1.4)	7.6 ± 0.0 (1.0)

Figuur 34: Geheugenallocatie non-keyed

3.4 Literatuurstudie

3.4.1 Introductie

De vergelijking tussen Angular en React is niets nieuw. Tijdens deze literatuurstudie wordt er gekeken naar wat verschillende bronnen hebben te zeggen over de twee. Spreken ze elkaar toe of tegen?

3.4.2 Bron 1: Angular vs. React - the tie breaker

De eerste bron komt vanuit de website genaamd: www.airpair.com. Amit Kaufman heeft de vergelijking tussen beide frameworks gedaan en test ze op verschillende vlakken.

Ten eerste gaat hij de verpakking vergelijken. Verpakking is het vermogen om de code uit te voeren en te implementeren zoals jij dat wil. Dit geeft ook de mogelijkheid om nieuwe functionaliteiten te ontwikkelen zonder de laadtijd te vertragen. Angular heeft volgens hem vrij weinig vermogen om dit te kunnen. Buiten het hergebruik van HTML-templates heeft het weinig andere mogelijkheden om aan verpakking te doen. Bij React gebruiken ze JavaScript. Hierbij kunnen ze een module lader genaamd RequireJS gebruiken. Dit geeft de mogelijkheid om grote delen code te laten Lazy loaden.

Bij Angular is de leercurve niet altijd in stijgende lijn. De levenscyclus is vrij complex en om deze fatsoenlijk te leren kennen moet je de code echt goed lezen. Bij React zou je hoge standaarden kunnen zetten in een week. Het heeft ook maar enkele levenscyclussen en deze zijn vanzelfsprekend. Het grootste voordeel dat ze op dit vlak met React hebben gevonden is dat ze nooit echt de code moeten lezen om het te snappen.

Amit heeft het in zijn artikel ook over de abstractie van beide frameworks, dit biedt snelle ontwikkeling en verbergt details die niet nodig zijn voor de ontwikkelaar. Ze vinden de abstractie van Angular een beetje lekkend. Dit wil zeggen dat je het onderliggende model moet snappen om er werkelijk mee te kunnen werken. Daarom moeten zoveel mensen de hoeken van Angular debuggen om hun code te laten werken. Er zijn veel concepten om dit te onderscheppen ,zoals richtlijn prioriteit, maar hierdoor kan je je andere prioriteiten niet controleren en gaan ze soms niet mooi samen in dezelfde HTML pagina. React zijn abstractie is minder flexibel in sommige delen, zoals het niet kunnen toevoegen van attributen aan je HTML tags, of een tag voor één component. Maar dit wordt wel opgelost door React zijn implementatie van *mixins*, dit zorgt ervoor dat er een overlap gebeurt op levenscyclusmethoden.

Op het vlak van modelcomplexiteit is Angular zijn performance gevoelig wanneer het op de scope aankomt. Dit komt door de *copy-n-compare*, wat wil zeggen dat je geen grote models kan gebruiken. Dit heeft zijn voor- en nadelen: het maakt je code simpeler en meer testbaar, maar het zorgt er wel voor dat je de dingen die je normaal gebruikt moet afbreken. React geeft je de keuze om dit wel of niet te doen.

Verder gaat hij ook wat dieper in op het debuggen van beide frameworks. Hij splitst dit op in het uitzoeken waarom je code niet werkt en het verstaan van de HTML uitvoer hierbij. Angular is een evenement gestuurd systeem. Dit is altijd makkelijker om te schrijven en moeilijker om

te debuggen, maar ze doen er wel goed aan om constructies te leveren die logisch zijn, zoals services. Als deze correct gebruikt worden dan maken ze de code een stuk makkelijker om te testen. Als er iets niet werkt in de Angular directieven is de ene optie om je code te herschrijven en de andere Angular zijn code zelf te debuggen. Bij React heb je twee scenario's: het updaten van een model en het éénrichtingsverkeer-renderen dat altijd hetzelfde is. Dit wil zeggen dat er minder plaatsen zijn om te zoeken naar bugs.

Het binden van Angular alleen via de scope zelf, dus voor complexe scenario's zoals asynchrone acties, zal je een model tussenin moeten gebruiken. React heeft voor het binden van een model iets dat "valueLink" heet, 1 attribuut voor zowel "value" als "onChange" velden. Dit concept lijkt simpel maar toch doet het zijn werk.

Templating is misschien nog wel het belangrijkste aspect: als je een webapplicatie of een gewone website aan het maken bent is het schrijven van de gebruikersomgeving hetgeen waar je het meeste mee bezig bent. Amit noemt ze panelen die veel informatie bevatten. React doet het hier minder goed, zoals je ziet in onderstaande afbeelding moet je een herhaling andersom schrijven.

```
var createItem = function(itemText) {
    return <li>{itemText}</li>;
};
return <ul>{this.props.items.map(createItem)}</ul>;
```

Mocht je dit in Angular doen dan ziet dit er zo uit:

```
<ul>
    <li ng-repeat="item in items">{{item}}</li>
</ul>
```

3.4.3 Bron 2: Angular vs React — the DEAL BREAKER

Dominik T., een software developer, heeft ook tijd gestoken in een vergelijking tussen Angular en React.

In de leercurve voor React zegt hij dat het weinig abstracties gebruikt, maar om de moeilijkste dingen te kunnen in React zal je toch voldoende tijd erin moeten stoppen. Angular vraagt wel meer tijd om het te snappen, je moet namelijk MVC, typescript,… leren. Angular zelf is ook een groot stuk dat meer tijd neemt om te leren.

Schaalbaarheid is voor Angular heel makkelijk door de krachtige CLI. Bij React is dit iets minder maar het is nog altijd goed testbaar.

Op vlak van de derde partij is React heel sterk, hoewel het niet werkt met DOM is het wel pure JavaScript. Maar zelfs de DOM libraries hebben hun alternatief in React. Angular zou het beter hebben gedaan als ze geen Typescript zouden gebruiken. Alle velden moeten namelijk een type hebben voor ze gecompileerd kunnen worden.

Voor bedrijven zou Angular een populaire keuze moeten zijn omdat het een gratis open source licentie heeft en het wordt ondersteund door Google. Met React moest je vroeger patentclausule betalen en dit was een groot probleem voor bedrijven. Recent zijn ze wel veranderd naar MIT.

Op het vlak van simpelheid en lengte van de code is Angular niet echt simpel. Het is meestal complex en er ontstaat vaak verwarring met derde partijen. React is redelijk makkelijk om te begrijpen maar het neemt meer tijd in beslag om een project op te zetten.

In zijn voorspellingen geeft hij ook nog snel mee wat we kunnen verwachten van beide frameworks. Angular zal proberen mensen te overtuigen om het te gebruiken en dit zal in zijn ogen niet goed uitpakken. React zal fiber introduceren.

3.4.4 Bron 3: Angular vs. React vs. Vue: A 2017 comparison

Uit deze bron baseren we ons op de mening van Jens Neuhaus hij heeft hier een vergelijking gegeven van het jaar 2017.

Angular en React zijn allebei ondersteund door grote bedrijven. Maar laten we eens kijken naar hun statistieken. Angular bevat 36 mensen op hun team pagina terwijl React er geen heeft. React heeft op Github wel meer dan 1000 medewerkers en meer dan 70000 sterren terwijl Angular over net geen 500 medewerkers beschikt en ook enkel 25000 sterren heeft. Zo kan je perfect zien wat vandaag de dag het populairste framework is.

Angular heeft dependency injection, dit is iets wat React niet heeft. Dit is een patroon waarin een object de afhankelijkheden levert aan een ander object. Dit leidt tot minder code die beter leesbaar is. Ook werkt Angular met MVC, React heeft enkel V, je moet M en C op je eigen zien op te lossen.

Angular is vrij opgeblazen. Als het bestand gecomprimeerd wordt is het nog altijd 143kb groot. Vergeleken met React, wat maar 43kb groot is, zal de laadtijd voor Angular veel langer zijn. React heeft Virtual DOM, wat ook nog eens de prestaties moet verbeteren.

Er is geen steile leercurve voor Angular, het heeft uitgebreide documentatie, maar soms kan je frustratie voelen omdat sommige aspecten moeilijker zijn dan ze klinken. Zelfs wanneer je een diepgaand inzicht hebt in javascript, moet je toch leren wat er gebeurt onder de kap van het kader. De setup zelf is vrij magisch en biedt veel pakketten. Dit kan negatief gezien worden omdat het zo groot is.

Met React zal je waarschijnlijk veel beslissingen moeten nemen met betrekking tot bibliotheken van derden. Er bestaan 16 verschillende fluxpakketten voor staatsbeheer om uit te kiezen in React alleen.

3.4.5 Persoonlijke reflectie

Mijn reflectie over deze literatuurstudie is dat ze min of meer elkaar wel gelijk geven en aanvullen. Enkel op de leercurve zijn kleine verschillen. Dit is in mijn ogen wel normaal omdat dit meningen zijn van verschillende personen die andere ervaringen hebben in het programmeerleven. Omdat ik zelf nog React moet gaan leren zou ik volgens deze bronnen een betere ervaring moeten hebben met React dan met Angular, dus ben ik benieuwd of dit uiteindelijk waar gaat zijn. Tot nu toe heb ik enkel positieve dingen te zeggen over Angular.

3.5 Conclusie

Wat er geconcludeerd kan worden uit alle vergelijkingen en meetresultaten is dat React een betere indruk achterlaat dan Angular. Op het vlak van de setup doen ze het beide zeer goed. Angular heeft zijn eigen CLI gemaakt en React heeft een eigen commando ontworpen wat het opzetten van beide projecten kinderspel maakt. Op vlak van de leercurve kan altijd gediscussieerd worden. Mocht je een backend-*developer* zijn die het MVC principe helemaal onder de knie heeft dan zal je Angular sneller onder de knie hebben. Daarentegen, als je enkel JavaScript kennis hebt, dan zal je React een stuk sneller kunnen leren. De leercurve van Angular is dan iets te vlak omdat Angular nog veel meer aspecten zoals MVC en Typescript heeft om te leren voordat je effectief kan beginnen met coderen. Op het vlak van documentatie en support doet React het ook beter. Procentueel doen ze het op Stackoverflow ongeveer evengoed, beide websites zijn ook goed gedocumenteerd. Maar de support van React op GitHub is een stuk sterker dan die van Angular. Dit heeft ook te maken met het feit dat grote bedrijven zoals Facebook en Netflix dit gebruiken, waaruit je kan afleiden dat dit framework niet snel zal wegvallen.

Op het vlak van de benchmark resultaten heeft Angular dan weer de bovenhand. Er kan geconcludeerd worden dat er gemiddeld duidelijk betere prestaties gehaald worden door het Angular framework. Hoewel React dan wel weer een heel stuk minder zwaar is dan Angular.

Voor JIDOKA is het nu interessant om te weten dat ze vanaf het begin de goede keuze hebben gemaakt door het React framework te gebruiken. Angular is zeker ook geen slecht framework, maar de keuze voor React is slim omdat JIDOKA direct schoolverlaters toelaat in hun bedrijf. Het is een framework dat makkelijk te leren is en verder ook sterk onderhouden wordt.

In dit onderzoek is nu de vergelijking gemaakt tussen React en Angular. Er zijn buiten deze frameworks nog opkomende JavaScript-frameworks zoals Vue.js waarmee een vergelijking mee kan gemaakt worden. Dit zou nog een interessante aanvulling zijn op dit onderzoek.

3.6 Persoonlijke reflectie

Vanaf het begin wist ik wel dat React en Angular enkele van de meest gebruikte frameworks waren in het ontwikkelen van een webapplicatie. Doorheen mijn stage heb ik gewerkt met Angular en ben ik er heel tevreden over geweest. Ik heb veel gelezen over Angular dat bedrijven erop hadden ingezet toen het nog AngularJs heette. Deze bedrijven hebben er toen heel veel spijt van gehad omdat ze niet konden updaten naar Angular2 en hogere versies. Hierop hebben ze toen goed gereageerd. Vanaf 2 is het dus ook helemaal backwards compatibel. Omdat er *Material UI* moest gebruikt worden heb ik moeten updaten van Angular 4 naar 5. Er stond een documentatie van alle klassen die verouderd waren en met wat ik ze moest vervangen. Dit was ongeveer een half uurtje werk en alles werkte weer zoals het moest. Ik ben eerlijk gezegd ook meer een man van de backend. Uit mijn onderzoek heb ik geleerd dat ik redelijk snel Angular zou snappen. Dit was ook wel het geval, na een dikke week had ik het volledig onder de knie.

Jidoka heeft me laten kijken naar een andere stageopdracht die gemaakt was in React. Ik heb hierbij ook een tijdje moeten bekijken hoe het ineen zat. Maar na verloop van tijd snapte ik dit framework ook wel. De werking van beide zijn compleet anders, maar ze zijn efficiënt op hun eigen manier.

Uit de literatuurstudie kon ik afleiden dat de werkwijze van React beter is dan die van Angular. Persoonlijk snap ik hier niet heel veel van, ik vind Angular een heel duidelijk framework dat ook een heel fijne programmeertaal gebruikt, namelijk Typescript. Omdat ik altijd object-georiënteerd heb geprogrammeerd is er weinig tot geen aanpassing gebeurd voor mij.

Mijn conclusie voor dit onderzoek is dat het beide goede frameworks zijn. Het jammere is dat Angular wordt afgemaakt op een fout die enkele jaren geleden is gebeurd, want er zit zeker in mijn ogen nog genoeg potentieel in dit framework om het nog beter te maken.

Bibliografie

[1] “Angular vs. React - the tie breaker”, airpair.com, 2017. [Online]. Available: https://www.airpair.com/angularjs/posts/angular-vs-react-the-tie-breaker

[2] “Angular vs React — the DEAL BREAKER”, hackernoon.com, 2017. [Online]. Available: https://hackernoon.com/angular-vs-react-the-deal-breaker-7d76c04496bc

[3] “Angular vs. React vs. Vue: A 2017 comparison”, medium.com, 2017. [Online]. Available: https://medium.com/unicorn-supplies/angular-vs-react-vs-vue-a-2017-comparison-c5c52d620176

[4] “Angular vs. React: Compilers”, medium.jonasbandi.net, 2017. [Online]. Available: https://medium.jonasbandi.net/angular-vs-react-compilers-45b279a8f571

[5] “Tutorial: Intro To React”, reactjs.org, 2017. [Online]. Available: https://reactjs.org/tutorial/tutorial.html

[6] “Angular Getting Started”, pluralsight.com, 2016. [Online]. Available: https://www.pluralsight.com/courses/angular-2-getting-started-update

[7] “How Virtual-DOM and diffing works in React”, medium.com, 2017. [Online]. Available: https://medium.com/@gethylgeorge/how-virtual-dom-and-diffing-works-in-react-6fc805f9f84e

[8] “Angular 2/5 - Custom Modal Window / Dialog Box”, jasonwatmore.com, 2017. [Online]. Available: http://jasonwatmore.com/post/2017/01/24/angular-2-custom-modal-window-dialog-box

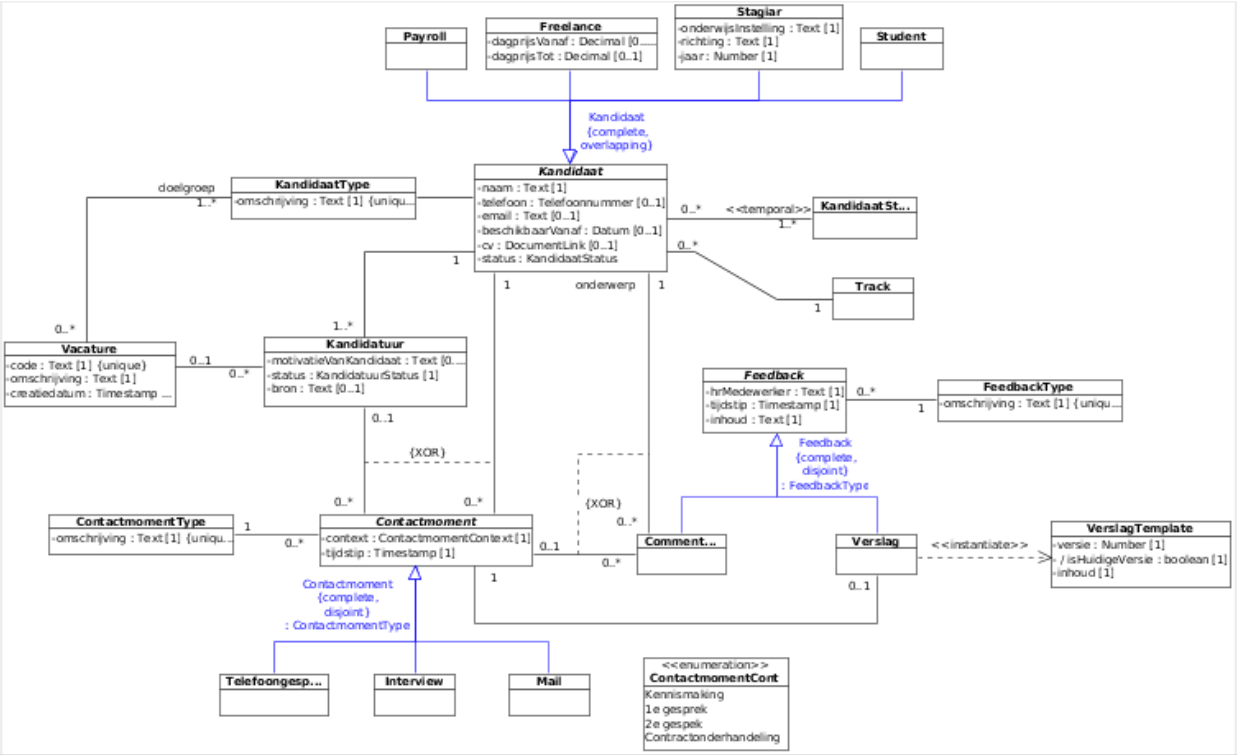
[9] “Spring Boot and OAuth2”, spring.io, 2017 [Online]. Available: https://spring.io/guides/tutorials/spring-boot-oauth2/

Bijlagen

Bijlage 1	Domeinmodel stageopdracht
Bijlage 2	UCM stageopdracht
Bijlage 3	Tabellen Oauth2.0

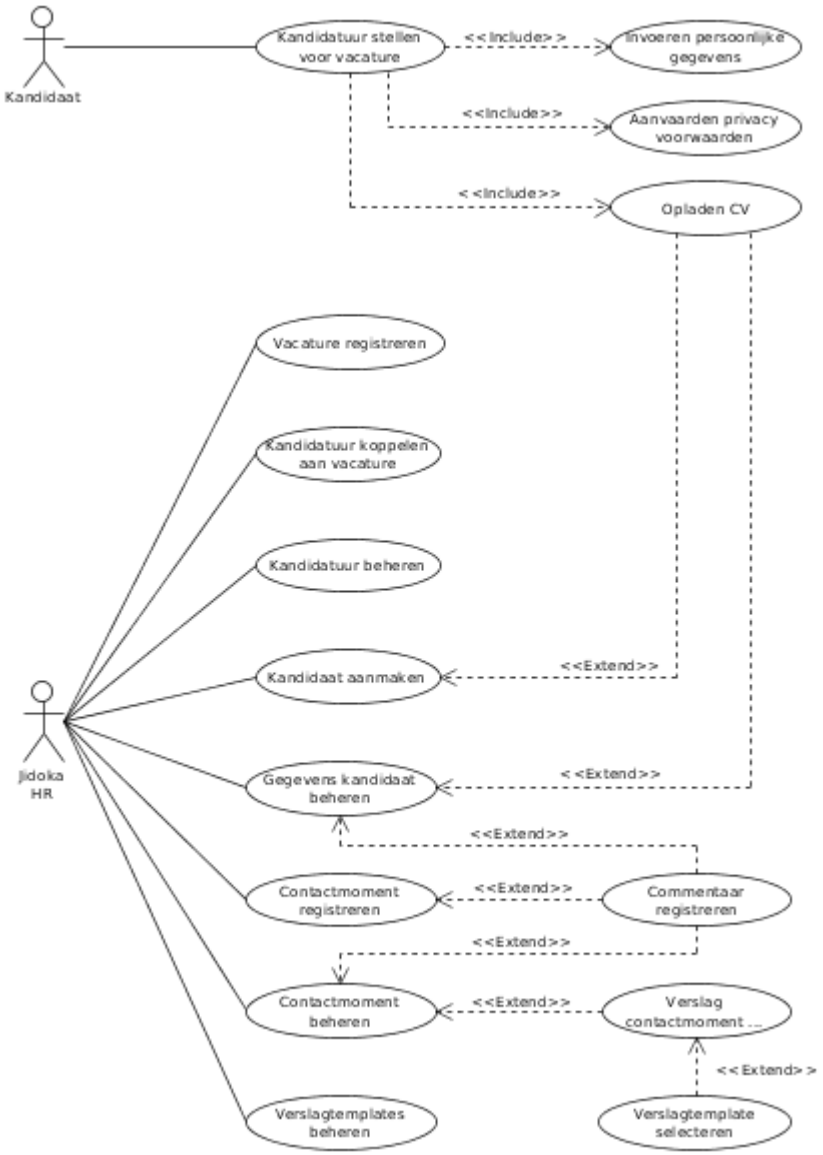
A. Bijlage 1: Domeinmodel stageopdracht

Initieel domeinmodel waaruit de stageopdracht begonnen is.



B. Bijlage 2: UCM stageopdracht

Initieel UCM waaruit de stageopdracht begonnen is.



C. Bijlage 3: Tabellen Oauth2.0

Alle tabellen die nodig zijn om Oauth2.0 te implementeren in Spring Boot

```
1.  CREATE TABLE oauth_client_details (
2.    client_id               VARCHAR(255) PRIMARY KEY,
3.    resource_ids            VARCHAR(255),
4.    client_secret           VARCHAR(255),
5.    scope                   VARCHAR(255),
6.    authorized_grant_types  VARCHAR(255),
7.    web_server_redirect_uri VARCHAR(255),
8.    authorities             VARCHAR(255),
9.    access_token_validity   INTEGER,
10.   refresh_token_validity  INTEGER,
11.   additional_information  VARCHAR(4096),
12.   autoapprove             VARCHAR(255)
13. );
14.
15. CREATE TABLE oauth_client_token (
16.   authentication_id VARCHAR(255) PRIMARY KEY,
17.   token_id          VARCHAR(255),
18.   token             bytea,
19.   user_name         VARCHAR(255),
20.   client_id         VARCHAR(255)
21. );
22.
23. CREATE TABLE client_details (
24.   appId                   VARCHAR(255) PRIMARY KEY,
25.   resourceIds             VARCHAR(255),
26.   appSecret               VARCHAR(255),
27.   scope                   VARCHAR(255),
28.   grantTypes              VARCHAR(255),
29.   redirectUrl             VARCHAR(255),
30.   authorities             VARCHAR(255),
31.   access_token_validity   INTEGER,
32.   refresh_token_validity  INTEGER,
33.   additionalInformation   VARCHAR(4096),
34.   autoApproveScopes       VARCHAR(255)
35. );
36.
37. CREATE TABLE oauth_access_token (
38.   authentication_id VARCHAR(255) PRIMARY KEY,
39.   token_id          VARCHAR(255),
40.   token             bytea,
41.   user_name         VARCHAR(255),
42.   client_id         VARCHAR(255),
43.   authentication    bytea,
44.   refresh_token     VARCHAR(255)
45. );
46.
47. CREATE TABLE oauth_refresh_token (
48.   token_id        VARCHAR(255),
49.   token           bytea,
50.   authentication  bytea
51. );
52.
53. CREATE TABLE oauth_code (
54.   code            VARCHAR(255),
55.   authentication  bytea
56. );
57.
58. CREATE TABLE oauth_approvals (
59.   userId          VARCHAR(255),
60.   clientId        VARCHAR(255),
61.   scope           VARCHAR(255),
```

```
  status          VARCHAR(10),
  expiresAt       TIMESTAMP,
  lastModifiedAt  TIMESTAMP
);

INSERT INTO oauth_client_details (client_id, access_token_validity, additional_information, authorities, authorized_grant_types, autoapprove, client_secret, refresh_token_validity, resource_ids, scope, web_server_redirect_uri)
VALUES ('recruitment-tool', 3600, null, 'ROLE_CLIENT,ROLE_TRUSTED_CLIENT', 'password,authorization_code,refresh_token,client_credentials', 'true', '**********', 2147483647, null, 'user_basic', null);
```