Risk analysis tool

For collaborations across supply chains

Afstudeerverslag

Student:	Menno Guldemond
Studentnummer:	12049530
Onderwijsinstelling:	Haagse Hogeschool | HBO-ICT Software Engineering (voltijd)
Opdrachtgever:	Almende B.V.

Referaat

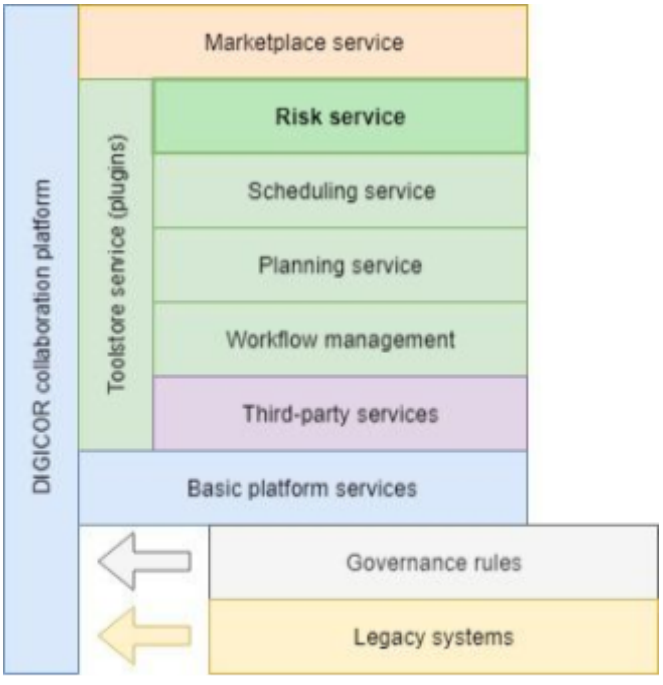
Menno Guldemond, Afstudeerverslag - Risk analysis tool for collaborations across supply chains.
HBO-ICT Software Engineering - De Haagse Hogeschool, Zoetermeer, Nederland.

Dit verslag beschrijft de werkzaamheden en resultaten in het kader van de bovengenoemde afstudeeropdracht. Ook worden de gemaakte keuzes beargumenteerd en verantwoord. De opdracht is uitgevoerd van februari 2019 t/m mei 2019.

De opdracht betreft de realisatie van software waarmee inzicht kan worden vergaard in de risico's met betrekking tot samenwerking binnen bevoorradingsketens als onderdeel van het DIGICOR project[1] (decentralized agile coordination Across supply chains), beschreven in hoofdstuk 3.3. Dit innovatieproject is gefinancierd door de Europese Commissie en heeft een hoog gehalte van onderzoek. Het onderdeel waaraan ik heb gewerkt is de risk service (zie figuur 1). Het project nadert nu het einde van de ontwikkelingsfase, maar heeft (nog) geen exploitatieplan.

Het werk bestond uit het ontwerp en de implementatie van een front-end applicatie, gemaakt met Angular en back-end software, gemaakt met Java en het Spring framework.

De gestelde projectdoelen inclusief de oplevering van bovengenoemde producten zijn binnen de stage gerealiseerd.



Figuur 1: Schematische weergave risk service binnen het DIGICOR platform.

[1] https://www.digicor-project.eu/

Samenvatting

Binnen een periode van vier maanden heb ik bij Almende B.V. software gerealiseerd als onderdeel van het DIGICOR project. DIGICOR is een platform waarin partijen binnen supply chains kunnen samenwerken om zo de efficiëntie te verbeteren, kosten te besparen, risico's te verminderen en innovatie te stimuleren. Mijn opdracht binnen dit project is bestemd voor de risicomodule. Dit is een module die verschillende risico's kan berekenen voor bijvoorbeeld planning, budgetten of verschillende belangen in de context van samenwerking met een andere partij.

DIGICOR is een Europees gesubsidieerd project waaraan verschillende bedrijven en universiteiten meewerken. Deze internationale samenwerking zorgde voor een leerzaam maar ook communicatief uitdagend project. De meeste partijen die een bijdrage leverden aan het project waren verantwoordelijk voor één of meer eigen modules binnen het platform. Almende leverde dus het deel betreffende de samenwerking risico's.

De grootste uitdaging van dit project voor mij waren de minder specifieke requirements, nieuwe technieken, andere werkwijze en de vergevorderde staat van het project ten tijde van het starten van mijn stage. Dat laatste zorgde ervoor dat veel zaken niet meer door mij te sturen waren. De nieuwe technieken zoals bijvoorbeeld Java, Spring framework en MongoDB waren een leuke uitdaging waarbij ik veel heb kunnen leren. Ook was de rapid prototyping-achtige aanpak van Almende een interessante ervaring.

Voor het einde van mijn stage heb ik de door Almende gestelde doelen met succes behaald en kan ik terugkijken op een hele leerzame en fijne periode. Ik voelde mij onderdeel van een team waarbij er goed contact was met de andere collega's en waarbij ik het gevoel kreeg dat ik gewaardeerd werd.

Dankwoord

Voor dit project heb ik heel veel samengewerkt met mensen van Almende en andere stakeholders binnen het DIGICOR project. Ik wil allen bedanken voor de leerzame tijd en de bijdrage aan de behaalde resultaten, in het bijzonder:

Carolyn Langen (projectleider Almende)
Voor de goede begeleiding, feedback en gezellige tijd.

Andries Stam (CEO Almende)
Voor de mooie kans bij Almende en de altijd goede feedback.

Ludo StellingWerff (Senior developer Almende)
Voor de goede technische discussies, nieuwe inzichten en vele leermomenten.

Ahmad Kamal (begeleidend examinator)
Voor de fijne begeleiding en feedback op mijn werk en verslag.

Alle Almende collega's
Voor alle gezelligheid en goede discussies.

Inleiding

De afgelopen maanden heb ik stage gelopen bij Almende B.V. te Rotterdam. Tijdens mijn tijd bij Almende heb ik software gerealiseerd die inzicht moet geven in risico's met betrekking tot de samenwerking binnen bevoorradingsketens als onderdeel van het DIGICOR project.

Met een half jaar te gaan voor mijn afstuderen moest ik een keuze maken. Mijn afstuderen kon ik doen voor een bedrijf waar ik al twee jaar werkzaam was, of ik zou mijn afstuderen gebruiken als een mogelijkheid om mijn horizon te verbreden en nieuwe ervaringen op te doen. Wetend dat het iets risicovoller was, besloot ik om te kiezen voor het tweede, en de kans te pakken om met nieuwe technieken te werken in een andere omgeving. Zodoende besloot ik om bij Almende te gaan afstuderen en een compleet nieuw project aan te gaan.

Het project bij Almende was anders dan mijn vorige werkervaringen. Almende doet veel researchprojecten, waarbij de gewenste eindresultaten vaak veel minder vastliggen en het eindproduct nog niet altijd klaar is voor de markt. Deze onderzoeksgerichte projecten zijn iets wat ik graag eens wilde ervaren. Ik hoopte dat in een omgeving als deze meer ruimte zou zijn voor experimenten en kennisdeling, en waarbij de focus meer zou liggen op nieuwe inzichten en resultaten, dan op heel procesmatig werken. Daarnaast had ik bijna exclusief ervaringen met Microsoft technologieën (denk aan .Net core, C#, MSSQL). Tijdens dit project heb ik met veel andere technieken gewerkt. Met deze stage heb ik mijn ervaring verbreed en heb ik een mening kunnen vormen op basis van de verschillende manieren van werken.

De afgelopen twee jaar ben ik mij steeds meer gaan interesseren voor moderne front-end applicaties. Hiermee bedoel ik single page applications of progressive web apps die gebruikmaken van web-components en andere nieuwe technieken. Ik zie het als een uitdaging om steeds complexere websites te maken die stabiel en onderhoudbaar zijn en ook goede prestaties leveren. In dit project komt dit ook naar voren.

Structuur

In dit document beschrijf ik het proces dat heeft geleid tot het uiteindelijk product. Ik beargumenteer de gemaakte keuzes en beschrijf het bedrijf, de werkwijze en de context van de opdracht. Daarnaast is er een globale planning gevolgd door alle keuzes, zowel op technisch als functioneel en design vlak. Vervolgens schrijf ik over de uiteindelijke architectuur van de applicatie en enkele aspecten of technieken die ik verder wil toelichten. Het document wordt afgesloten met een beschrijving van het eindproduct en een reflectie van mijn stage inclusief de verantwoording voor het voldoen aan de verschillende beroepstaken die horen bij mijn opleiding.

Op de volgende pagina vindt u de verklarende begrippenlijst en achter in het document vindt u de verschillende bronnen en bijlagen.

Begrippenlijst

Term of afkorting	Betekenis
Agent	Een software entiteit die een andere entiteit vertegenwoordigt. In de context van mijn project is een agent een software entiteit die autonoom kan functioneren.
API	Afkorting van Application Programming Interface.
Back-end	Een term om de software te benoemen waarmee de gebruiker **geen** direct contact heeft. Vaak een aanspreekpunt voor andere software componenten, bijvoorbeeld een server applicatie.
DOM	Afkorting van Document Object Model. Een vertegenwoordiging van de structuur van bijvoorbeeld een Html document.
ES6	Afkorting van ECMAScript 6. Een standaardisatie voor JavaScript.
Front-end	Een term om de software te benoemen waarmee de gebruiker direct contact heeft d.m.v. een visuele interface.
Governance	Binnen de ict een term om regels en controle aan te duiden ter verhoging van de kwaliteit van een product.
HTML	Een “markup language” die de structuur en layout van een webpagina beschrijft.
Library / bibliotheek	Een software collectie die functionaliteit en middelen beschikbaar stelt.
Mocking	Een term binnen software-ontwikkeling die het simuleren van objecten beschrijft, meestal voor testdoeleinden.
MVP	Afkorting van Minimum Viable Product. Een product dat met zo min mogelijk inspanning wordt gemaakt en het uiteindelijke product demonstreert.
NLP	Afkorting van Natural Language Processing. Een veld van informatica en kunstmatige intelligentie met betrekking tot het verwerken van menselijke taal.
OEM	Afkorting van Original Equipment Manufacturer. Bedrijven die producten leveren als merkleverancier voor eigen eindproducten.
OTAP	Afkorting van ontwikkel, test, acceptatie, productie. Dit is een ict-paradigma waarbij verschillende omgevingen gescheiden worden.
PWA	Afkorting van Progressive Web Apps

Sentiment analyse	Een term die het analyseren van een stuk tekst beschrijft met als doel de mate van positiviteit of negativiteit te berekenen.
SPA	Afkorting van Single Page Application. Een webpagina die zich op één pagina bevindt.
SME	Afkorting van Small and Medium Enterprises. Midden- en Kleinbedrijf (MKB) in het Nederlands.
TRL	Afkorting van Technology Readiness Level. Een NASA standaard om een staat/niveau van techniek aan te geven.
UI	Afkorting van User Interface.
XPath	Dit is een query taal om door xml/html-elementen te navigeren.

Inhoudsopgave

1 Almende

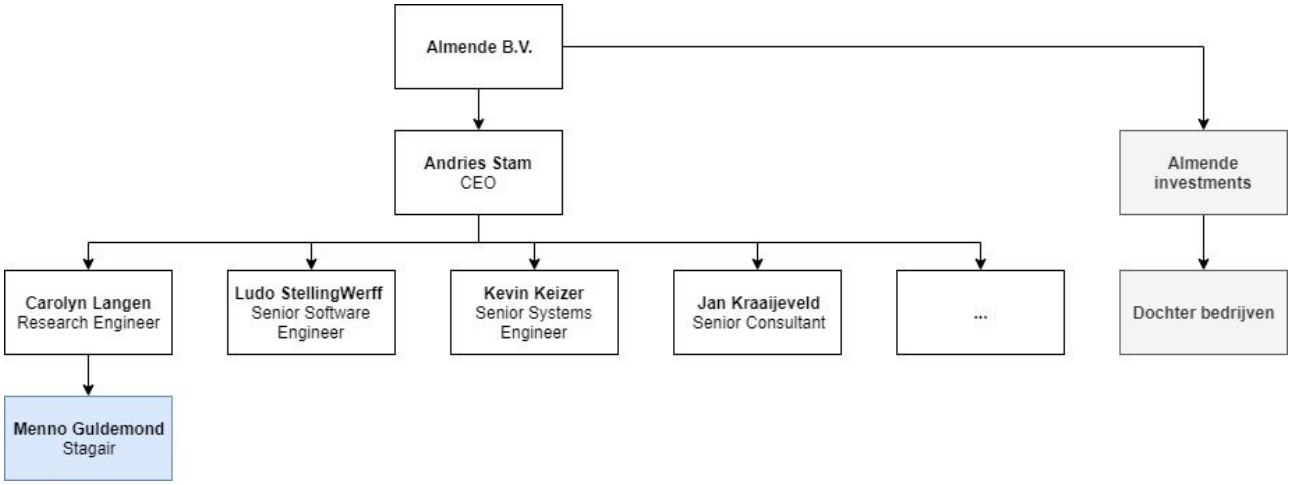
1.1 Bedrijfsachtergrond

Almende B.V. is gevestigd in Rotterdam en opgericht in 2000. Het bedrijf voert R&D-activiteiten uit in verschillende ict-domeinen, wisselend van gezondheidszorg tot productiesystemen. Bijna alle projecten hebben een element van zelforganisatie. Almende doet mee aan veel gesubsidieerde Europese en Nederlandse projecten samen met andere industriële en academische partners.

Het aantal werknemers binnen Almende fluctueert regelmatig, maar is op het moment van schrijven rond de vijftien mensen. Het team bestaat uit mensen met verschillende vaardigheden, gevolgde opleidingen (minimaal B.Sc) en afkomst. Deze fluctuatie is het resultaat van projecten die gecommercialiseerd worden in dochterbedrijven. Dit wil zeggen succesvolle projecten of projecten met veel potentie die in sommige gevallen worden ondergebracht onder een nieuw bedrijf dat het product of de dienst van het project gaan exploiteren. Op die manier beginnen één of meer medewerkers van Almende een soort start-up met dit product en krimpt het Almende team weer.

Almende heeft een vrij platte bedrijfsstructuur waar veel mogelijkheid is voor bottom-up communicatie. In figuur 2 is te zien dat boven de werknemers van Almende alleen de CEO staat. Daarnaast vallen alle dochterbedrijven onder Almende Investments, wat weer onder Almende valt.

Figuur 2: versimpeld organogram Almende B.V.



2 Werkwijze

Bij Almende wordt niet volgens een hele duidelijke werkwijze gewerkt. Tot op heden heb ik vooral volgens de SCRUM methodiek gewerkt. Bij Almende herken ik wel enkele SCRUM elementen, zoals: ongeveer om de week reflecteren op de voortgang en daarna nieuwe plannen maken, een MVP[2] maken voordat het verder uitgewerkt wordt en een soort backlog bijhouden, al bestaat dat alleen uit een tekstbestandje in de cloud. Samen met mijn begeleider bij Almende, die min of meer de rol van product owner vervult, stellen wij de wensen op in dit document. Deze kunnen geprioriteerd en gepland worden. Dit doen wij eens per week. Op deze manier hebben we dus een soort agile approach, maar we houden niet echt de regels van SCRUM aan.

Om van een idee tot een product te komen worden er normaal eerst user stories gemaakt en daarna wordt een oplossing bedacht. Deze wordt dan overlegd met de projectverantwoordelijke en/of CEO. Na goedkeuring worden eventuele aanpassingen gedaan en wordt er een MVP gemaakt. Nadat deze gereviewd is door de projectverantwoordelijke wordt het echte product uitgewerkt.

Eke dinsdag is er een conference call met alle partijen waarbij ik altijd bij aansloot. Hier werd door iedereen verteld wat zij die week hebben gedaan en wat de globale planning van de komende week is. Dit noemen ze zelf ook de “stand-up round”, een beetje zoals in SCRUM. Tevens maakt Almende gebruik van rapid prototyping principes (dit proces beschrijf ik op de volgende pagina).

Voor het samenwerken gebruiken wij het platform Samepage. Dit is een platform waar we (team) chats hebben met iedereen in het project en waar we zaken zoals productdocumentatie, user-stories, planningen en vergadernotulen kunnen beheren. Dit is enigszins vergelijkbaar met zoiets als Confluence, wat een soortgelijke, maar wat bekendere tool is.

Daarnaast wordt GitLab gebruikt voor versiebeheer. Dit is een privé git server/omgeving. Hier is ook een Continuous Integration pipeline opgezet. Bij het committen van changes wordt er automatisch een build gemaakt op de servers en worden de tests uitgevoerd van het aangepaste project. Wanneer er een build is gemaakt en de tests geslaagd zijn, kan er een deploy gedaan worden naar de productieomgeving. Er is alleen een ontwikkel- en een productieomgeving. Er is dus in principe geen test- of acceptatie-omgeving. Wel worden de eerder genoemde tests uitgevoerd en tijdens een build worden ook de stijl en opmaakregels van de code gecheckt. In het geval van falen krijgt de maker een mailtje en kan er geen deploy worden gedaan.

[2] https://en.wikipedia.org/wiki/Minimum_viable_product

2.1 Rapid prototyping

Zoals eerder genoemd maakt Almende gebruik van rapid prototyping. Dit wil zeggen dat ze zo snel mogelijk een minimale versie van het uiteindelijke product maken. Dit kan gebruikt worden om te zien of het idee realistisch en uitvoerbaar is. Vervolgens kan dit prototype dienen als basis voor het echte product, maar vaak wordt deze dan opnieuw gemaakt omdat dan zaken zoals onderhoudbaarheid en veiligheid vanaf het begin meegenomen kunnen worden in het bouwproces.

Over het algemeen ben ik hier een voorstander van, mits het goed ingezet wordt. Ik zal dit toelichten door de, naar mijn mening, positieve en negatieve punten te benoemen.

Voordelen:

- Je komt vroeg in het ontwikkelproces in aanraking met potentiële problemen en kunt daar dan nog op bijsturen of dingen herzien. Dit vermindert het risico.
- Je kunt de klant snel iets laten zien, zodat je feedback krijgt en realistische verwachtingen kunt creëren.

Nadelen:

- Het gevaar is aanwezig, dat door de snelle aanpak slordig gewerkt wordt en dat dit niet wordt aangepast/herschreven voor het uiteindelijke product.
- Rapid prototyping is lastig te doen met grote teams en/of projecten.

Over het algemeen vind ik rapid prototyping een krachtige aanpak, zolang je een geïsoleerd probleem hebt, wat niet enorm is in schaal en zich niet spreidt over verschillende domeinen en/of teams. Rapid Prototyping biedt mij de kans om een oplossing te bedenken en de problemen en uitdagingen daarbij snel te identificeren. Dit is een extra groot voordeel als iemand niet de problemen kan voorzien die zich in een later stadium voordoen door gebrek aan ervaring of kennis. Iteratief werken, ongeacht of het rapid prototyping betreft, heeft sowieso mijn voorkeur. Daarnaast vind ik het wel belangrijk om de langetermijndoelen en vereisten vrij duidelijk te hebben omdat die veel impact kunnen hebben in de vorm van wijzigingen. Wanneer ik een idee heb wat voor soort wijzigingen er ongeveer aan kunnen komen, kan ik abstracties in de software maken om mij hiertegen te beschermen.
Bij rapid prototyping kan het risico ontstaan dat men te veel bezig is met de functionaliteit van de prototypes en er te weinig visie is over wat de software moet gaan kunnen als deze volwassener wordt.

3 De opdracht

3.1 Aanleiding

Grote bedrijven, zoals de stakeholder Airbus, werken (indirect) samen met honderden bedrijven. Als er bijvoorbeeld een nieuw model vliegtuig moet worden gemaakt, worden er aanbestedingen uitgezet voor verschillende onderdelen van het vliegtuig, bijvoorbeeld voor alle apparatuur in de toiletcabine. Airbus heeft ervaring met een aantal partijen en deze sturen ze dan de aanbesteding. Het probleem hiermee is dat ze daardoor niet snel met nieuwe partijen werken, en dit is mogelijk slecht voor de innovatie, kosten en efficiëntie. De reden dat ze niet zelf nieuwe partijen zoeken is omdat dit tijdrovend is en ze niet zo goed weten wat het risico is van het samenwerken met een nieuwe partij. Dit is één van de problemen waarvoor wij een oplossing willen maken.

3.2 Probleemstelling

Het probleemdomein van deze afstudeeropdracht is supply chain en risicoanalyse. Consortiums van bedrijven willen meer inzicht in de samenwerking met nieuwe partners. Dit is onderdeel van het DIGICOR project (beschreven in hoofdstuk 3.3). Het doel van Almende voor dit project is om meer inzicht te geven in samenwerkingsrisico's, te onderzoeken hoe uitvoerbaar het is om voorspellingen te kunnen doen over samenwerkingsrisico's en de bevindingen te gebruiken in toekomstige implementaties. Samenwerkingsrisico's kunnen worden gezien in de breedste zin van het woord, bijvoorbeeld: projectvertraging, budgetproblemen, reputatieschade, conflicterende agenda's etc. Deze vormen een risico voor het project en/of het consortium die door samenwerking kunnen ontstaan.

Het DIGICOR project wordt opgeleverd op TRL 7[3]. Dit houdt in dat het product een demonstratie is van een prototype in een echte omgeving. De andere partijen uit het project refereren er ook naar als een "product delivered close to market". Er is ook nog niet een definitief exploitatieplan voor als dit project levensvatbaar wordt bevonden.

De specifieke probleemstelling kan als volgende worden geformuleerd:

1. Op dit moment heeft het DIGICOR platform niet veel data van bedrijven om risico's mee te kunnen berekenen. Almende wil daarom iets ontwikkelen om automatisch artikelen over bedrijven die op het web te vinden zijn te verzamelen en analyseren om zo bijvoorbeeld een uitspraak te kunnen doen over het sentiment dat leeft over het bedrijf.
2. Het component moet geïntegreerd worden in een event-based architectuur die verschillende bronnen van informatie ondersteunt om een risicoanalyse op te baseren. Een groot deel van deze architectuur moet nog worden ontworpen.

[3] Technology readiness level (oorspronkelijke definitie van NASA) https://www.nasa.gov/directorates/heo/scan/engineering/technology/txt_accordion1.html

3. De bevindingen (ofwel risicoanalyse uitslagen) moeten gevisualiseerd worden voor eindgebruikers in een webapplicatie door middel van een dashboard.

De probleemstelling is dus: Hoe kan de bestaande DIGICOR architectuur worden uitgebreid met een softwarecomponent voor het automatisch verzamelen en analyseren van data van het internet over bedrijven om iets te kunnen zeggen over het risico van een samenwerking met een consortium van bedrijven en hoe kunnen de resultaten geproduceerd door deze module, als deze gecombineerd is met andere beschikbare data, gevisualiseerd worden voor eindgebruikers.

3.3 Het DIGICOR project

DIGICOR is een platform dat tools en diensten aanbiedt om producerende bedrijven en dienstverleners samenwerkingsnetwerken te laten maken en beheren. Het platform zorgt dat ook kleine maar innovatieve bedrijven kunnen integreren in de complexe supply chain van grote OEM's[4] wat tot op heden minder gebruikelijk was.

Het doel is om de last van het opzetten van samenwerkingsnetwerken en de tijd om succesvolle aanbestedingen te realiseren te verminderen evenals het beheer van productie en logistieke netwerken te versimpelen.

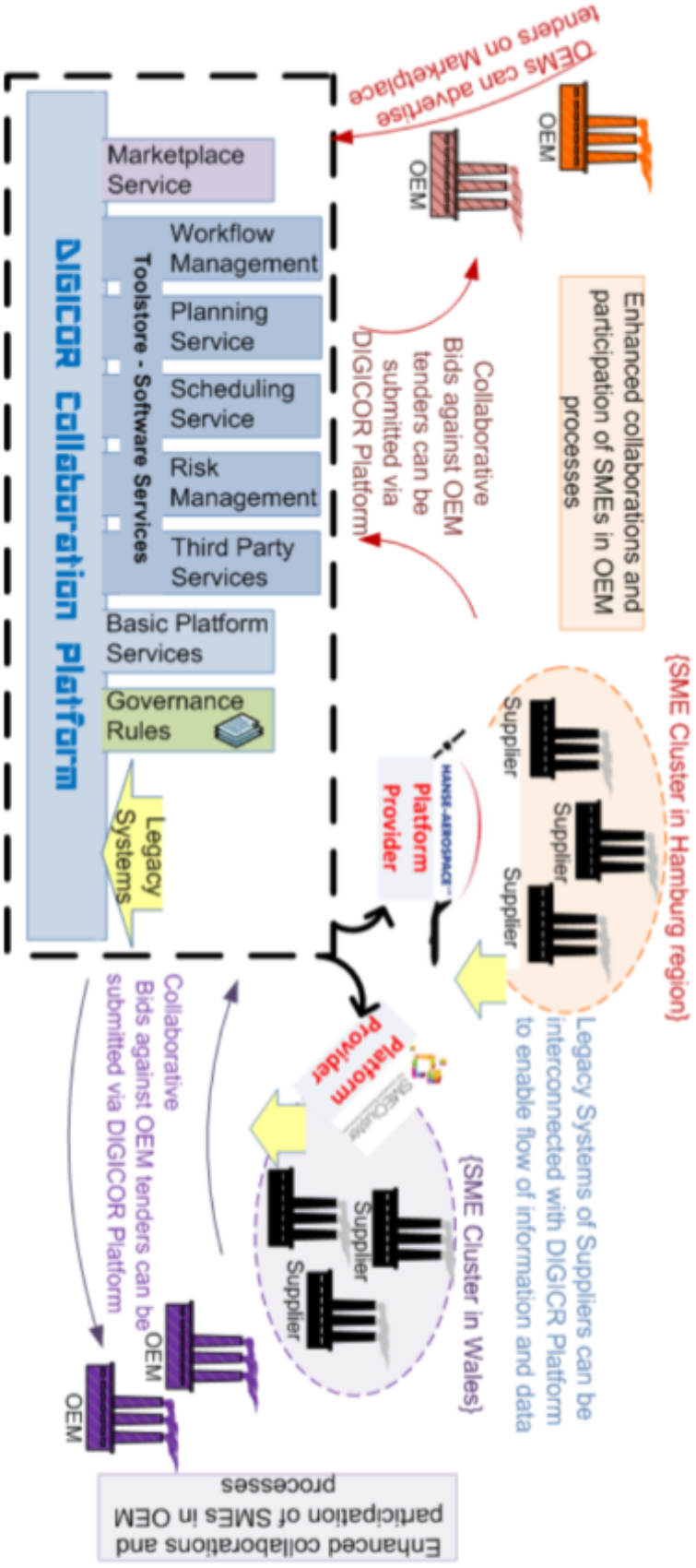
DIGICOR is een project dat onderdeel is van het Horizon 2020[5] programma. Met Horizon 2020 willen de Europese Commissie en de Nederlandse overheid wetenschap en innovatie stimuleren in het bedrijfsleven en de academische wereld. Zo kunnen zij het concurrentievermogen van Europa vergroten.

In figuur 3 op de volgende pagina is de conceptuele architectuur van het DIGICOR platform te zien.

[4] Original equipment manufacturer

[5] https://ec.europa.eu/programmes/horizon2020/

Figuur 3: Concept architectuur DIGICOR



3.4 Stakeholders

Mijn werk is zoals eerder vermeld onderdeel van het DIGICOR project. Bij dit project zijn veel partijen betrokken. Ik groepeer deze partijen in verschillende categorieën.

Opdrachtgever:
Europese Commissie, die het project financiert en het eindproduct beoordeelt.

Eindgebruikers:
- OEM's zoals Airbus en Hanse Aerospace, die het platform gebruiken om aanbestedingen uit te zetten en overzicht van de projecten te verkrijgen.
- SME's[6] die aanbestedingen zoeken op het platform en het werk daarvoor uitvoeren.

DIGICOR partners (ontwikkelen het platform):
- Airbus group (initiatiefnemer, projectcöordinator)
- Certicon (Technisch verantwoordelijke voor platform)
- Almende
- ICE
- Control 2K
- Comaus
- The University of Manchester
- Singular Logic
- Hanse Aerospace
- Czech Technical University Prague
- Fortiss

Airbus als projectcoördinator en Certicon als technisch verantwoordelijke zijn de belangrijkste stakeholders die meewerken aan de ontwikkeling van het DIGICOR platform.

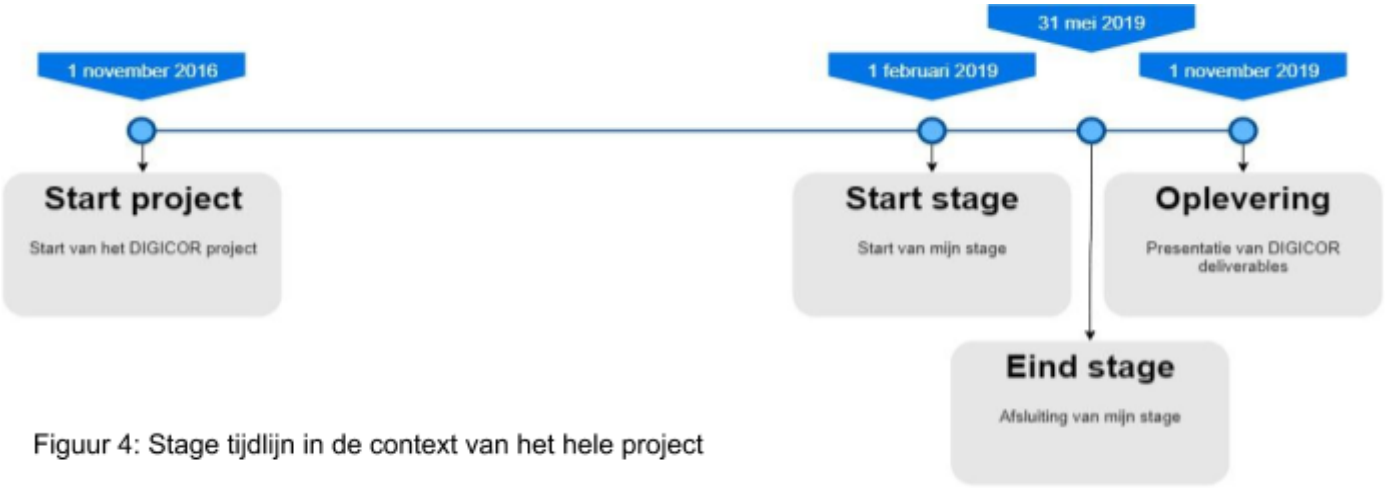
De Europese Commissie is de enige financierder van dit project. Omdat het om een "Research innovation action" gaat, zijn 100% van de kosten van het project gedekt door de EC. Daarmee zijn zij één van de belangrijkste stakeholders en beoordelen zij het eindresultaat met een review.

[6] Small and medium-sized enterprises (MKB in Nederland)

3.5 Aanvangssituatie

Vier maanden geleden, toen ik startte met dit project, was het DIGICOR project zelf al meer dan 2 jaar bezig. Aangezien het project eindigt op 1 november 2019, stroomde ik dus in, in de laatste fases van dit project (zie figuur 4).

Toen ik mijn stage startte, waren er bijna geen data om het platform mee te testen en ontwikkelen. Een van de doelstellingen was dus om data te vergaren om risicoanalyse mee te kunnen uitvoeren. Daarnaast was er vanuit Almende maar één werknemer actief op het DIGICOR project. Mijn begeleider had niet veel ervaring met REST API's en front-end applicaties, maar had wel al enkele risicorecepten uitgewerkt en modellen bedacht.
Mijn werk was om haar technisch te ondersteunen en te helpen met de realisatie van de REST-API en de front-end applicatie.



Figuur 4: Stage tijdlijn in de context van het hele project

3.6 Requirements

In het plan van aanpak zijn de requirements uitgeschreven bij aanvang van de stage.
Ze worden hier deels herhaald en de wijzigingen toegelicht.

In principe is Almende als mijn product owner leidend in de uiteindelijke keuzes voor mijn werk binnen dit project. De nadruk ligt op het succesvol opleveren van de deliverables voor het Horizon 2020 programma en het vergaren van nieuwe kennis en inzichten.

Technische requirements

1. De diensten en recepten moeten uitbreidbaar zijn voor nieuwe implementaties.
2. Web scraper moet geïsoleerd/onafhankelijk van andere onderdelen kunnen werken.
3. Integratie van services met de DIGICOR event store en het ecosysteem. Al is er door Almende en Certicon besloten om meer met REST calls te gaan doen dan via events gedurende mijn stage.
4. Het moet mogelijk zijn om een recept te gebruiken dat buiten het platform wordt gehost. (Deze requirement is later toegevoegd, in de 3e maand van de stage).

Functionele requirements

1. De gebruiker wil inzicht kunnen krijgen in het risico van samenwerken met een bepaalde partij.
2. De gebruiker wil een recept voor risicoberekening kunnen configureren.
3. De gebruiker wil kunnen inzien waaruit een risico is opgebouwd (de sub-risico's).
4. De gebruiker wil een globaal overzicht van alle risico’s voor zijn/haar bedrijf verkrijgen met een dashboard.
5. De gebruiker wil kunnen aangeven welke subrisico’s zwaarder meewegen in de berekening van een overkoepelend risico.
6. Een ontwikkelaar wil een extern recept registreren voor gebruik binnen de risk-tool (later toegevoegd, zie punt 4 van technische requirements).

3.7 Constraints

Constraints worden ook in het plan van aanpak genoemd, maar deze zijn licht gewijzigd. Met contraints bedoel ik alle beperkende factoren die effect hebben op het deel van het project waar ik aan werk.

- Alle software en documentatie moet uiterlijk 31 mei 2019 opgeleverd worden.
- Er moet gebruik worden gemaakt van de bestaande MongoDB.
- Alle software moet draaien op Amazon Web Services (AWS).
- De back-end moet met Java geschreven worden (Certicon wilt het aantal te onderhouden talen beperken).
- Front-end moet worden gemaakt met Angular (wens Certicon, de front-end kan dan makkelijk worden geïntegreerd in de bestaande Angular applicatie).

3.8 Risico's

In deze paragraaf worden de risico's bij aanvang van mijn stage beschreven, evenals projecrisico's die zich later tijdens de stage presenteerden.

- Veel data die we zouden willen gebruiken voor risicoanalyse zijn niet beschikbaar voor ons. Denk aan informatie van bedrijven over financiën, project (uitloop)tijd, etc.
- Doordat het project niet in productie wordt genomen bij oplevering en ook niet op dat niveau wordt opgeleverd, kan het lastig zijn om volledig duidelijke en concrete eisen en beperkingen te verkrijgen van stakeholders.
- Doordat er meer nadruk ligt op iets demo-waardigs in plaats van een schaalbaar en stabiel platform, worden keuzes gemaakt die de kwaliteit kunnen beïnvloeden.
- Almende heeft niet de volledige controle over de software-omgeving en technische keuzes.
- Communicatie en planning zijn lastiger doordat stakeholders uit verschillende landen afkomstig zijn.

3.9 User stories

Voor alle onderstaande user stories zijn de vereisten dat een gebruiker is ingelogd en de juiste rechten heeft om de risk tool te gebruiken. Verdere vereisten/aannames worden specifiek benoemd. Ik heb voor user stories gekozen omdat ik dit gewend ben van de scrum aanpak. User stories bieden een goede manier om duidelijk te krijgen wat de functionaliteit van het product moet gaan zijn zonder het over technische en implementatie details te hebben.

1. Een gebruiker wilt in de lijst van potentiële bedrijven voor uitvoering van een aanbesteding een *algemene* risico-indicatie zien.
    a. **ALS** een gebruiker een aanbesteding in het systeem heeft staan.
    b. **EN** er zijn potentiële bedrijven voor de uitvoering.
    c. **EN** er is een risicoscore berekend.
    d. **DAN** wil hij bij de lijst met potentiële bedrijven een risico-indicatie zien.
    e. **ANDERS** staat er in de lijst bij het bedrijf een indicatie dat er nog geen risico bekend is.
2. Een gebruiker wil een risico-overzicht via het RAS dashboard zien.
    a. **ALS** een gebruiker klikt op de risico-indicatie van een bedrijf.
    b. **EN** er risico data beschikbaar is voor dat bedrijf.
    c. **DAN** krijgt de gebruiker een dashboard te zien met een globaal overzicht van de beschikbare berekende risico's.
3. Een gebruiker wil informatie van een specifieke risicoscore inzien.
    a. **ALS** een gebruiker klikt op een risicoscore vanuit het dashboard.
    b. **DAN** krijgt de gebruiker een meer gedetailleerd overzicht te zien van het betreffende risico.
4. Een gebruiker wil weten uit welke sub-risico’s een risico is opgebouwd.
    a. **ALS** een gebruiker zich op het scherm van een specifiek risico bevindt.
    b. **EN** het betreffende risico heeft sub-risico’s.
    c. **DAN** ziet de gebruiker onder de informatie een globaal overzicht van de sub-risicoscores.
5. Een gebruiker wil informatie van een specifiek sub-risico inzien.
    a. **ALS** een gebruiker klikt op een sub-risico vanuit een risico-overzicht.
    b. **DAN** verandert het hoofdrisico van het scherm naar die van het sub-risico.
6. Een gebruiker wil vanuit een risico-overzicht naar het bovenliggende risico waar het onderdeel van is.
    a. **ALS** een gebruiker zich op het scherm van een specifiek risico bevindt.
    b. **EN** het betreffende risico is onderdeel van een algemener risico.
    c. **DAN** kan de gebruiker navigeren naar dat risico.
7. Een gebruiker wil een specifiek recept kiezen om een risicoscore te bepalen.
    a. **ALS** een gebruiker zich op het scherm van een specifiek risico bevindt.
    b. **EN** er zijn voor dat risico meerdere recepten beschikbaar.
    c. **DAN** kan de gebruiker het recept wijzigen in de gebruikersinterface.

8. Een gebruiker wil de gewichten van een samengesteld risico aanpassen.
    a. **ALS** een gebruiker zich op het scherm van een specifiek risico bevindt.
    b. **EN** het recept betreft een gewogen gemiddelde.
    c. **DAN** kan de gebruiker op de gewichtsindicatie klikken om het gewichtenformulier te openen.
    d. **DAN** kan de gebruiker gewichten aanpassen en deze opslaan.
9. Een ontwikkelaar wil zijn/haar eigen service om een recept te berekenen gebruiken in de risktool.
    a. **ALS** de ontwikkelaar een receptservice heeft die voldoet aan de regels volgens de API documentatie.
    b. **DAN** kan de ontwikkelaar via een REST call een service registreren.

4 Planning

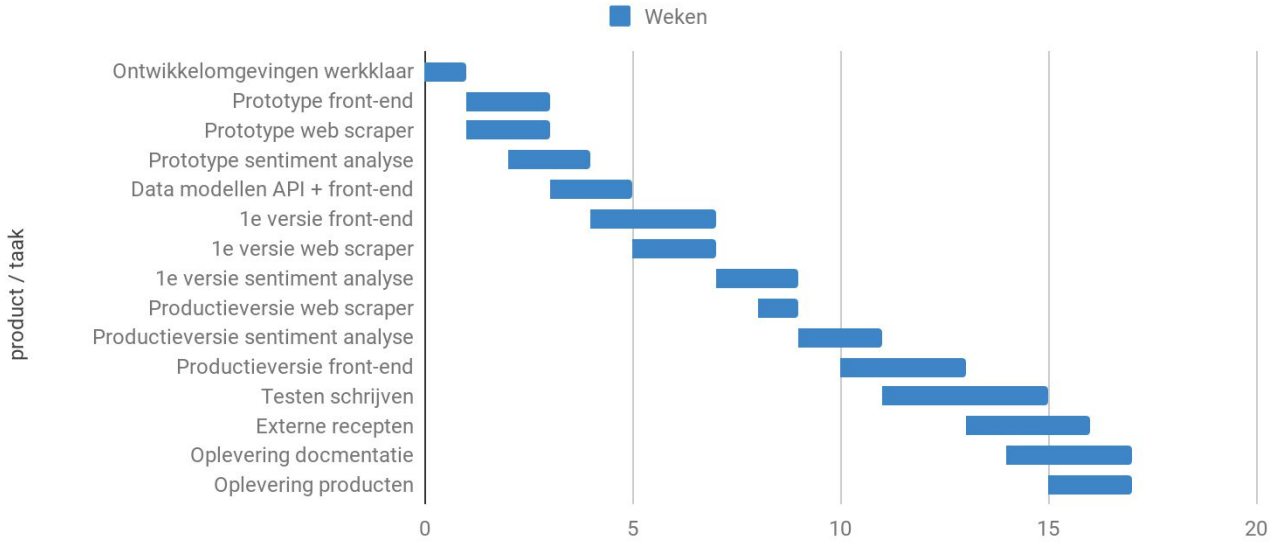
Bij aanvang van het project hebben ik samen met Almende een globale planning opgezet. De planning is gebaseerd op mijlpalen.

De oorspronkelijke mijlpalen zoals beschreven in het plan van aanpak:

1. Alle ontwikkelomgevingen werkend op mijn pc (voor week 2)
2. Prototype front-end risk visualization component (voor week 2)
3. Prototype article web-crawling (voor week 2)
4. Prototype sentiment analysis (voor week 3)
5. Integratie met bestaande DIGICOR API / event store (voor week 5)
6. Datamodel voor risico's in context van front-end en communicatie (voor week 7)
7. 1e versie front-end (voor week 8)
8. 1e versie web-crawler (voor week 8)
9. 1e versie sentimentanalysis (voor week 9)
10. Half-way-evaluatie met eventuele bijsturing of scopeverandering (voor week 10)
11. Opleveren documentatie aan Almende (voor week 17)
12. Opleveren alle producten (voor week 17)
13. Opleveren afstudeerdocument opleiding (voor week 17)

In figuur 5 is een schematische weergave te zien van de uiteindelijk uitgevoerde taken. De taken zijn omschreven in weken omdat wij na een week terugkijken op het voltooide werk. De stage duurde in totaal 17 werkweken. Het diagram geeft per taak aan in welke week deze gestart zijn en in welke deze zijn geëindigd.

Figuur 5: Tijdsweergave werkzaamheden



5 Technische keuzes

De requirements en restricties zoals beschreven in het plan van aanpak (zie bijlagen) geven mij een zekere mate van beperking en vrijheid om (technische) keuzes te maken.
In dit hoofdstuk beargumenteer ik deze keuzes en benoem ik de alternatieven.

5.1 Back-end

Bij het maken van REST-API requests heb ik ervoor gekozen om het dataverkeer zo klein mogelijk te houden. Zo voorkom ik dat er veel data worden verstuurd die onnodig zijn. Daarnaast gebeurt tegenwoordig vrij veel met een mobiele dataverbinding, die niet altijd en overal snel is. Dit is niet consistent doorgevoerd bij onze API's, maar ik wilde dit zelf in ieder geval doen om te laten zien dat ik hierover heb nagedacht.

5.1.1 Microservices / agents

Het project heeft een microservice-architectuur. De service waaraan ik heb gewerkt, heeft een eigen database waar alleen deze service mee mag communiceren. De uitvoering van sentiment analyses kost relatief veel tijd. Hierdoor is de behoefte ontstaan om deze operaties asynchroon te kunnen doen. Ik had drie voor de hand liggende opties:

1. De (bestaande) code meer asynchroon schrijven ("async - await" principe)
2. Een losse service maken voor de betreffende taken
3. Een agent based architectuur ontwerpen

Almende heeft een eigen web-agent based platform: Eve. Ik was nog niet echt bekend met een dergelijk systeem. Het leek mij een goede kans om hiermee ervaring op te doen als het ook de requirements zou ondersteunen. Het feit dat agents met een hoge mate van zelfstandigheid en in isolate kunnen werken, evenals het makkelijk kunnen wisselen van agent-implementaties, ondersteunen de gerelateerde requirements omtrent modulariteit en aanpasbaarheid zoals benoemd in het plan van aanpak. De agenten zijn ook goed schaalbaar en zorgen voor flexibiliteit.

Een losse service voor het web-scrapen zou een optie kunnen zijn, aangezien die ook een lage koppeling heeft, uitbreidbaar kan zijn en het meest geïsoleerd is van alle opties.
Het heeft echter de meeste implementatietijd, moet nog ingeregeld worden achter de API gateway en geeft nog een service om te onderhouden.

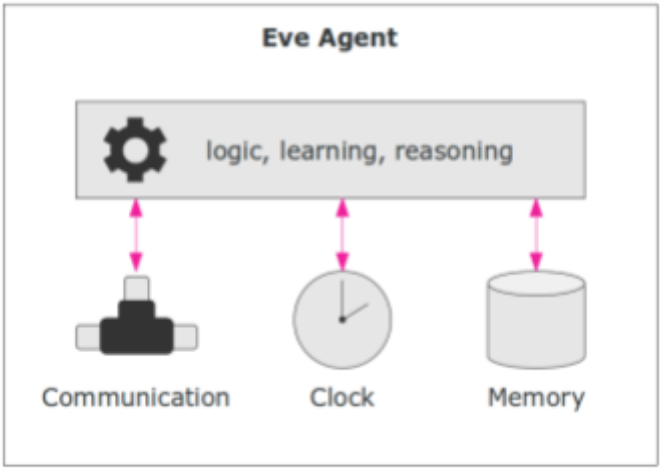
De algemene risk serivce de webscraping functies laten uitvoeren kan ook. Alleen is dit meer gekoppeld. Dit is ook eenvoudig te implementeren, maar is minder makkelijk uit te breiden en te wijzigen dan de agents.

Ik heb gekozen voor een op agents gebaseerde architectuur, omdat dit het naar mijn idee makkelijker maakt om nieuw soort gedrag aan het systeem toe te voegen. Ook is het prettig dat de agent asynchroon en in zijn eigen omgeving het werk kan doen.

Een gebruiker of process doet een verzoek tot het verzamelen van artikelen over een bedrijf, maar hoeft en kan vaak niet direct het resultaat krijgen. Er wordt alleen een reactie gegeven of de aanvraag goed is ontvangen en dat het process in werking is gesteld. Het verzamelen van artikelen met webscraping is een relatief langzame operatie. Als het process is afgerond, kunnen andere processen/agents worden aangestuurd.

De agents hebben weinig overhead en zorgen voor een goede encapsulatie.
Één soort agent kan artikelen scrapen van nieuwssites, de ander berekent reputatierisico's. Er hoeft geen afhankelijkheid te zijn tussen deze twee. Het enige dat vastligt is de manier van communiceren en de inputs en outputs van specifieke agents.
De algemene reputatierisico scores wordt niet opgeslagen. De recepten voor berekening kunnen anders zijn en dit kan continu veranderen.

Figuur 6: Agent context



De agent moet onafhankelijk kunnen opereren van de entiteit die deze vertegenwoordigt.
Zoals te zien in deze figuur, heeft een agent zijn eigen geheugen en "scheduler" en gedefinieerde manier van communiceren. De agent kan geïsoleerd en parallel aan andere processen draaien.

Overzicht van de ondersteuning van de gewenste requirements.

	Async functies in risk service	Losse "scraping" service	Agents
Low coupling	-	++	+
Korte implementatietijd	++	-	+
Goed aanpasbaar	-	+	++
Goed uitbreidbaar	+	+	+

5.1.2 Sentiment Analyse

Een onderdeel van de risicoservice is het reputatierisico berekenen. Dit is het risico omtrent de reputatie of het imago van een bedrijf. Denk hierbij bijvoorbeeld aan negatieve media-aandacht. Ik moest een service beschikbaar maken die een score zou genereren voor een bedrijf. Het doel, met het oog op een realistische planning, lag voor Almende op het hebben van data om makkelijker de risicotool verder te ontwikkelen. Almende en ik wilden reputatie scores genereren via sentimentanalyse op nieuwsartikelen. We namen daarbij voor lief dat de data mogelijk niet (altijd) een realistisch beeld zouden geven.
De belangrijkste requirements waren dus om in een korte tijd een goed beeld te kunnen krijgen of een risicorecept als dit haalbaar is en om enigszins realistische data te verkrijgen.

Sentimentanalyse wordt vaak met talen als R en Python gedaan. De meeste mensen die bij Almende werken en iets met machine-learning hebben gedaan werkten met Python. Echter was er een constraint van de stakeholder Certicon, die technisch verantwoordelijk is, om de service met Java te maken.

Er zijn drie manieren waarop vaak sentimentanalyse wordt gedaan. Deze zijn:

1. Machine learning
2. Lexicon gebaseerd (woordenlijst)
3. Hybride (combinatie van bovenstaande)

Voor het project was de eerste ingeving om een woordenlijst-principe zoals de Afinn[7] woordenlijst. Het nadeel van deze methode is dat deze geen context meeneemt in zijn beoordeling. Het is een woord-voor-woord-scoresysteem waarbij context, sarcasme, ontkennende statements e.d. verloren gaan. Dit zou voor ons projectdoel niet veel uitmaken, aangezien we vooral op zoek waren naar data, waarbij de accuratie minder belangrijk was. Uiteindelijk hebben we toch voor de machine-learning-aanpak gekozen, omdat bleek dat deze snel te realiseren was. Er was een getraind model beschikbaar van Stanford University. Dit zou ons hopelijk een wat meer realistisch beeld geven, en het gaf mij tevens een mogelijkheid om me bekend te maken met dit soort modellen. Vanwege beperkingen in tijd en scope was besloten dat een eigen model of neuraal netwerk trainen waren uitgesloten.

Overzicht ondersteuning van requirements.

	Machine learning	Lexicon gebaseerd	Hybride
Gebruikt context	+	-	+
Korte implementatietijd	+	++	-
Acuraat	+	-	++
Lerend vermogen	+	-	+

[7] http://www2.imm.dtu.dk/pubdb/views/publication_details.php?id=6010

5.2 Front-end

Voor de front-end applicatie stond vast dat dit met Angular gemaakt zou worden. Verschillende partijen binnen het DIGICOR project maken eigen modules voor de Angular applicatie. Van mij werd dit ook verwacht. Ondanks dat het framework al was bepaald, kon ik nog genoeg keuzes maken.

5.2.1 Model-driven vs template-driven forms

Bij het maken van forms (invulformulieren), die de meeste webapplicaties wel hebben, zijn er twee manieren om de data-binding (koppeling tussen de data en de gebruikersinterface) aan te pakken met Angular. De twee veel gebruikte methodes zijn template- en model-driven forms. Zoals de naam doet vermoeden wordt in het eerste geval de form gedreven door de template (ofwel Html). Dit doe je door op een form of input element een ngModel te definiëren, die verwijst naar het datamodel. Bij de tweede is de form logica onderdeel van het TypeScript component. Waarbij je in de template een ngControl definieert, die verwijst naar de control in het TypeScript component.

De keuze voor model-driven forms is vooral gemaakt, omdat ik het liefst zo weinig mogelijk logica in de Html (of view) wil hebben. Het component (de TypeScript class) is makkelijk te unit-testen. Daarnaast heeft Angular in een recente publicatie aangegeven dat ngModel mogelijk gaat vervallen in een komende release.

Overzicht van voordelen van template-driven en model-driven forms.

	Template-driven	Model-Driven
Logica niet in view	-	+
Goed uitbreidbaar	-	+
Complex gedrag makkelijker	-	+
HTML simpeler	+	-
Goed te Unit-testen	-	+

5.2.2 State management

Naar mijn mening hebben de meeste apps state management nodig.
Verschillende componenten in de app moeten vaak weten wat de staat is van bijvoorbeeld de gebruiker, de staat van de UI of van bepaalde data.

Je kunt alle data doorgeven via interactie tussen de componenten, maar dit wordt snel rommelig en zorgt er vaak voor dat componenten dingen gaan doorgeven waar ze niks mee te maken hebben. Een app waarbij alle staat door de componenten wordt bijgehouden en doorgegeven is naar mijn mening alleen iets voor prototypes en hele kleine apps.

Een andere optie is om services bepaalde variabelen te laten vasthouden. Dit was de methode die bij aanvang van het project werd gebruikt. Dit zorgt ervoor dat componenten niet ook zelf staat gedupliceerd hoeven bij te houden doordat de services dit doen. Componenten kunnen dan aan een service een stukje staat vragen en deze eventueel ook aanpassen. Deze aanpak werkt prima, maar is op grote schaal vaak lastig te beheren.

De laatste optie die het overwegen waard was, is een state management pattern zoals Redux. Bij mijn aanvang van het project werd er geen redux pattern[8] gebruikt binnen het Angular project van DIGICOR. Binnen single page applications kan het best lastig zijn om grip te hebben op alle data flows en applicatie state. Als je dit niet goed kunt managen, dan kan het gedrag de applicatie onvoorspelbaar worden. Daartegen wilde ik mijn werk beschermen.

Er zijn andere patterns die een soortgelijk doel dienen, bijvoorbeeld het “command query responsibility segregation pattern” en het “event sourcing pattern”. Beide vereisen echter een volledig eigen implementatie in tegenstelling tot redux, waarvan een goed onderhouden en gedocumenteerde bibliotheek bestaat. Daarnaast is er nog een reden waarom ik voor redux kies. Het forceert ontwikkelaars om zich aan bepaalde regels te houden, wat de leesbaarheid verbetert en het onderhouden van de applicatie makkelijker maakt.

Overzicht van de voordelen van de verschillende soorten aanpak.

	Component interactie	Services met state	Redux
Simpel	+	+	-
Scheiding state-changes en bijeffecten	-	-	+
Gestandaardiseerd	-	-	+
1 bron van waarheid	-	+	++
Performance	-	+	+
Goed testbaar	-	+	++

[8] https://github.com/ngrx/store

6 Visueel design keuzes

Keuzes op het gebied van design en layout zijn in veel gevallen uitbesteed aan UI/UX medewerkers, maar bij Almende hebben we hier geen collega voor. Om het design toch een kans van slagen te geven probeer ik deze via een stappenplan op te stellen. Het algemene design en de kleurstelling waren al gedaan voordat ik aan het project begon. Er wordt in zekere mate een material[9] design-achtige stijl aangehouden, maar niet alle regels worden hierin nageleefd.

Mijn stappen om tot een visueel ontwerp te komen:

1. Beschrijven waarom een gebruiker naar een bepaald scherm gaat.
2. Wat wil een gebruiker op dat scherm wil doen?
3. Wat is de kerninformatie voor de gebruiker?
4. Wat zijn de dominante elementen op de pagina?
5. Met welke elementen kan de gebruiker interactie hebben?
6. Hoe en wanneer verlaat de gebruiker de pagina?

Dit zijn stappen die ik zelf heb ervaren als goede hulp om een beeld te krijgen van het doel van de view. Op deze manier kan ik de belangrijkste elementen duidelijk weergeven en structuur bieden, om het vervolgens mooi te maken en aan elkaar te knopen.

Voor het risicodashboard is het belangrijk dat er een globaal overzicht wordt getoond van alle risicofactoren voor het bedrijf van de gebruiker. De gebruiker moet in één overzicht kunnen zien welke risicofactoren er voor zijn/haar bedrijf zijn berekend en welke externe risico's er zijn. Deze requirement is in een later stadium bepaald na een meeting met mijn begeleider. De uitdaging is hier vooral om niet te veel te laten zien zodat de gebruiker het dashboard makkelijk kan begrijpen. Er komen simpele elementen die een duidelijk overzicht bieden over het geheel, waarvandaan genavigeerd kan worden naar details. Dit is mijn insteek van het risicodashboard. Er moest in ieder geval een duidelijk overzicht te zien zijn van eigen risico's en risico's van externe partijen waarmee (mogelijk) wordt samengewerkt.

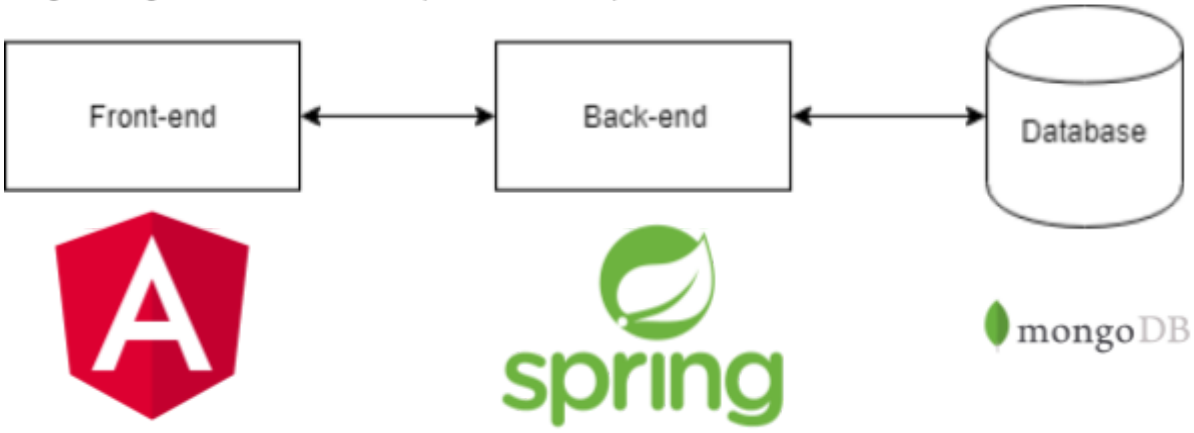
[9] https://material.io/design/

7 Technische toelichting

7.1 Algemeen

In figuur 7 zijn de gebruikte technieken weergegeven. De front-end is gemaakt met Angular en geschreven in TypeScript. De back-end is gemaakt met het Spring framework en geschreven in Java. MongoDB was de gekozen database. Al deze technieken stonden al vast bij aanvang van mijn stage.

Figuur 7: gebruikte technieken (software stack)



Van de technieken waarmee gewerkt wordt binnen dit project, heb ik alleen ervaring met Angular en in mindere mate met JavaScript binnen de browser en als Node JS server. De overige technieken zijn dus nieuw en bieden mij de mogelijkheid om nieuwe dingen te leren. Ook al lijken sommige technieken op elkaar, de nuances zijn verschillend. Vaak vind ik ook de denkwijze/insteek van een techniek/platform anders.

De afgelopen jaren op mijn studie en in mijn bijbaan heb ik vooral gewerkt met .NET. Vooral met .NET core REST API's en C# als taal in het algemeen. Als ik dat vergelijk met Java dan lijken de talen best veel op elkaar, maar de ervaring voelt voor mij heel anders. .NET voelt gestructureerder en dogmatisch[10] (het enige woord dat ik kon vinden wat dit goed beschreef). Microsoft, de beheerder van .NET, heeft een hele duidelijke mening over **hoe** je ermee zou moeten werken. Er worden duidelijke regels opgelegd op het gebied van standaarden en conventies. Het voordeel is dat code van verschillende mensen meer op elkaar zou moeten lijken. Dit verbetert de leesbaarheid en maakt daarmee onderhoud makkelijker en fouten bij wijzigingen minder waarschijnlijk.

[10] streng en onbuigbaar, niet vatbaar voor discussie.

Zelf vind ik de Java code tot nu toe iets minder leesbaar/overzichtelijk, al weet ik niet of dit komt door mijn geringe ervaring hiermee, door de gebruikte frameworks, of omdat het objectief gezien echt minder leesbaar is. Het is niet in een mate die problemen oplevert, want ik kan wel prima mijn doelen bereiken met de technieken die in dit project gebruikt worden.

Naamconventies (voor zover daar een duidelijke mondiale standaard voor is) binnen Java vind ik (lees subjectief) echt minder goed dan in talen als C#. Zoals Oracle het benoemt en hoe het in ons project gebruikt wordt, kan ik nooit in een oogopslag zien wat een variabele precies is. Ik moet altijd naar het begin van de class of het document of met mijn muis over de variabele om te zien wat het is. Ik ben van mening dat deze stap volledig overbodig is als je duidelijke naamconventies aanhoudt.

Als je dit echt duidelijk adviseert en communiceert zoals bijv. Microsoft[11], dan is alleen al met de naamconventies aan de **naam** van een variabele zien:

- Wat de scope van een variabel is.
- Wat het ontsluitingsniveau van een variabele is.
- Of het type een interface of een class betreft.
- Of de functie een query of commando betreft.

Dit bespaart in mijn ogen tijd, en nog belangrijker, kan fouten helpen voorkomen.

Verder is een specifieke eigenaardigheid bijvoorbeeld het doen van query's met de Spring library voor MongoDB. Welke query wordt uitgevoerd, wordt bepaald door de naam van de functie. De naam wordt dus geparsed[12] tot een functie, en vereist bijvoorbeeld een naam als “findOneByCompanyId”. Al is dit op zich goed te lezen, de werking is naar mijn mening veel te obscuur. Dan geef ik toch de voorkeur aan een linq statement die wordt omgezet in een SQL query zoals bij het gebruik van het Entity Framework in c#, waarbij het genoemde voorbeeld omschreven zou worden als:

DbSet.FirstOrDefault(x => x.CompanyId = id);

Dit is niet zozeer meer of minder leesbaar qua taal, maar geeft veel duidelijker aan dat dit hetgeen is wat de query bepaalt. In het eerder genoemde Java voorbeeld kan iemand zonder kennis van de werking (zoals ik bij aanvang), denken dat het puur een omschrijvende naam is. De ontwikkelaar zou dan misschien de naam van de functie anders willen noemen, om wat voor reden dan ook, om er vervolgens achter te komen dat de software niet meer werkt (zoals eerst).

[11]https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions

[12] https://en.wikipedia.org/wiki/Parsing

7.2 Scraping

Scraping is data verzamelen van informatie van webpagina's. Hier zijn verschillende manieren en frameworks voor in verschillende talen, al maken ze bijna altijd gebruik van XPath[13] om elementen te queryen. Mijn oorspronkelijke prototype om dit te doen heb ik geschreven in NodeJs met een library die Puppeteer heet. De keuze hiervoor had ik gemaakt omdat ik hiermee snel een werkend programma kon hebben, en het voor mij natuurlijk voelde om met javascript een webpagina te benaderen.

Echter, het uiteindelijke doel was om het in de Java back-end te implementeren.
Na wat uitzoekwerk kwam HtmlUnit[14] als een goede kandidaat uit de bus omdat dit een van de meest gebruikte bibliotheken was om dit soort werk te doen. Het voordeel is dat HtmlUnit een headless browser gebruikt wat zorgt voor meer snelheid.

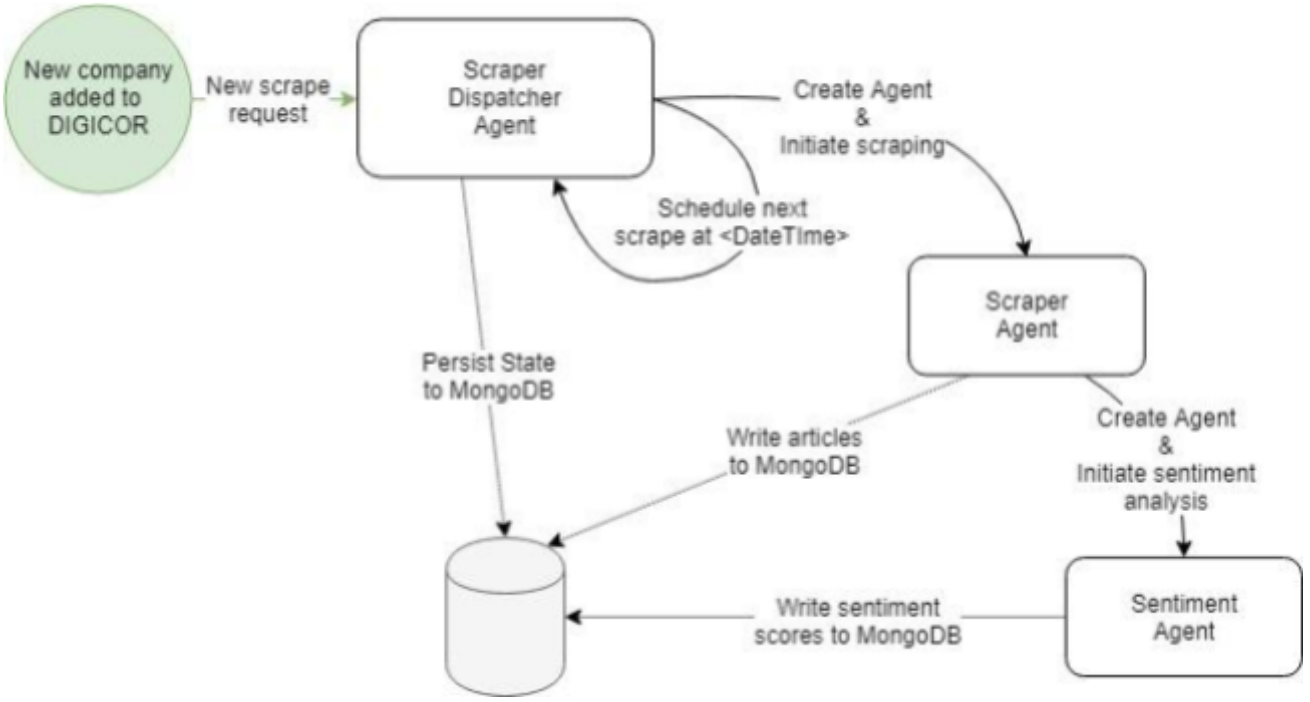
Het duurde niet lang of ik liep tegen een probleem aan, namelijk dat ik met het queryen via XPath afhankelijk ben van de structuur van de websites, en die hebben de neiging om regelmatig te veranderen. Ik gebruikte de google news site om artikelen te zoeken en dan de links naar die artikelen uit elementen te halen. Omdat google de structuur van deze links had gewijzigd kon mijn scraper agent de artikelen niet meer ophalen. Om dit systeem minder afhankelijk te maken heb ik gezocht naar een andere manier om aan deze links te komen dan het gewoon wijzigen naar de nieuwe structuur. De manier waar ik op uitkwam is om gebruik te maken van de Google Custom Search Engine. Het voordeel hiervan is dat je het kan aanspreken als een API die JSON teruggeeft. Op die manier zou ik niet meer afhankelijk zijn van de structuur van de pagina, en zou de JSON altijd de oorspronkelijke URL bevatten van het artikel waarin ik geïnteresseerd ben.

Het blijft een lastig probleem, want de site waar ik uiteindelijk uitkom (ik gebruik nu die van de BBC) kan mogelijk ook wijzigen en dan ondervind ik soortgelijke problemen. Het doel van deze module is echter om data te vergaren om onze risicotool verder uit te werken.
De overweging die ik in overleg met mijn begeleider heb gemaakt, is dat deze oplossing het doel dient en om die reden acceptabel is.

[13] https://en.wikipedia.org/wiki/XPath
[14] http://htmlunit.sourceforge.net/

In figuur 8 is een schematisch overzicht te zien van de web-scraping flow. Als er een nieuw bedrijf is, wordt via een subscription op het CompanyCreated event een verzoek gestuurd naar de ScraperDispatcherAgent. Deze agent is verantwoordelijk voor het bijhouden van de globale planning van scrapen en moet ScraperAgents aan het werk zetten. De state van de ScraperDispatcherAgent wordt gepersisteerd, zodat bij het uitvallen van deze service, deze staat na opstarten hersteld kan worden en alle scrape-acties direct weer ingepland kunnen worden. De ScraperAgents draaien geïsoleerd en zijn alleen verantwoordelijk voor het verzamelen en wegschrijven van de artikelen over een bepaald bedrijf, en heeft verder geen weet van de ScraperDispatcherAgent.

Figuur 8: Agent communicatie in context van sentiment analyse



De agent based architectuur vertoont gelijkenissen met een microservices-architectuur, maar kan op een heel ander niveau worden toegepast.

7.3 Prototypes

Voor het prototype van de webscraping service heb ik javascript gebruikt. Hierbij maakte ik gebruik van NodeJS icm met de libraries Express en Puppeteer. Express is een veel gebruikt web-applicatie framework en puppeteer is een library om webpagina's te navigeren met een (headless)chromium browser. De keuze hiervoor had ik gemaakt omdat ik snel een werkende service kon maken zonder een hele omgeving op te zetten en de gedachte was dat het in een docker container niet heel veel uitmaakt welke techniek gebruikt wordt.

De aanname dat de taal niet zou uitmaken was maar half waar. Het bleek in de praktijk lastig om met alle andere services te integreren en de partij die het serverbeheer deed wilde het liefst zo min mogelijk talen onderhouden. Het doel van het prototype was wel geslaagd. We konden informatie vergaren over bedrijven en de verkregen inzichten en methodieken waren makkelijk over te zetten naar een andere taal. Ik ben tevreden met het resultaat omdat er weinig tijd verloren is en het doel behaald werd. Volgende keer zal ik wel beter uitzoeken wat de vereisten zijn van het systeem en de omgeving bij **alle** stakeholders.

Bij het kiezen van een bibliotheek kijk ik meestal naar de mate waarin de bibliotheek onderhouden wordt en het aantal gebruikers. Ik hoop hiermee iets te kiezen wat zo lang mogelijk wordt ondersteund en waar veel informatie over beschikbaar is.

Het front-end prototype heb ik direct geschreven in Angular omdat het gebruik van Angular een vereiste was en zo veel van het werk direct verwerkt kan worden in het uiteindelijke product. Voor het prototype heb ik gewerkt met mock data om zo op een snelle manier de structuur en het uiterlijk van de front-end te kunnen maken en voor te leggen aan de stakeholders voor feedback.

Figuur 9: voorbeeld mocking front-end

```
getTeamFormationRisk(companyIds: string[]): Observable<RecipeNumber> {
    // For now, return a local object.
    return of(this.mockRecipe);
}
```

Zoals te zien is in figuur 9, heb ik de interface voor het ophalen van één van de risicodata in het prototype wel zo gemaakt dat het exact de soort interface heeft zoals het die later in het echte systeem heeft. De data van het lokale mock-data object wordt met de RxJS "of" operator als een Observable (asynchrone stream aan data) teruggegeven net zoals dat gebeurt bij een http request. Ook verwacht het al de input die het nodig zal hebben. Dit moet er in theorie voor zorgen dat op de plekken waar deze informatie gebruikt wordt, geen wijzigingen meer nodig zijn om over te gaan op echte data.

7.4 Database en modellen

De gebruikte database om de risicorecepten en -scores weg te schrijven is een MongoDB database. Deze was in gebruik voordat ik mij aansloot bij dit project. Voor het doel van het project en Almende lijkt me dit een prima keuze.

Er is nog niet heel veel zicht op wat de risicodata gaan inhouden en de gekozen database is flexibel genoeg om wijzigingen minder ingrijpend te maken. De recursieve natuur van de risicodata zou er waarschijnlijk voor zorgen dat dit gedeelte in het geval van een SQL database in een JSON veld terecht zou komen. Daarnaast ligt door de onderzoeksgerichte focus van het project de nadruk meer op een makkelijk werkbare en flexibele oplossing. Waar van toepassing moet er wel worden gekeken naar indexering op de velden van de objecten om de snelheid ook met grotere hoeveelheden data te kunnen waarborgen.

Wat ik als een voordeel ervaar na mijn werk met een MongoDB is dat een dergelijke database, die eigenlijk niet echt een strakke structuur heeft, minder beperkingen oplegt voor de ontwikkelaars die de data produceren. Er is een minder grote last om de structuur van te voren te plannen en toekomstbestendig te maken en het staat makkelijk laten evolueren van de data toe. Het nadeel dat ik hierin zie, is dat de plek waar de structuur van de data beheerd moet worden eigenlijk verschuift naar de code die de data moet lezen. De code moet om kunnen gaan met de variabele data of gewijzigd worden als de data verandert. Hierdoor verschuift de last van structuur eigenlijk van de database naar de systemen die de data consumeren. Er is een kans dat er een sterke koppeling ontstaat tussen het systeem dat de data schrijft en de systemen die de data lezen.

In principe is de Risk Tool een zelfstandige service met een eigen database. Wij verwerken wel events van het kernsysteem van DIGICOR. Elke microservice heeft in principe een eigen database. Elke service kan zijn eigen data beheren/beschermen, het enige nadeel is dat eventuele gekopieerde data gesynchroniseerd moeten worden.

7.5 Docker

Alle modules die gemaakt worden binnen het DIGICOR project draaien in een Docker container.

Docker geeft ontwikkelaars/systeembeheerders de mogelijkheid om hun code te draaien in geïsoleerde omgevingen die containers worden genoemd. Containers bundelen hun eigen applicatie, tools, libraries en configuratie. De containers kunnen gebruikmaken van dezelfde OS-kernel, waardoor ze minder middelen gebruiken dan een Virtual Machine.

Één van de grootste voordelen die ik zelf zie bij het gebruik van Docker is dat het een oplossing biedt voor het zogenaamde het-werkte-wel-op-mijn-systeem-probleem. Omdat de applicatie in zijn eigen omgeving draait, maak en test je in theorie de applicatie in de staat zoals die ook in productie is, op wat voor server dan ook.

Voordelen van Docker:

- Containers zijn een stuk kleiner dan VM's en gebruiken minder middelen.
- Containers starten snel op, wat handig is voor opschalen of voor herstarten.
- Lost het het-werkte-wel-op-mijn-systeem-probleem op.

Nadelen van Docker:

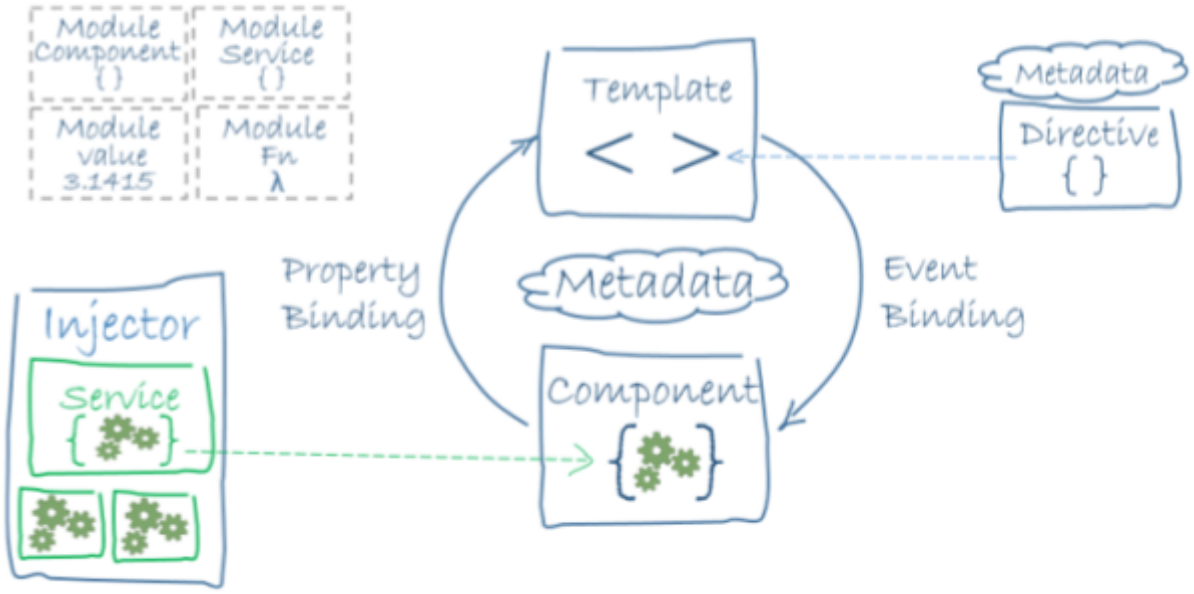
- Docker heeft alleen volledige performance op Linux. Er is support voor Windows en Mac, maar daar wordt een virtualisatielaag gebruikt tussen Docker en het OS.
- Omdat alle containers hetzelfde OS delen en root access hebben, is security een grotere uitdaging.

7.6 Angular

Angular is een open source Google framework om webapplicaties te maken. Het maakt gebruik van de Web Components[15] API om herbruikbare HTML elementen te maken met hun eigen geëncapsuleerde gedrag en uiterlijk. Angular biedt een platform voor web, mobiel en desktop. Angular is voor mij een goede keuze door zijn duidelijke en goede tooling, snelheid, performance en de community van ontwikkelaars waardoor er heel veel informatie beschikbaar is.

Een Angular applicatie is opgebouwd uit modules. Een module bevat op zijn beurt weer componenten, service providers, directives en meer bestanden die binnen de scope van die module vallen. De modules geven een Angular applicatie modulariteit. Modules kunnen dingen van andere modules importeren en zelf ook onderdelen exporteren voor het gebruik door andere modules. In figuur 10 staat een overzicht van hoe een Angular module ongeveer in elkaar zit.

Figuur 10: Architectuur van Angular (bron: angular.io/guide/architecture)



Het framework vind ik het meest geschikt voor grotere applicaties waar meer mensen aan werken. Doordat Angular heel duidelijk uitlegt hoe je dingen doet en schrijft is de code goed leesbaar voor een andere Angular programmeur. Dit ervaarde ik ook bij het DIGICOR project. Ik had vrijwel geen vragen over hoe de front-end in elkaar zat. Alles was voor de hand liggend. Doordat het framework vrij groot is en je echt op de Angular manier werkt, is de leercurve wel iets steiler dan sommige andere frameworks en kost het initieel iets meer tijd om een applicatie te maken. Dit is naar mijn mening een van de redenen waarom bijvoorbeeld React grotere gebruikersaantallen kent. Om een collega te quoten: “Met React programmeer je in JavaScript, met Angular programmeer je in Angular”. Hier zit een kern van waarheid in, ondanks dat ik Angular zelf heel prettig vind om in te werken.

[15] https://www.webcomponents.org/

Het is in Angular wel lastig om gebruik te maken van oudere javascript libraries, omdat er wordt uitgegaan van het JavaScript module systeem dat bestaat sinds ES6.

Angular maakt gebruik van de nieuwste technieken op het web en verschillende design patterns. Dependency injection, observer pattern, reactive programming zijn voorbeelden van technieken/paradigma's die je in Angular tegenkomt. Ook maakt Angular gebruik van web components, typescript en es6 features. Dit geeft je bijvoorbeeld block-scope in de javascript/typescript code, wat heel natuurlijk voelt als je van C# komt zoals ik. De herbruikbaarheid en encapsulatie die web components brengen, zorgen voor minder herhalende code en voorspelbaarder en toetsbaar gedrag. De verschillende componenten van de app zijn een stuk kleiner en daardoor duidelijker.

Het idee van de webcomponents is dat een component verantwoordelijk is voor zijn eigen scope. Een component kan op het scherm gezet worden en data gevoed krijgen, maar is verantwoordelijk voor zijn eigen uiterlijk en gedrag. Hij respecteert de ruimte die hij van zijn parent krijgt. De parent kan de publiekelijke interface van de child components gebruiken, maar weet niks van de implementatie van deze componenten. Het child component kan met het gebruik van events de parent weer van iets op de hoogte brengen.

In het geval van mijn risk module is er een risk-service die verantwoordelijk is voor de communicatie met de back-end voor het ophalen/updaten van data. De risk-service wordt geregistreerd bij de dependency injector op het risk-module niveau. Dit zorgt ervoor dat de componenten die het nodig hebben in hun constructor kunnen vragen om een instantie van deze service.

Daarnaast maak ik het onderscheid tussen slimme en domme componenten (ook wel smart en presentation components genoemd). Zoals bijvoorbeeld in dit artikel:

https://blog.angular-university.io/angular-2-smart-components-vs-presentation-components-whats-the-difference-when-to-use-each-and-why/

Dit is niet een universele visie, maar iets waarin ik mij kan vinden, omdat het helpt bij het bouwen van onderhoudbare applicaties. Domme componenten zijn vaak generiek en inzetbaar door de hele applicatie terwijl slimme componenten de echte functionaliteit van de applicatie uitdragen. Het verschil tussen deze manieren van een component maken resulteert in de volgende eigenschappen.

Slim component:

- Haalt zelf data op.
- Krijgt vaak dependencies geïnjecteerd.
- Heeft vaak domeinkennis of begrip van context.

Dom component:

- Heeft bijna altijd input en vaak ook output.
- Is zich vaak niet bewust van de context.

Ik neem als voorbeeld de Checkbox van de Material Angular bibliotheek. Dit is overduidelijk een dom (ofwel presentatie-) component. Een Checkbox component weet niks van het domein of de context. Als input kan ik misschien zeggen welk labeltje erbij moet staan, of de waarde waar of onwaar is, of de Checkbox uitgeschakeld moet zijn en of het labeltje voor of achter de Checkbox moet komen. De Checkbox produceert een event wanneer die van waarde verandert, zodat het slimme component die de Checkbox gebruikt daarop kan reageren. Hieraan zie je dat alle verantwoordelijkheid ligt bij het slimme component. De Checkbox weet alleen dat er op geklikt is en dat het dan een event moet produceren, maar het heeft geen idee wat er met dat event gebeurt. Dit zorgt ervoor dat dit component generiek inzetbaar is.

Bij het maken van componenten in (Angular) applicaties is het dus zinnig om na te denken over wat voor soort component dit is. Zo maak je, waar mogelijk, goed herbruikbare componenten die niets hoeven te weten van de context waarin ze gebruikt worden. Dit houdt de applicatie onderhoudbaar.

7.7 Reactive programming en RxJS

Omdat ik de laatste tijd binnen en buiten mijn studie veel bezig ben geweest met moderne webapplicaties en omdat ik het een interessant onderwerp en mooie techniek vind, wil ik hier ingaan op Reactive Programming.

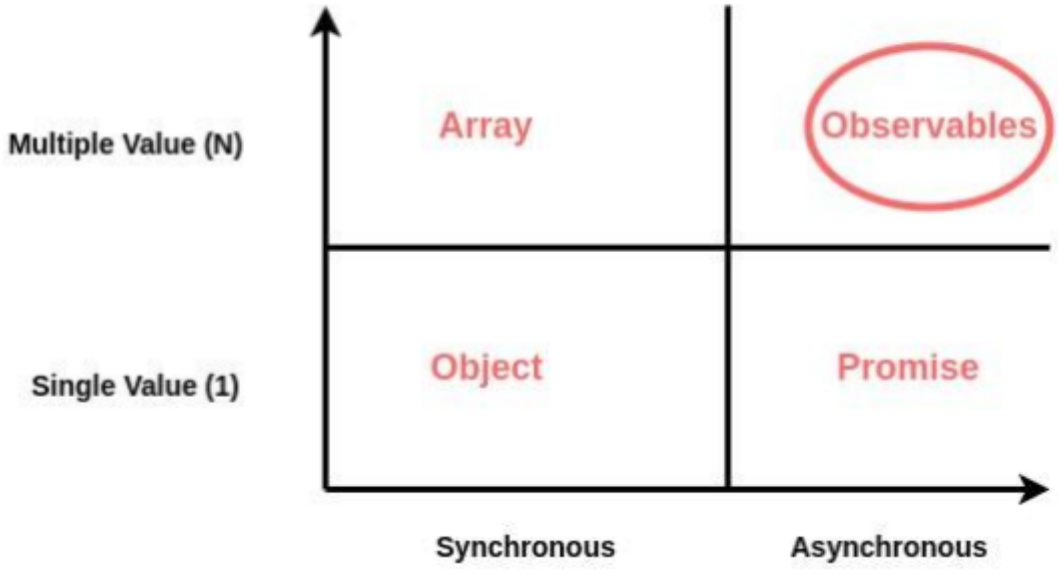
Reactive Programming is een programmeerparadigma dat bedacht is om om te gaan met asynchrone stromen(streams) van data. Het is daarmee een andere manier van programmeren waarbij de software reageert op veranderingen. Dit in tegenstelling tot procedureel programmeren waarbij je meer expliciet code schrijft om deze veranderingen af te handelen.

Je kan een stream zien als een opeenvolging van gebeurtenissen. Dit kan van alles zijn: toetsenbordinput van een gebruiker, veranderende data van een server, clicks van een muis en animaties zijn slechts enkele voorbeelden. Bij reactive programming luister je naar dergelijke streams en reageer je daarop.

Een probleem waar je met JavaScript mee te maken krijgt, is dat de enige manier (tot ES6) om een berekening te laten wachten en de rest later te laten evalueren, is om dat deel in een callback te stoppen. Één van de problemen die reactive programming wil oplossen is het zogenoemde callback hell-fenomeen. Een term die ontstaan is om de problemen te beschrijven die zich voordoen door het steeds verder nesten van dergelijke callbacks.

RxJS (Reactive Extensions for JavaScript) is een javaScript bibliotheek waarvan Angular ook gebruikmaakt. RxJS maakt gebruik van Observables. Een Observable is een functie met karakteristieken bedoeld voor het omgaan met asynchrone data streams. In figuur 11 is te zien waar het doel van Observables ligt ten opzichte van andere bekende programmeerfuncties/objecten.

Figuur 11: Context observables



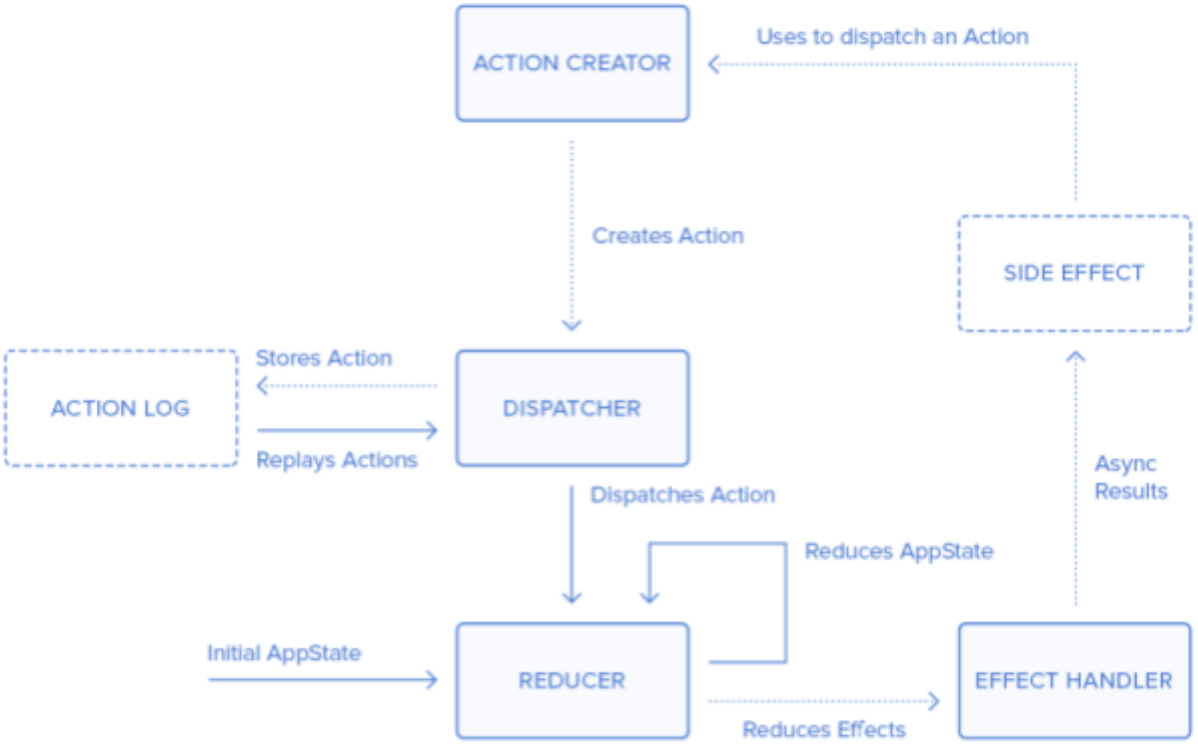
7.8 Redux

Het redux pattern is afgeleid van het Flux pattern, bedacht door Facebook. Een doel van Redux is om state changes voorspelbaar te maken. Het zorgt voor een strenge en unidirectionele stroom aan data. Zie figuur 12 voor een weergave van het design pattern .

Redux is gebaseerd op drie principes:

1. Er is maar één bron van waarheid.
2. De state is read-only.
3. Veranderingen in staat vinden alleen plaats door pure functions (functies zonder bijwerkingen en met dezelfde output bij dezelfde input).

Figuur 12: Schematische weergave van het redux pattern



Je kan redux versimpeld uitleggen door te zeggen dat het uit drie delen bestaat. Namelijk: reducers, actions en effects. Reducers zijn functies mutaties in de staat verwerken. Dit doen ze **altijd** door een nieuw state object te maken zodat er geen referentie ontstaat naar de oude state (die niet aangepast mag worden). Actions definiëren de acties die men kan uitvoeren en wat een eventuele payload van zo'n actie is. Effects zijn dingen die niet direct de staat veranderen, maar wel een effect hebben als gevolg van een actie (een soort bijeffect).

7.9 SOLID principles

Omdat ik grotendeels objectgeoriënteerd werk en bijna elke moderne taal objectgeoriënteerde principes gebruikt, zal ik de SOLID principles benoemen in dit verslag. Ook al zijn het wat oudere principes, ik vind het één van de beste richtlijnen om te gebruiken bij de bouw van een object georiënteerd systeem. Dit zijn vijf principes waar ik mij zo veel mogelijk aan probeer te houden bij het maken van software. De naam SOLID komt van de eerste letter van de vijf principes.

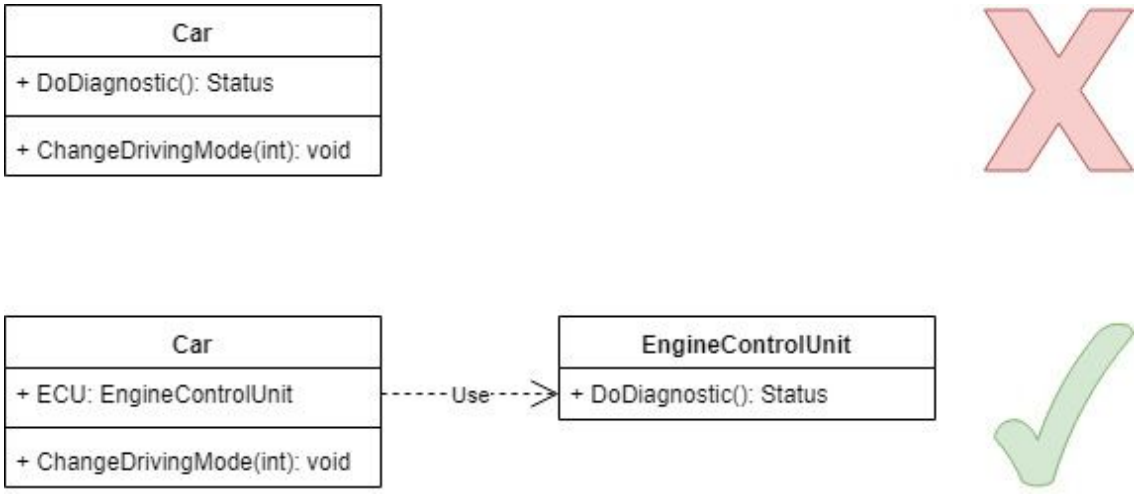
- **S**ingle responsibility principle
- **O**pen–closed principle
- **L**iskov substitution principle
- **I**nterface segregation principle
- **D**ependency inversion principle

De eerste keer dat ik van deze principes hoorde was in een YouTube video van Robert C Martin, opgenomen in de bronnen [bron 5]. Het doel van deze principes is om te helpen om stabiele, uitbreidbare en makkelijk onderhoudbare software te schrijven. Ik zal van ieder principe een korte toelichting geven van de betekenis en het doel, waarbij alle benamingen vrij vertaald zijn vanuit de oorspronkelijk Engelse definitie en naar mijn interpretatie. Ik zal de voorbeelden uitleggen in de context van één van mijn hobby's: auto’s.

7.9.1 Single responsibility principle

Een class zou maar één reden moeten hebben om te veranderen. Dit is zo omdat als een class twee of meer redenen heeft om te veranderen, deze veranderingen kunnen ontstaan om hele verschillende redenen. Het zou slecht ontwerp zijn om deze twee dingen te koppelen in dezelfde class terwijl ze een andere reden hebben om te veranderen. De verandering van de een, zou het gedrag van de tweede kunnen beïnvloeden. Het doel van dit principe is om de classes robuuster/betrouwbaarder te maken. In figuur 13 zie je dat Car twee functies heeft. Één om een diagnose te doen van de auto en de ander om de rijmodus te veranderen. Deze functies kunnen om hele andere redenen om een verandering vragen, maar de verandering van de één kan impact hebben op de werking van de ander. In het tweede voorbeeld zie je dat de diagnose functie is verhuisd naar een ECU (de boordcomputer). Deze kan niet de private fields van de Car aanpassen en daarmee is de andere functionaliteit van de Car beter beschermd tegen wijzigingen van de ECU.

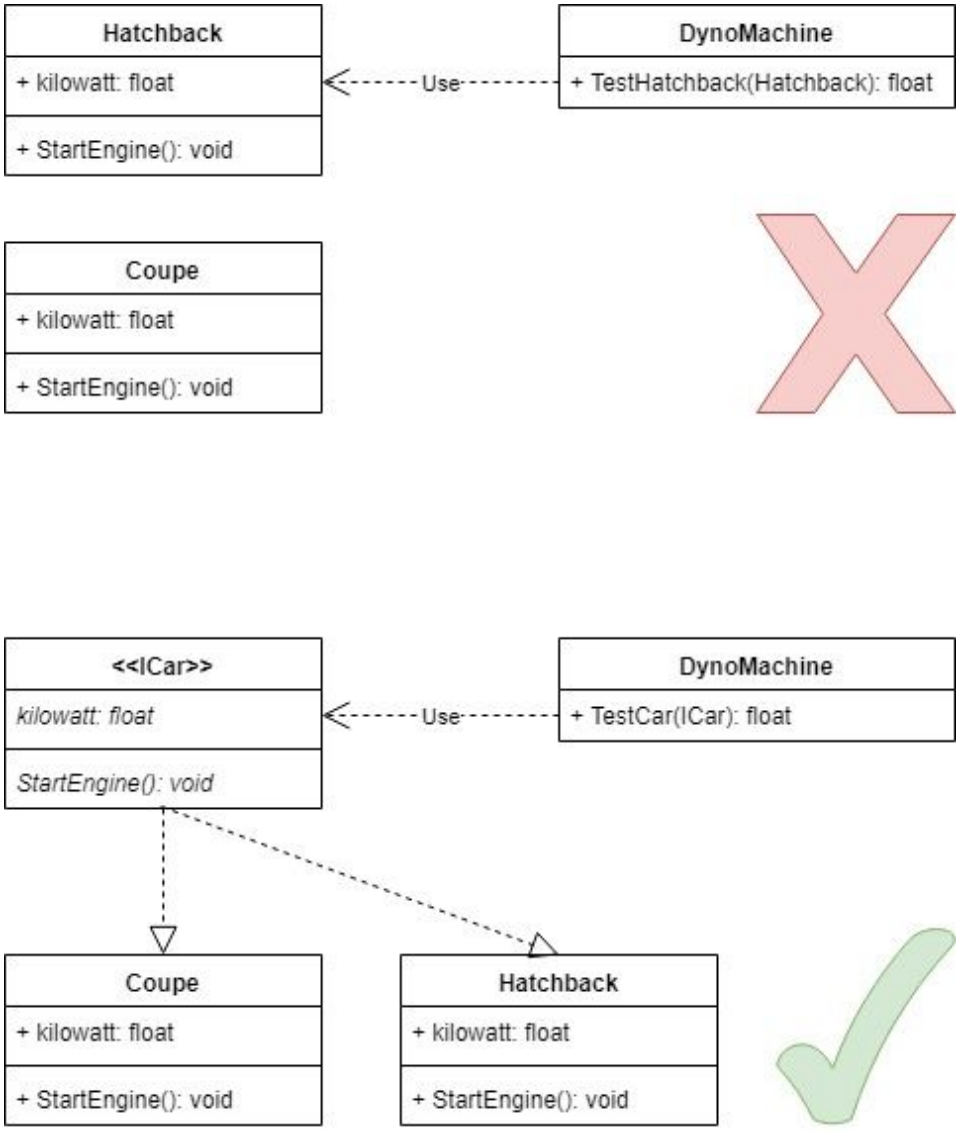
Figuur 13: Single responsibility



7.9.2 Open–closed principle

Het gedrag van een class moeten kunnen worden uitgebreid, zonder deze class aan te passen. Dit principe zou moeten zorgen voor onderhoudbare en herbruikbare code. In principe mag de source code van een class niet aangepast worden, maar het gedrag moet wel aangepast kunnen worden. De manier waarop dit kan worden bereikt is door middel van abstracties. In figuur 14 is te zien hoe een dyno machine een auto kan testen op vermogen. In het eerste voorbeeld is er een TestHatchback functie. Als de dyno machine ook een coupé moet kunnen testen, moet de DynoMachine class worden aangepast. Dit is niet volgens het Open-closed principle. In het tweede voorbeeld is te zien dat er een abstractie is gemaakt door middel van een interface (ICar). Op deze manier kan de de dyno elk type auto testen die deze interface implementeert.

Figuur 14: Open-closed

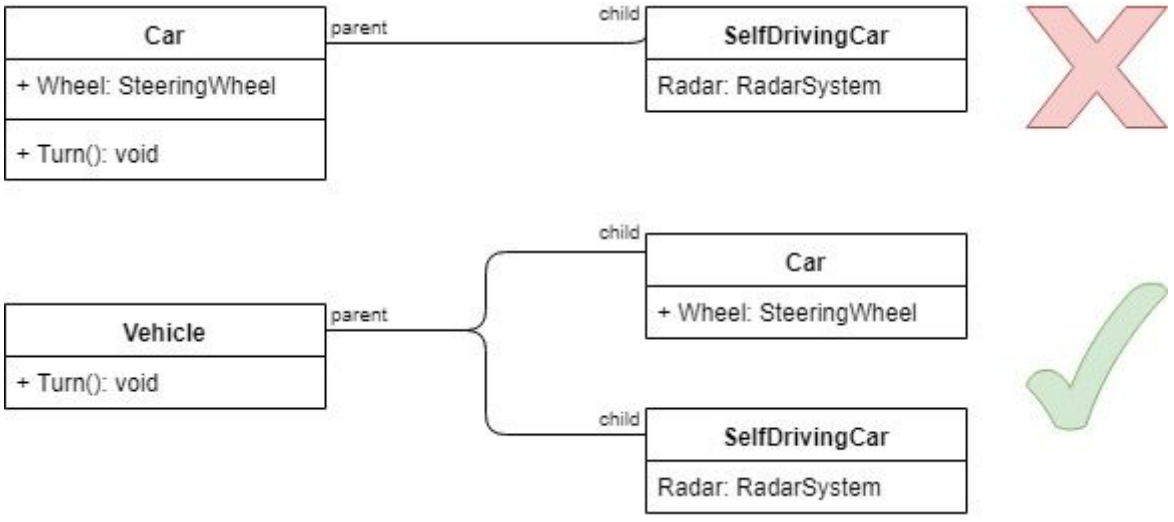


7.9.3 Liskov substitution principle

Child classes moeten verwisselbaar zijn voor hun parent class. Dit principe gaat over het goed toepassen van overerving, ofwel, wanneer is er echt sprake van een overerving relatie? In figuur 15 is een voorbeeld te zien over een zelfrijdende auto en een gewone auto. De zelfrijdende auto is een speciaal soort auto, waarmee dus gesproken kan worden van een “is-een” relatie (zelfrijdende auto is een auto), maar in dit codevoorbeeld is er totaal geen sprake van deze relatie. Een Car heeft een stuur, terwijl de zelfrijdende auto in dit voorbeeld helemaal geen stuur heeft. Stel dat er een stuk code is dat het stuur van een auto kan verwisselen en verwacht als input een Car. Dan kan het zo zijn dat deze een SelfDrivingCar krijgt, maar wat moet deze code doen als er helemaal geen stuur is. Eigenlijk is er in dit voorbeeld met een Wheel in de Car class, helemaal geen overerving relatie. Beide zouden wel kunnen erven van een class zoals Vehicle in voorbeeld twee, dat algemener is. Zoals Robert C. Martin het zegt: Omdat iets in de echte wereld een relatie heeft, wil niet zeggen dat het in code diezelfde relatie heeft.

“Representatives do not share the relations of the things they represent.”
- Robert C. Martin

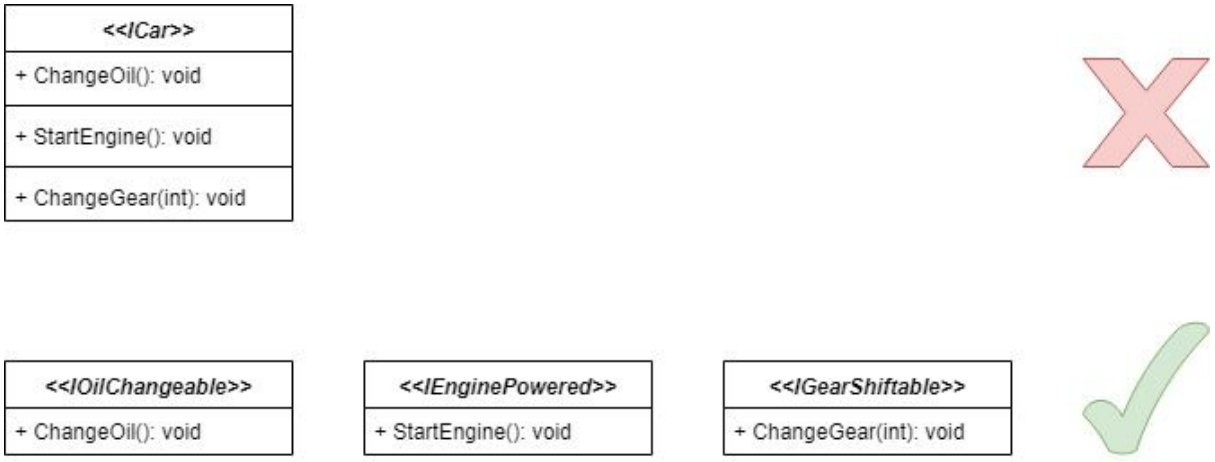
Figuur 15 Liskov substitution



7.9.4 Interface segregation principle

Meerdere specifieke interfaces zijn beter dan één algemenere interface. Het is vaak lastig om een grote interface te maken die perfect is. Daarbij is het dan vaak ook zo dat iets wat die interface implementeert, meer moet implementeren dan eigenlijk nodig is. Stel we hebben een ElectricCar class voor een elektrische auto en we willen interfaces implementeren zoals in figuur 16. In het eerste voorbeeld zouden we dan ICar implementeren, maar de elektrische auto heeft helemaal geen olie of versnellingen. We moeten dus veel meer implementeren dan nodig, al zijn het lege functies. In het tweede voorbeeld kunnen we er voor kiezen om bijvoorbeeld alleen IEnginePowered te implementeren en dan hebben we geen onnodige functies.

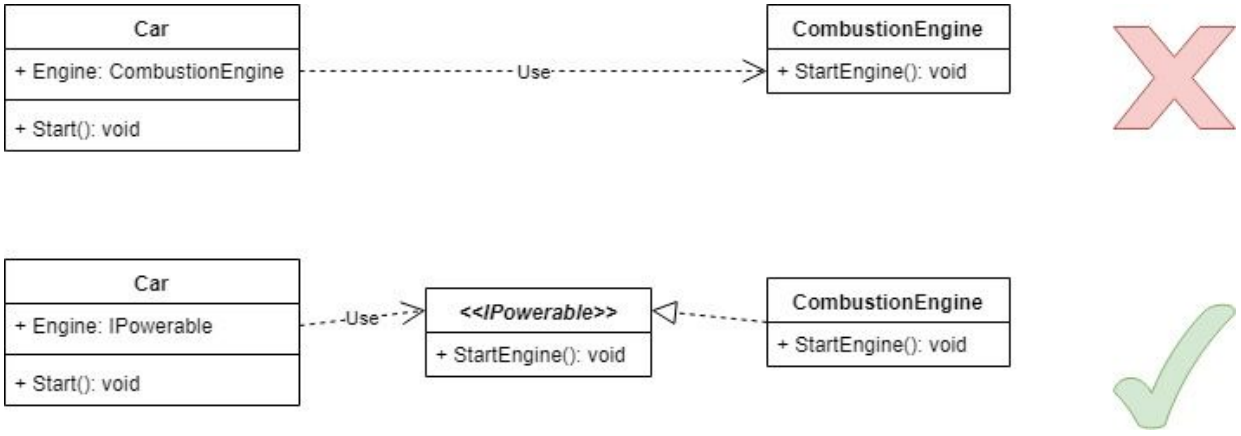
Figuur 16: Interface Segregation



7.9.5 Dependency inversion principle

Heb afhankelijkheden naar abstracties, niet naar implementaties. Het laatste principe van de SOLID principes stelt dat meer high-level classes niet afhankelijk moeten zijn van low-level classes. Zowel details als high-level classes zouden afhankelijk moeten zijn van abstracties. In figuur 17 wordt de CombustionEngine gebruikt door de Car class. Als de CombustionEngine verandert, moet de Car ook veranderen (of in ieder geval opnieuw compileren). Terwijl in het tweede voorbeeld dit niet nodig is. Het lijkt mij een gezonde situatie dat de high-level modules immuun zijn voor veranderingen aan low-level beleid. De auto is afhankelijk voor iets wat het aandrijft, maar niet op een specifieke motor. Op deze manier kunnen we de motor wisselen zonder hiervoor de source code van de Car aan te passen door een systeem als dependency injection te gebruiken.

Figuur 17: Dependency inversion

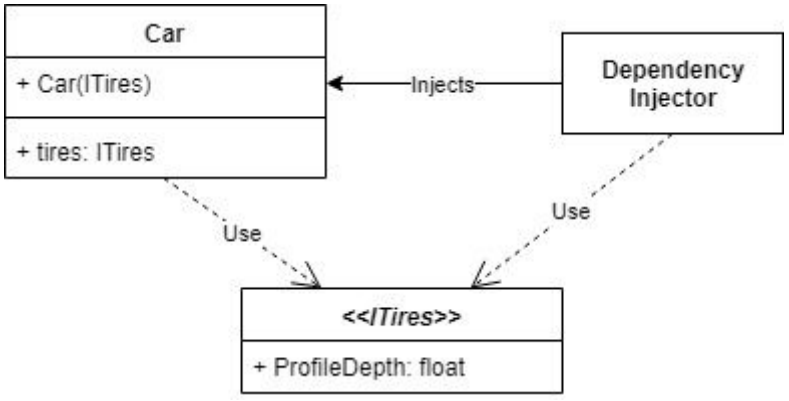


7.10 Dependency injection

In het voorgaande hoofdstuk werd dependency inversion (of inversion of control) genoemd als één van de SOLID principes. Dependency injection is een design pattern dat inversion of control gebruikt om dependencies op te lossen op een manier waarop toch lage koppeling behouden blijft. De term injection beschrijft het leveren van een dependency aan een class die deze nodig heeft. Je kan dependency injection zien als de tussenpersoon in code die al het werk doet om de objecten te creëren waar een afhankelijkheid naar is.

Om in de analogie van auto's aan te houden, als we een Car class hebben die zelf zijn banden instantieert, dan moeten we de Car class aanpassen als we willen dat die andere banden heeft. Figuur 18 is een versimpelde visualisatie van een dependency injection systeem. De car weet niet precies welke banden die krijgt, die komen mee in de constructor (in dit voorbeeld). De DependencyInjector (of de builder) creëert ergens een concrete implementatie van banden (bijvoorbeeld "new SportsTires()"). Op die manier kunnen er runtime andere banden aan de Car class geleverd worden. Het dependency injection systeem bepaalt dus welke banden de Car class krijgt en is verantwoordelijk voor het maken van deze objecten, weten wie deze nodig hebben en het doorgeven van dependencies aan deze afhankelijke classes.

Figuur 18: Dependency injection



In het bovengenoemde voorbeeld wordt de dependency meegegeven in de constructor voor de afhankelijke class. Er zijn ook andere manieren om dependency injection te doen, zoals via een setter of een specifiek geïmplementeerde interface op de afhankelijke classes. Persoonlijk vind ik het een prettige manier om injection via de constructor te doen, want dat voelt voor mij natuurlijk. Een object heeft bepaalde afhankelijkheden, en wat is een beter moment om die aan te leveren dan op het moment van creatie.

Enkele voor- en nadelen bij het gebruik van dependency injection:

Voordelen

- De applicatie uitbreiden wordt makkelijker, evenals het hergebruiken van onderdelen uit de applicatie.
- Het vermindert coupling.
- Stimuleert het maken van duidelijke interfaces, wat de afhankelijkheden duidelijker maakt.
- Unit testen wordt makkelijker gemaakt door makkelijker te wisselen implementaties naar mock-versies.

Nadelen

- Sommige errors worden pas in run-time duidelijk in plaats van tijdens compileren.
- Vereist meer types in talen zoals C#, Java, etc.
- Kan complexer zijn om te begrijpen voor ontwikkelaars, vereist hogere mate van abstractie.

8 Realisatie

Mijn werk aan het project was een aanvulling op een bestaand systeem. De front-end is volledig geïntegreerd in het bestaande project en onderdeel van dezelfde source code. De back-end heeft wel integratie, maar is een losstaand project met een eigen source code repository en database. Beide producten zijn volledig opgeleverd volgens de vooraf gestelde doelen. Ik wil ook enkele voorbeelden van code in de eindproducten laten zien en toelichten.

8.1 Front-end

Voor de front-end heb ik mijn code samengebundeld in een risk-module als onderdeel van het DIGICOR Angular webproject. Deze module biedt dashboard een detail views voor risico-inzage en geeft de gebruiker de mogelijkheid om risicorecepten te configureren. Daarnaast biedt deze module functionaliteit voor andere modules binnen de front-end applicatie omtrent risico-informatie en -visualisatie.

Naast de implementaties die ik nodig had om de gewenste functionaliteit te bouwen, heb ik ook getracht verbeteringen aan te brengen die het hele project ten goede zouden komen. Één van de verbeteringen die ik wil uitlichten is het toevoegen van een abstractie van de standaard HttpClient. In figuur 19 is een sectie hieruit te zien die dient als een wrapper om de standaard get request van de Angular HttpClient.

Mijn doel met deze code was om ongewenste dependencies weg te halen en om het doen van requests te versimpelen. De meeste requests gaan naar onze eigen back-end en vereisen autorisatie. Om deze requests te doen is er een token nodig van de AuthService. Dit zou ervoor zorgen, zoals eerst ook het geval was, dat bijna elke service een dependency heeft op de HttpClient en de AuthService. Deze AuthorizedHttpClient, zoals ik hem heb genoemd, abstraheert dit weg. Men kan door middel van mijn AuthorizedHttpClient http requests doen zonder een dependency naar de AuthService en zonder zelf deze token aan de request headers toe te moeten voegen.

Nadat ik dit in het project had geïntroduceerd en had toegelicht bij Certicon, de technisch verantwoordelijke partij, hebben zij besloten om deze service overal in het project toe te passen. Daarom ben ik erg blij met deze implementatie.

Enkele screenshots van de front-end zijn te vinden in de bijlagen.

Figuur 19: Sectie met Get request uit “authorized-http-client.service.ts”

```
    /**
     * Authorized Get
     *
     * @param url
     * @param options
     */
    get<T>(url: string, options: any = {}): Observable<T> {
        return Observable.create(observer => {
            if (!options.headers) {
                options.headers = {};
            }

            this.authService.getToken().subscribe((token: CognitoIdToken)  => {
                if (token) {
                    options.headers.Authorization = 'Bearer ' + this.authService.token.getJwtToken();

                    this.http.get<T>(url, options).subscribe(
                        response => {
                            observer.next(response);
                            observer.complete();
                        },
                        error => {
                            observer.error(error);
                            observer.complete();
                        }
                    );
                } else {
                    this.unauthorized();
                    observer.complete();
                }
            });
        });
    }
```

8.2 Back-end

Mijn onderdeel van de back-end bestaat uit de risk-tool back-end service met een eigen database. Deze stelt een API beschikbaar aan andere services en de front-end om met alle blootgestelde risicodata om te gaan. Hiernaast is er ook afhandeling van events vanuit het DIGICOR systeem en zijn er configuratiemogelijkheden. Via Swagger[16] wordt er ook API documentatie geleverd voor andere ontwikkelaars.

Ik heb als voorbeeld ook nog een externe receptenservice gemaakt die informatie van de aandelenbeurs gebruikt om een financieel risico te bepalen. Deze is te vinden in de bijlagen. Het voorbeeld laat zien hoe de risk-tool een externe service kan gebruiken als recept en hoe ontwikkelaars deze kunnen inregelen in het systeem.

[16] https://swagger.io/

Om deze externe recepten te kunnen laten uitvoeren door ons huidige systeem heb ik een ExternalRecipe class gemaakt. Dit is een implementatie van een recept die de compute functie van een recept gebruikt om via een http request een extern recept aan te spreken. De code hiervan is te zien in figuur 20. Dit was een makkelijke manier om zonder veel aanpassingen externe recepten mogelijk te maken. Deze requirement werd pas 4 weken voor het einde van mijn stage bedacht. We zijn door deze oplossing in staat om recepten toe te voegen zonder de back-end opnieuw te deployen.

Figuur 20: Compute functie van de “ExternalRecipe” class

```
public RecipeNumber compute(JsonNode params) throws JsonProcessingException {
    JsonNode jsonSetting = params.get("setting");
    JsonNode jsonCompany = params.get("company");

    try {
        ObjectMapper objectMapper = new ObjectMapper();
        RecipeSettingEntity setting = objectMapper.treeToValue(jsonSetting, RecipeSettingEntity.class);

        // Create request
        HttpClient client = HttpClientBuilder.create().build();
        HttpPost request = new HttpPost(setting.url);
        request.addHeader( name: "Content-Type",  value: "application/json");

        request.setEntity(new StringEntity(jsonCompany.toString()));

        HttpResponse response = client.execute(request);
        request.releaseConnection();

        if (response.getStatusLine().getStatusCode() == 200) {
            BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(response.getEntity().getContent()));
            StringBuilder result = new StringBuilder();
            String line = "";
            while ((line = bufferedReader.readLine()) != null) {
                result.append(line);
            }

            if (result.length() > 0) {
                return objectMapper.readValue(result.toString(), RecipeNumber.class);
            }
        }

    } catch (IOException e) {
        e.printStackTrace();

    }

    return null;
}
```

8.3 Testen

Beide producten maken gebruik van unit tests en voor de back-end zijn ook enkele integratietesten geschreven. Een aantal tests in het back-end project zijn niet door mij geschreven, maar voor de tests die ik zelf maak, houd ik altijd het Arrange-Act-Assert patroon aan. Alle testen zijn te vinden in de source code van de bijlagen.

In Angular kan je ook prima het gedrag van componenten testen. In figuur 21 is te zien hoe met een test wordt gekeken of de kleur van het risico-icoontje wel juist is met een bepaalde waarde. Eerst wordt er een waarde gezet, vervolgens forceer ik de change detection van Angular en daarna haal ik de stijl op uit het element en kijk ik of de kleur juist is. Met tests als deze kan je testen hoe een component eruitziet en welke elementen zichtbaar zijn, maar je kan ook prima interne waardes en gedrag valideren.

Figuur 21: Voorbeeld van een unit test in Angular

```
28    it('should have a red color when the value is 71 (or higher)', () => {
29      fixture.componentInstance.value = 71;
30
31      fixture.detectChanges();
32
33      const textStyle = fixture.componentInstance.getTextStyle();
34      expect(textStyle['color']).toBe('red');
35    });
```

In figuur 22 is een voorbeeld te zien van een test in de Java omgeving. Deze test maakt gebruik van een mock-database. De test in dit voorbeeld wil er in ieder geval zeker van zijn dat als er een geldige company id wordt meegegeven, er een uitslag terugkomt. Tegelijkertijd zou er bij een niet valide company id geen resultaat terug moeten komen.

Figuur 22: Test van een sentiment recipe (Java back-end)

```
@Test
public void givenValidCompanyIdGetSentimentRecipeShouldReturnRecipeNumber() throws JsonProcessingException {
    RecipeNumber recipeNumber = sentimentRecipeAgent.getSentimentRecipe( companyId: "1");

    assertThat(recipeNumber).isNotNull();
}

@Test
public void givenInvalidCompanyIdGetSentimentRecipeShouldReturnNull() throws JsonProcessingException {
    String invalidCompanyId = "0000000000";

    RecipeNumber recipeNumber = sentimentRecipeAgent.getSentimentRecipe(invalidCompanyId);

    assertThat(recipeNumber).isNull();
}
```

8.4 Documentatie

Almende zag ook graag dat ik een aantal zaken documenteerde. API documentatie genereren we automatisch met de documentatie tools van Swagger. Daarnaast heb ik user-stories opgeleverd, diagrammen getekend en algemene documentatie en demo scenario’s uitgeschreven.

In opdracht van Almende heb ik een document geschreven specifiek voor de opzet van reputatierisico's. Deze is te vinden in de bijlagen (Reputation risks documentation). Eveneens is in de bijlagen het algemene opleverdocument voor Almende (Completion document) toegevoegd.

9 Reflectie

Terugkijkend op het project heb ik een een goed gevoel overgehouden aan mijn tijd bij Almende. Het was een project met voor mij unieke uitdagingen. Zo ben ik gewend om echt onderdeel van een ontwikkelteam te zijn, waar ik hier meer op mezelf was aangewezen qua techniek. Ook het ontbreken van een echt duidelijk ontwikkelproces zoals scrum was een grote uitdaging. Het was in ieder geval een leerzame periode waarbij er veel zaken goed gingen, maar er ook verbeterpunten waren.

Goede punten:

- Contact met collega's en goede discussies.
- Resultaatgerichtheid.
- Nieuwe technieken en denkwijzen heel leerzaam.
- Open en informele sfeer.
- Mogelijkheid om iets nieuws te proberen, ruimte voor persoonlijke ontwikkeling.

Verbeterpunten:

- Ik had kleinere commits/wijzigingen moeten doen per keer.
- Meer van te voren helemaal uitdenken, ondanks de rapid prototyping aanpak.
- Standaarden/governance meer benadrukken, ondanks dat dit binnen Almende minder prioriteit heeft. (ik vind het wel belangrijk).
- Beter omgaan bij samenwerking met veel partijen die verschillende agenda's hebben.

9.1 Samenwerking

De internationale samenwerking tussen de betrokken bedrijven zorgt voor een leerzame internationale omgeving, maar bracht ook de volgende moeilijkheden met zich mee:

- Verschillende feestdagen voor elk land maakt plannen lastiger.
- Telefonisch/chat contact is niet altijd even effectief als live.
- Verschillende culturen maakt communicatie soms minder effectief.
- Voor meetings moet gevlogen worden.

Natuurlijk waren er juist ook veel positieve aspecten aan deze internationale samenwerking. Zelf vind ik vooral de verschillende culturen en achtergronden bijdragen aan een goede oplossing doordat deze vanuit verschillende oogpunten wordt benaderd. Ook was het voor mij leerzaam om een project te doen met verschillende partijen en kon ik aan mijn Engelse taalvaardigheden werken.

9.2 Verantwoording competenties

A1 Analyseren probleem domein
Voor aanvang van het project en voor de start van de bouw van (tussen)producten heb ik een overleg gehad met de opdrachtgever. Hier heb ik requirements opgesteld waarbij het probleem inzichtelijk werd en hebben we de scope bepaald. Zoals te lezen is in het hoofdstuk werkwijze, heb ik ook gekeken naar welke manier van werken goed zou passen binnen het team van Almende.

Gc Kritisch en methodisch werken
Door met mijn collega's te overleggen en discussiëren heb ik getracht om de juiste keuzes te maken voor onze doelen. Daarnaast heb ik mijn keuzes altijd geprobeerd beargumenteren tegenover stakeholders en heb ik resultaten gevalideerd tegenover de desbetreffende verantwoordelijke personen. Verder heb ik gedurende mijn stage altijd geprobeerd mee te denken over de problemen van anderen en stond ik open voor feedback op mijn eigen werk.

Gf Leren leren
Veel technieken waarmee ik heb gewerkt waren nieuw, wat een grote uitdaging was voor mij. Daarnaast heb ik ook aan mijn Engels gewerkt en meerdere presentaties gegeven. Ook op het gebied van communicatie en werken in internationale teams heb ik inzichten verkregen. Naast mijn stageopdracht heb ik veel geleerd van mijn collega's en heb ik mogen meedenken over oplossingen van andere projecten. Juist omdat mijn achtergrond zo anders is dan die van veel van mijn collega's hebben wij veel van elkaar kunnen leren.

D14 Realiseren van software
Voor dit project heb ik een single page application gebouwd met Angular. Daarnaast heb ik gebruikgemaakt van verschillende andere frameworks en heb ik ook een serverapplicatie ontwikkeld. De broncode van deze applicaties is te vinden in de meegestuurde bestanden. Verdere toelichting over de realisatie van de software is te lezen in hoofdstuk 8.

C6 Ontwerpen software
Tijdens mijn werkzaamheden aan dit project heb ik software-architecturen opgezet en deze beargumenteerd. Daarnaast heb ik verschillende design patterns toegepast, gekozen tussen oplossingsrichtingen en ontwerpdiagrammen getekend. De agent gebaseerde architectuur met de bijpassende verantwoordelijkheden is een voorbeeld van mijn ontwerpwerk. Zie figuur 8 in hoofdstuk 7.2. Ook de modulariteit en data-flow van de front-end componenten is iets waar aardig wat ontwerpwerk in heeft gezeten.

10 Conclusie

Er kan worden teruggekeken op een productieve en leerzame periode waarbij de projectdoelen succesvol zijn gerealiseerd. Qua oplevering was het doel om een webapplicatie en een back-end applicatie op te leveren en dit zijn dus ook de twee producten die door mij voor Almende zijn opgeleverd.

De grootste uitdaging voor mij tijdens de stage was het ontbreken van een echt duidelijk ontwikkelproces zoals scrum. Dit, gecombineerd met het feit dat het product nooit tot een productieniveau werd ontwikkeld, zorgde voor een minder gestructureerd project dan dat ik gewend was. Met de kennis van nu zou ik veel meer tijd besteed hebben aan de requirementsfase en de ontwerpfase.

Almende heeft wel zijn doelen kunnen halen voor het project en daar ben ik erg blij mee. Met de gemaakte doelen is er data vergaard om risicorecepten te testen en op deze manier zijn er ook meer demo scenario’s toegevoegd.

Als het project toch nog geëxploiteerd gaat worden, of als de techniek meegenomen gaat worden naar een nieuw project dan zou ik de aanpak wel iets veranderen. Ik zou meer inzetten op governance, standaardisatie en een eenduidige manier van werken met de technieken. Dat zal de onderhoudbaarheid en aanpasbaarheid van de applicaties ten goede komen.

Bronnen

Deze bronnen zijn gebruikt als naslagwerk (API documentaties) en informatiebronnen voor mijn verslag. Genoteerd volgens de IEEE[17] standaard.

1. angular.io. (2019). Angular. [online] Available at: https://angular.io/docs [Accessed Feb-May 2019].
2. digicor-project. (2019). [online] Available at: https://www.digicor-project.eu/ [Accessed Feb-May 2019].
3. Almende B.V. (2019). Eve - Introduction. [online] Eve.almende.com. Available at: https://eve.almende.com/ [Accessed Feb-May 2019].
4. stanfordnlp.github.io. (2019). Stanford CoreNLP – Natural language software | Stanford CoreNLP. [online] Available at: https://stanfordnlp.github.io/CoreNLP/ [Accessed Feb-May 2019].
5. Martin, R. (2019). Bob Martin SOLID Principles of Object Oriented and Agile Design. [online] YouTube. Available at: https://www.youtube.com/watch?v=TMuno5RZNeE [Accessed 25th April 2019].

[17] https://www.ieee.org/

Bijlagen

Enkele documenten zijn in het Engels geschreven in verband met de wens van Almende.

1. Afstudeerplan
2. Plan van aanpak
3. Adviezen voor huidige front-end
4. Documentatie reputatierisico's (Reputation risks documentation)
5. Opleverdocument (completion document) van de software voor Almende
6. Evaluatieformulier bedrijfsmentor
7. Screenshots front-end

Afstudeerplan

Informatie afstudeerder en gastbedrijf (*structuur niet wijzigen*)

Afstudeerblok: 2019-1.1 (start uiterlijk 4 februari 2019)
Startdatum uitvoering afstudeeropdracht: 1 februari 2019
Inleverdatum afstudeerdossier volgens jaarrooster: 31 mei 2019

Studentnummer: 12049530
Achternaam: dhr Guldemond
Voorletters: M.N.
Roepnaam: Menno
Adres: Minister Aalberselaan 31
Postcode: 2285EN
Woonplaats: Rijswijk
Telefoonnummer: n.v.t.
Mobiel nummer: 0640848077
Privé emailadres: mennoguldemond@live.nl

Opleiding: HBO-ICT Software engineering
Locatie: Zoetermeer
Variant: voltijd

Naam studieloopbaanbegeleider: *Nanny Jacobs*
Naam begeleidend examinator: *Ahmad Kamal*
Naam expert examinator: *Dave Stikkolorum*

Naam bedrijf: Almende B.V.
Afdeling bedrijf:
Bezoekadres bedrijf: Stationsplein 45 (unit D1. 116)
Postcode bezoekadres: 3013AK
Postbusnummer:
Postcode postbusnummer:
Plaats: Rotterdam
Telefoon bedrijf: 0104049444
Telefax bedrijf: n.v.t.
Internetsite bedrijf: www.almende.com

Achternaam opdrachtgever: dhr Stam
Voorletters opdrachtgever: A.W.
Titulatuur opdrachtgever:
Functie opdrachtgever: CEO
Doorkiesnummer opdrachtgever:
Email opdrachtgever: andries@almende.org

Achternaam bedrijfsmentor: mw Langen
Voorletters bedrijfsmentor: C.D.
Titulatuur bedrijfsmentor:
Functie bedrijfsmentor: Research Engineer
Doorkiesnummer bedrijfsmentor:
Email bedrijfsmentor: carolyn@almende.org

Doorkiesnummer afstudeerder:
Functie afstudeerder (deeltijd/duaal):

Titel afstudeeropdracht:

Ontwikkeling van een risicoanalyse webapplicatie voor een bedrijfs samenwerkings platform in opdracht van Almende B.V.

Opdrachtomschrijving

1. **Bedrijf**

Almende is een in Rotterdam gevestigd bedrijf en opgericht in 2000. Het bedrijf voert R&D activiteiten uit in verschillende ICT domeinen, wisselend van gezondheidszorg tot productie systemen. Almende is betrokken bij verschillende Europese en Nederlandse gefinancierde projecten samen met andere industriële en academische partners.

Het aantal werknemers fluctueert regelmatig, maar is op het moment van schrijven rond de vijftien mensen. Deze fluctuatie is het resultaat van projecten die gecommercialiseerd worden in dochterbedrijven. Het team bestaat uit mensen met verschillende vaardigheden, gevolgde opleidingen (minimaal B.Sc) en afkomst.

Almende's missie is om innovatieve ICT oplossingen te maken die mensen de middelen geven om hun leven beter te organiseren in een steeds complexere wereld.

2. **Probleemstelling**

Het probleemdomein van mijn opdracht is supply chain en risicoanalyse. Consortiums van bedrijven willen meer inzicht in de samenwerking met nieuwe partners. Dit is onderdeel van het H2020 DIGICOR project (decentralized agile coordination Across supply chains). Het doel van Almende voor dit project is om meer inzicht te geven in samenwerking risico's, te onderzoeken hoe uitvoerbaar het is om voorspellingen te kunnen doen over samenwerking risico's en de bevindingen te gebruiken in toekomstige implementaties.

De specifieke probleemstelling kan als volgende geformuleerd worden:

1. Op dit moment heeft het platform niet veel data van bedrijven om mee te werken. Almende wil daarom iets willen ontwikkelen om automatisch data van bedrijven die op het web te vinden is te verzamelen en analyseren.
2. Het component moet geïntegreerd worden in een event-based architectuur die verschillende bronnen van informatie ondersteund om een risicoanalyse op te baseren. Een groot deel van deze architectuur moet nog worden ontworpen.
3. De bevindingen (ofwel risicoanalyse uitslagen) moeten gevisualiseerd worden voor eindgebruikers in een webapplicatie.

De probleemstelling is dus: Op wat voor manier kan de bestaande DIGICOR architectuur uitgebreid worden met een software component voor het automatisch verzamelen en analyseren van data van het internet over bedrijven om iets te kunnen zeggen over het risico van een samenwerking met een consortium van bedrijven en hoe kunnen de resultaten geproduceerd door dit component, als deze gecombineerd is met andere beschikbare data, gevisualiseerd worden voor eindgebruikers.

De scope van het project is de technische realisatie van het nieuwe software component, architecturaal ontwerp en front-end visualisatie. Almende zal verantwoordelijk zijn voor de data analyse en aggregatie technieken.

3. Doelstelling van de afstudeeropdracht

Aan het einde van de opdracht wil ik een systeem opleveren waarmee Almende data kan verzamelen om aannames te kunnen maken over de reputatie status van bedrijven. Gecombineerd met andere informatiebronnen en toegepaste risico analyse technieken wil ik een webapplicatie opleveren waar “risico scores” worden getoond op basis van berekeningen met deze bronnen van data. Dit bevat ook de subscores van de relevante domeinen.

4. Resultaat

Wanneer opgeleverd, zal mijn product inzichten geven in samenwerking risico’s als onderdeel van het DIGICOR platform. De architectuur zal modulair ontworpen zijn zodat nieuwe databronnen makkelijk geïntegreerd kunnen worden. Alle resultaten zullen worden gevisualiseerd in Angular componenten, zodat ze makkelijk kunnen worden toegevoegd aan de bestaande webapplicatie. Tevens zal door het verzamelen en structureren van de nieuwe data, verdere verbeteringen en ontwikkeling van analyse producten makkelijker gemaakt worden.

5. Uit te voeren werkzaamheden, inclusief een globale fasering, mijlpalen en bijbehorende activiteiten

Gedurende het project zal er ongeveer 20% overhead zijn om de kwaliteit van mijn afstudeer documentatie en verslagen te garanderen en om aan andere afspraken omtrent het afstuderen te voldoen. Dit geeft mijn ongeveer 70 dagen om aan het project te werken. Niet alle taken zullen noodzakelijk op hele of achtereenvolgende dagen plaatsvinden.

- Meeting met DIGICOR collaborators/stakeholders in Rotterdam. [1 dag]
- Plan van aanpak maken. [6 dagen]
- Bekend worden met nieuwe technieken (web crawling, java environment, Docker). [4 dagen]
- Ontwerpen van architectuur en data structuren. [6 dagen]
- Module voor dataverzameling maken met web crawling/scraping. [11 dagen]
- Onderzoek doen naar sentiment analyse strategieën en implementaties [2 dagen]
- Verzamelde data structureren en met sentimentanalyse tot relateerbare data omvormen. [8 dagen]
- Angular project opzetten om de data te visualiseren in een webapplicatie. [11 dagen]
- Integratie van verschillende onderdelen van het product en verbeteringen waar mogelijk. [3 dagen]
- Integreren met de DIGICOR event store en data aanleveren. [3 dagen]
- Functionaliteit en uitbreidbaarheid van het product testen. [2 dagen]
- Documenteren van API's / applicaties. [2 dagen]
- Werk voor het overdragen. [3 dagen]
- (Continue) Feedback vragen en verwerken. [5 dagen]
- Contact met, of bezoek van de begeleider van mijn opleiding. [1 dag]

6. **Op te leveren (tussen)producten**

- Web applicatie of web componenten voor het visualiseren van risicoanalyse.
- Web crawler/scraper om informatie over bedrijven te verzamelen.
- Sentiment analyse systeem om reputatie risico data te verwerken.
- Integratie met de DIGICOR event store en ecosysteem.
- Architecturale oplossing voor het omgaan met een groeiend aantal informatiebronnen.

7. **Te demonstreren competenties en wijze waarop**

Verplichte competenties

A1 – Analyseren probleem domein. (Analyse probleem en oplossingsrichting)
Gc – Kritisch en methodisch werken. (Verantwoording voor keuzes en zelfreflectie)
Gf – Leren leren. (Leren werken met nieuwe technieken en onderzoek naar technologische oplossingen en selecties van bruikbare technieken)

Gekozen competenties

D14 – Realiseren van software. (Algemene oplevering van mijn product)
C6 – Ontwerpen software (Architectuur voor gedistribueerde event processing, design patterns, etc.)

Plan van aanpak

Risk analysis tool

Plan van aanpak voor afstudeeropdracht
Student: Menno Guldemond
Studentnummer: 12049530

Onderwijsinstelling:	Haagse Hogeschool | HBO-ICT Software Engineering (voltijd)
Opdrachtgever:	Almende B.V.

Inhoudsopgave

1. Inleiding

Het project bij Almende zal anders zijn dan mijn vorige werkervaringen. Almende doet veel research projecten, waarbij de gewenste eindresultaten vaak veel minder vastliggen en het eindproduct nog niet klaar is voor de markt. Deze onderzoeksgerichte projecten zijn iets wat ik graag eens wilde ervaren. Ik hoop dat in een omgeving als deze meer ruimte is voor experimenten en kennis deling, en waarbij de focus meer ligt op nieuwe inzichten en resultaten, in plaats van heel procesmatig werken. Daarnaast heb ik bijna exclusief ervaringen met Microsoft technologieën (denk aan .Net core, C#, MSSQL). Tijdens dit project zal ik met veel andere technieken werken. Na deze periode hoop ik een betere mening te kunnen vormen op de verschillen in deze manieren van werken en ook mijn ervaring verbreed te hebben.

Bij Almende werken ze redelijk volgens het principe van rapid prototyping en gebruiken ze niet een methodiek als SCRUM. Om hier op aan te sluiten zal ik ook vrij snel prototypes ontwikkelen en met de vergaarde inzichten verder ontwikkelen of een nieuwe opzet maken en die verder iteratief uitgewerken.

Dit project wordt uitgevoerd door meerdere bedrijven uit verschillende landen. De voertaal is Engels en samenwerking op afstand is aan de orde van de dag. Almende voert slechts een klein deel uit van de werkzaamheden en een deel van de architectuur, technieken is bepaald door andere bedrijven.

2. Organisatiebeschrijving

Almende is een in Rotterdam gevestigd bedrijf en opgericht in 2000. Het bedrijf voert R&D activiteiten uit in verschillende ICT domeinen, wisselend van gezondheidszorg tot productie systemen. Almende is betrokken bij verschillende Europese en Nederlandse gefinancierde projecten samen met andere industriële en academische partners.

Het aantal werknemers fluctueert regelmatig, maar is op het moment van schrijven rond de vijftien mensen. Deze fluctuatie is het resultaat van projecten die gecommercialiseerd worden in dochterbedrijven. Het team bestaat uit mensen met verschillende vaardigheden, gevolgde opleidingen (minimaal B.Sc) en afkomst.

3. Probleemstelling

Het probleemdomein van deze opdracht is supply chain en risicoanalyse. Consortiums van bedrijven willen meer inzicht in de samenwerking met nieuwe partners. Dit is onderdeel van het H2020 DIGICOR project (decentralized agile coordination Across supply chains). Het doel van Almende voor dit project is om meer inzicht te geven in samenwerking risico's, te onderzoeken hoe uitvoerbaar het is om voorspellingen te kunnen doen over samenwerking risico's en de bevindingen te gebruiken in toekomstige implementaties.

Het DIGICOR project wordt opgeleverd op TRL (Technology readiness level) 7. Dit houdt in dat het product tot op een niveau van "Close to market" ontwikkeld wordt. Er is ook nog niet een definitief exploitatieplan voor als dit project levensvatbaar wordt bevonden.

De specifieke probleemstelling kan als volgende geformuleerd worden:

1. Op dit moment heeft het platform niet veel data van bedrijven om mee te werken. Almende wil daarom iets willen ontwikkelen om automatisch data van bedrijven die op het web te vinden is te verzamelen en analyseren.
2. Het component moet geïntegreerd worden in een event-based architectuur die verschillende bronnen van informatie ondersteund om een risicoanalyse op te baseren. Een groot deel van deze architectuur moet nog worden ontworpen.
3. De bevindingen (ofwel risicoanalyse uitslagen) moeten gevisualiseerd worden voor eindgebruikers in een webapplicatie.

De probleemstelling is dus: Op wat voor manier kan de bestaande DIGICOR architectuur uitgebreid worden met een software component voor het automatisch verzamelen en analyseren van data van het internet over bedrijven om iets te kunnen zeggen over het risico van een samenwerking met een consortium van bedrijven en hoe kunnen de resultaten geproduceerd door dit component, als deze gecombineerd is met andere beschikbare data, gevisualiseerd worden voor eindgebruikers.

4. Requirements

De requirements zijn opgesteld door het analyseren van het probleemdomein, gesprekken met de opdrachtgever/stakeholders en het bestuderen van bestaande use-cases evenals het maken van eigen use-cases (zie use-cases bijlage).

In principe is Almende als mijn product owner leidend in de uiteindelijke keuzes voor mijn werk binnen dit project. De nadruk ligt op het succesvol opleveren van de “deliverables” voor het Horizon 2020 programma en het vergaren van nieuwe kennis en inzichten.

De wens van Almende is om geautomatiseerd risicoanalyses te kunnen doen en de resultaten overzichtelijk te visualiseren. Om dit te realiseren worden de volgende globale onderdelen door mij opgeleverd. *(alle requirements zijn gesorteerd op prioriteit of mate van belangrijkheid)*

1. Web components compatibel binnen de bestaande Angular webapplicatie voor de visualisatie van risicoanalyse resultaten en het aanpassen/inzien van de risico recepten/rekenmethoden.
2. Web scraping mechanisme om artikelen over bedrijven te vergaren ter input voor reputatierisico scorebepaling.
3. Sentiment analyse mechanisme om artikelen te verwerken en om te zetten naar een reputatie score.

Technische requirements

1. De diensten en recepten moeten uitbreidbaar zijn voor nieuwe implementaties.
2. Web scraper moet geïsoleerd/onafhankelijk kunnen werken van andere modules.
3. Integratie van services met de DIGICOR event store en het ecosysteem.

Functionele requirements

1. De gebruiker wil inzicht kunnen krijgen in het risico van samenwerken met een bepaalde partij.
2. De gebruiker wil een recept kunnen configureren of wijzigen.
3. De gebruiker wil kunnen inzien waaruit een risico is opgebouwd.
4. De gebruiker wil een globaal overzicht van alle risico’s voor zijn/haar bedrijf d.m.v. een dashboard.
5. De gebruiker wil kunnen aangeven welke risico’s zwaarder meewegen in de algemene berekening.

Beperkingen

- De back-end moet met Java geschreven worden (Certicon wilt het aantal te onderhouden talen beperken).
- Front-end moet gemaakt worden met Angular (wens Certicon, kan dan makkelijk geïntegreerd worden in de bestaande Angular applicatie).
- Er wordt geen gebruik gemaakt van state management door iets als Redux.

Risico’s

- Veel data waar we mee willen werken in het platform is niet beschikbaar voor ons. Denk aan informatie van bedrijven over financiën, project (uitloop)tijd, etc.
- Doordat het project niet in productie wordt genomen bij oplevering en ook niet op dat niveau wordt opgeleverd kan het lastig zijn om volledig duidelijke en concrete eisen en beperkingen te verkrijgen van stakeholders.
- Doordat er meer nadruk ligt op iets demo-waardigs i.p.v. een schaalbaar en stabiel platform, worden keuzes gemaakt die de kwaliteit kunnen beïnvloeden.

5. User stories

Voor alle onderstaande user stories zijn de vereisten dat een gebruiker is ingelogd en de juiste rechten heeft om de “Risk Analysis Service” (RAS) te gebruiken. Verdere vereisten/aannames worden specifiek benoemd.

1. Een gebruiker wilt in de lijst van potentiële bedrijven voor uitvoering van een aanbesteding een *algemene* risico-indicatie zien.
    a. **ALS** een gebruiker een aanbesteding in het systeem heeft staan.
    b. **EN** er zijn potentiële bedrijven voor de uitvoering.
    c. **EN** er is een risicoscore berekend.
    d. **DAN** wilt hij bij de lijst met potentiële bedrijven een risico-indicatie zien.
    e. **ANDERS** staat er in de lijst bij het bedrijf een indicatie dat er nog geen risico bekend is.
2. Een gebruiker wilt een risico overzicht via het RAS dashboard zien.
    a. **ALS** een gebruiker klikt op de risico indicatie van een bedrijf.
    b. **EN** er risico data beschikbaar is voor dat bedrijf.
    c. **DAN** krijgt de gebruiker een dashboard te zien met een globaal overzicht van de beschikbare berekende risico's.
3. Een gebruiker wilt informatie van een specifieke risicoscore inzien.
    a. **ALS** een gebruiker klikt op een risicoscore vanuit het dashboard.
    b. **DAN** krijgt de gebruiker een meer gedetailleerd overzicht te zien van het betreffende risico.
4. Een gebruiker wilt weten uit welke sub-risico’s een risico opgebouwd.
    a. **ALS** een gebruiker zich op het scherm van een specifiek risico bevindt.
    b. **EN** het betreffende risico heeft sub-risico’s.
    c. **DAN** ziet de gebruiker onder de informatie een globaal overzicht van de sub-risicoscores.
5. Een gebruiker wilt informatie van een specifiek sub-risico inzien.
    a. **ALS** een gebruiker klikt op een sub-risico vanuit een risico overzicht.
    b. **DAN** verandert het “hoofd-risico” van het scherm naar die van het sub-risico.
6. Een gebruiker wilt vanuit een risico overzicht naar het bovenliggende risico waar het onderdeel van is.
    a. **ALS** een gebruiker zich op het scherm van een specifiek risico bevindt.
    b. **EN** het betreffende risico is onderdeel van een algemener risico.
    c. **DAN** Kan de gebruiker navigeren naar dat risico.
7. Gebruiker wilt een specifiek recept kiezen om een risicoscore te bepalen.
    a. **ALS** een gebruiker zich op het scherm van een specifiek risico bevindt.
    b. **EN** er zijn voor dat risico meerdere recepten beschikbaar
    c. **DAN** kan de gebruiker het recept wijzigen in de gebruikersinterface.

6. Planning / fasering

Op de werkwijze aansluitend van Almende zal ik voor de planning een deadline aanhouden en een duidelijke scope bepalen, maar niet mijn werk terugbrengen tot een aantal uren werklast. Het project heeft een hoge mate van onderzoek en experimentatie en de deliverables zijn niet in hoog detail beschreven. De focus ligt op het verkrijgen van inzichten en zo goed mogelijk voldoen aan de deliverables.

Dit is een andere manier van werken dan dat ik tot op heden gewend was. Maar het biedt een goede ervaring om volgens een andere denkwijze en methodiek te werken. Ik zal continue en zo snel mogelijk feedback proberen te vergaren om snel te kunnen inspringen op de nieuwe inzichten en eventueel wijzigende wensen.

Mijn planning zal worden opgedeeld in mijlpalen met een ruwe inschatting qua tijdsbestek.

Mijlpalen

1. Alle ontwikkel omgevingen werkend op mijn pc (voor week 2)
2. Prototype front-end risk visualization component(voor week 2)
3. Prototype article web-crawling (voor week 2)
4. Prototype sentiment analysis (voor week 3)
5. Integratie met bestaande DIGICOR API / event store (voor week 5)
6. Data model voor risico’s in context van front-end en communicatie (voor week 7)
7. 1e versie front-end (voor week 8)
8. 1e versie web-crawler (voor week 8)
9. 1e versie sentiment analysis (voor week 9)
10. “Half-way” evaluatie met eventuele bijsturing of scope verandering (voor week 10)
11. Opleveren documentatie aan Almende (voor week 17)
12. Opleveren alle producten (voor week 17)
13. Opleveren afstudeer document opleiding (voor week 17)

7. Technieken

Voor dit project zijn er een aantal technieken bepaald door andere bedrijven die aan dit project meewerken. Daarnaast loopt dit project al twee jaar en zal Q4 van 2019 opgeleverd worden (op TRL 7, close to market).

Back-end

Hieronder valt alles te maken met het vergaren van data voor risicoanalyse en de REST-API die de data en functionaliteit biedt voor de front-end.

- Java (JDK 8 from Oracle)
- Spring framework[18]
- Maven[19]
- Docker[20]
- EVE[21] (Almende's own web-based agent platform)

Front-end

De front-end applicatie waar ik aan werk is geschreven in Angular (niet te verwarren met AngularJS). Ik zal de benodigde componenten ook schrijven in Angular en verpakken in een eigen module.

- Angular[22] (versie 7)

[18] https://spring.io/
[19] https://maven.apache.org/
[20] https://www.docker.com/
[21] https://eve.almende.com/
[22] https://angular.io/

Adviezen voor huidige front-end

Dit document wil ik wijden aan de restricties, implementaties en mijn mening over de huidige versie van de DIGICOR front-end. Doordat verschillende partijen aan dit product werken, en omdat niet alle partijen veel ervaring hebben met moderne web-component gebaseerde front-end applicaties zijn er zaken die ik persoonlijk anders had aangepakt. Hieronder een overzicht van mijn adviezen om de applicatie te verbeteren met daarbij wat argumentatie.

1. Redux globaler toevoegen voor state management.
    a. Zorgt voor voorspelbaarder gedrag.
    b. Kan onwenselijk dependencies verminderen.
    c. Voorkomt onoverzichtelijkheid.
2. Inline styles vermijden.
    a. Inline style is lastig te overschrijven.
    b. Teken van ondoordachte styling regels en minder goede structuur.
3. Abstractie maken van de Http Service om authenticatie te gebruiken.
    a. Nu heeft elke service die http requests doet een dependency op de authenticatie service, mocht hier iets aan wijzigen, dan moet men elke service aanpassen.
4. Onnodige 3rd party bibliotheken verwijderen.
    a. Sommige bibliotheken doen gelijke dingen, hiervoor zou één bibliotheek als oplossing gekozen moeten worden.
    b. Sommige bibliotheken worden helemaal niet gebruikt, maar zitten wel in het project.
5. Proberen meer modules pas in te laden als ze echt nodig zijn.
    a. Door modules pas later te laden kan de opstarttijd van de applicatie gereduceerd worden.
6. Shared module goed inzetten.
    a. De shared module zou modules moeten blootstellen die door veel andere modules gebruikt worden en niet zaken die op een specifieke plek gebruikt worden.
    b. Services kunnen niet op de normale manier als een singleton fungeren als ze “provide” worden in de shared module, en horen daar in veel gevallen niet thuis.
7. Geen elementen ophalen door de DOM te query-en.
    a. Dit hoort niet thuis in een Angular applicatie. Er hoort gewerkt te worden met Data en event binding of met View Childs.
    b. Dit is fragiel, en kan niet goed omgaan met de shadow dom.
8. Houd/maak componenten kleiner.
    a. Dit zorgt voor makkelijker te onderhouden componenten.
    b. Dit maakt componenten makkelijker generiek inzetbaar.

Reputation risks documentation

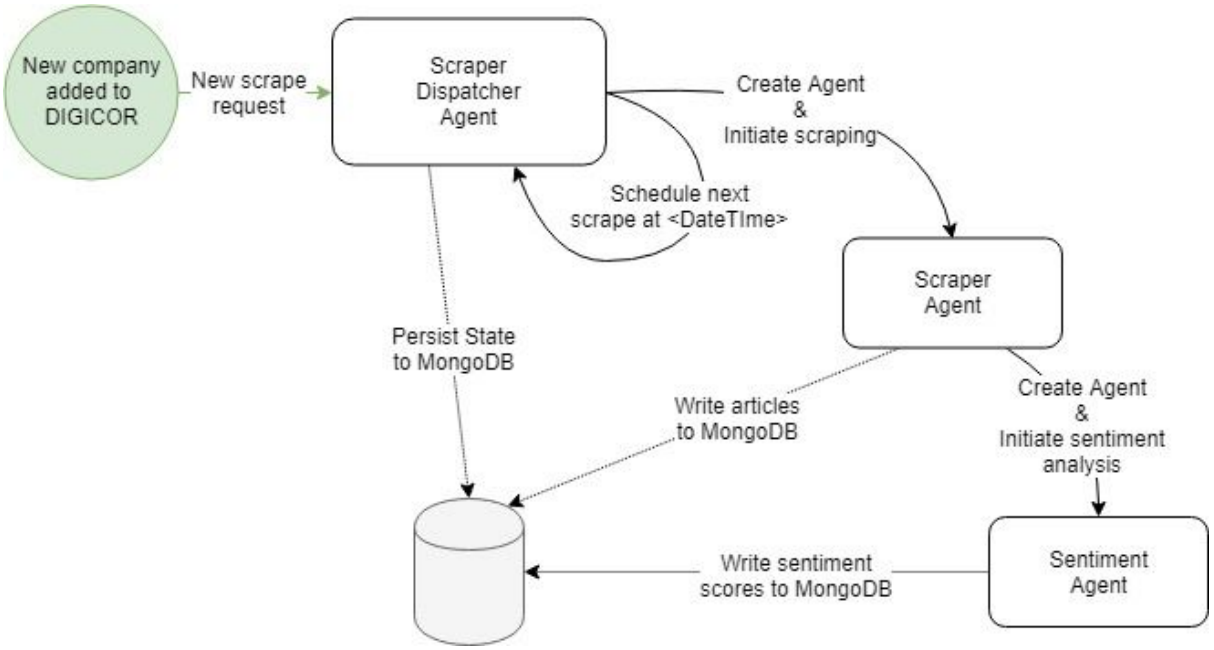
For the goal of having more risks and data to develop the platform we have introduced the concept of a reputation risk.
There are currently two reputation risk recipe implementations.

- semi-randomly generated reputation risk
- sentiment reputation risk

The sentiment reputation risk was added providing real data as an alternative to the semi-randomly generated reputation risk. When a new company is created, a software agent will start gathering articles regarding the given company. Once collected, another agent will perform a sentiment analysis on the articles.

Architecture

The risk tool makes use of the web-based agent platform Eve. Each agent has one responsibility, and multiple agents can work in parallel. there is a schematic overview of the agents involved in collecting the sentiment data In the figure below.



Scraper dispatcher agent

- The entry point for new companies that need to have their reputation risk calculated.
- Initiates scraper agents.
- Schedule interval at which to update reputation data.
- Persist schedule state to the database to continue in case the system would go down.

Scraper agent

- Gathers articles from news outlets.
- Checks for new articles comparing the already processed ones.
- Writes articles and relevant meta data to the database.
- Initiates sentiment agents.

Sentiment agent

- Using a natural language processing model to decompose and and analyze the sentiment of articles.
- Writes weighted sentiment scores to the database.

Sentiment Recipe Agent (retrieving data)

- Gives a RecipeNumber given a CompanyId.
- Used for retrieving a calculated company reputation risk.

Completion document

DIGICOR risk-tool

This document describes the overall structure and information about the software made for the risk-tool as part of the DIGICOR project.

Front-end

The front-end risk-module source code is available in the web-1.0 Angular project, available at: https://git.digicor-platform.eu/digicor/web-1.0 following the file path: src\app\main\tools\registered-tools\risk

Dependencies of risk-module (excluding standard Angular libraries):

- RxJS
- Angular: Material
- Angular: Flex Layout
- AuthorizedHttpClient (made by Almende/Certicon)
- AuthenticationGuard (made by Certicon)

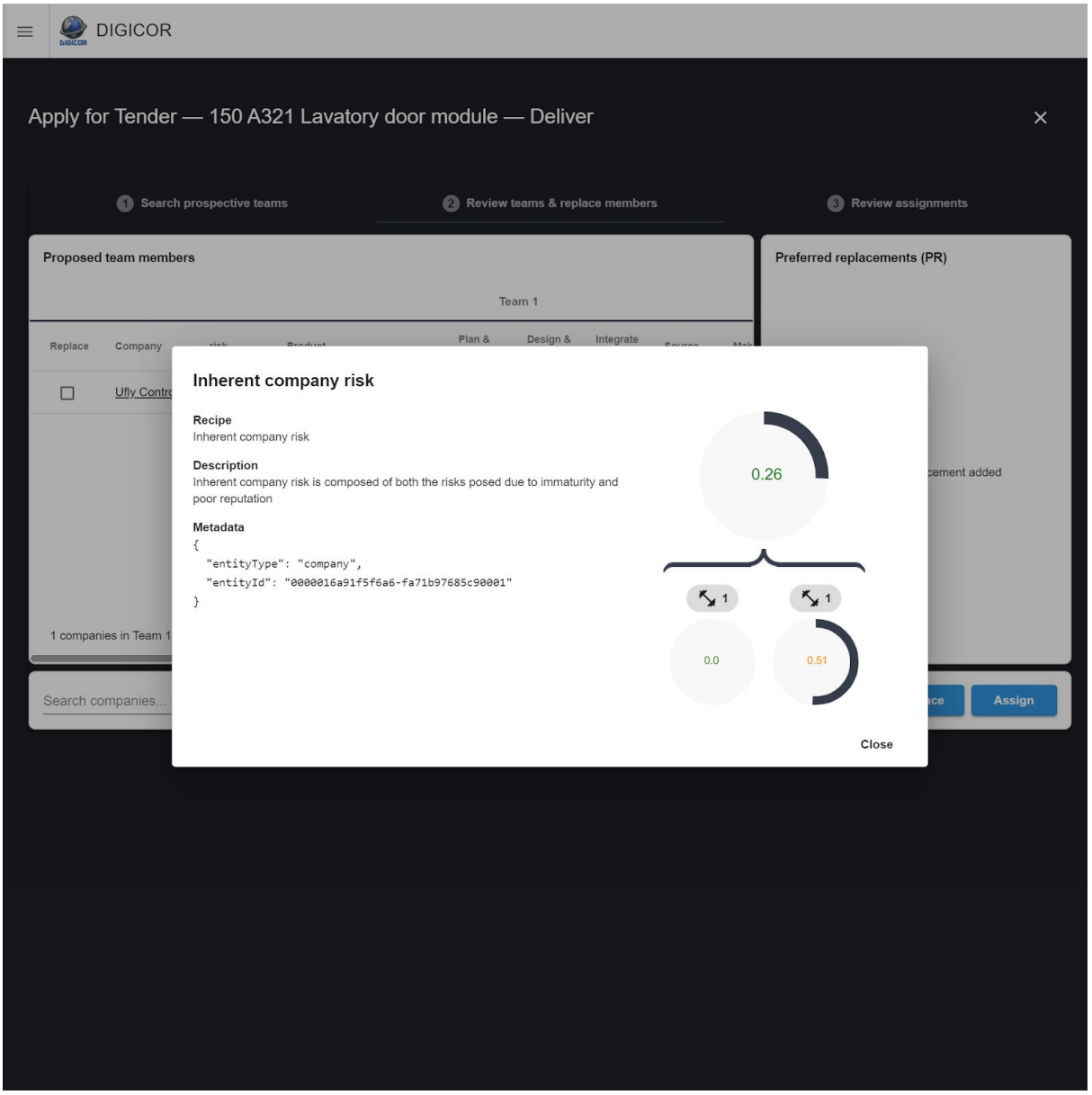
Back-end

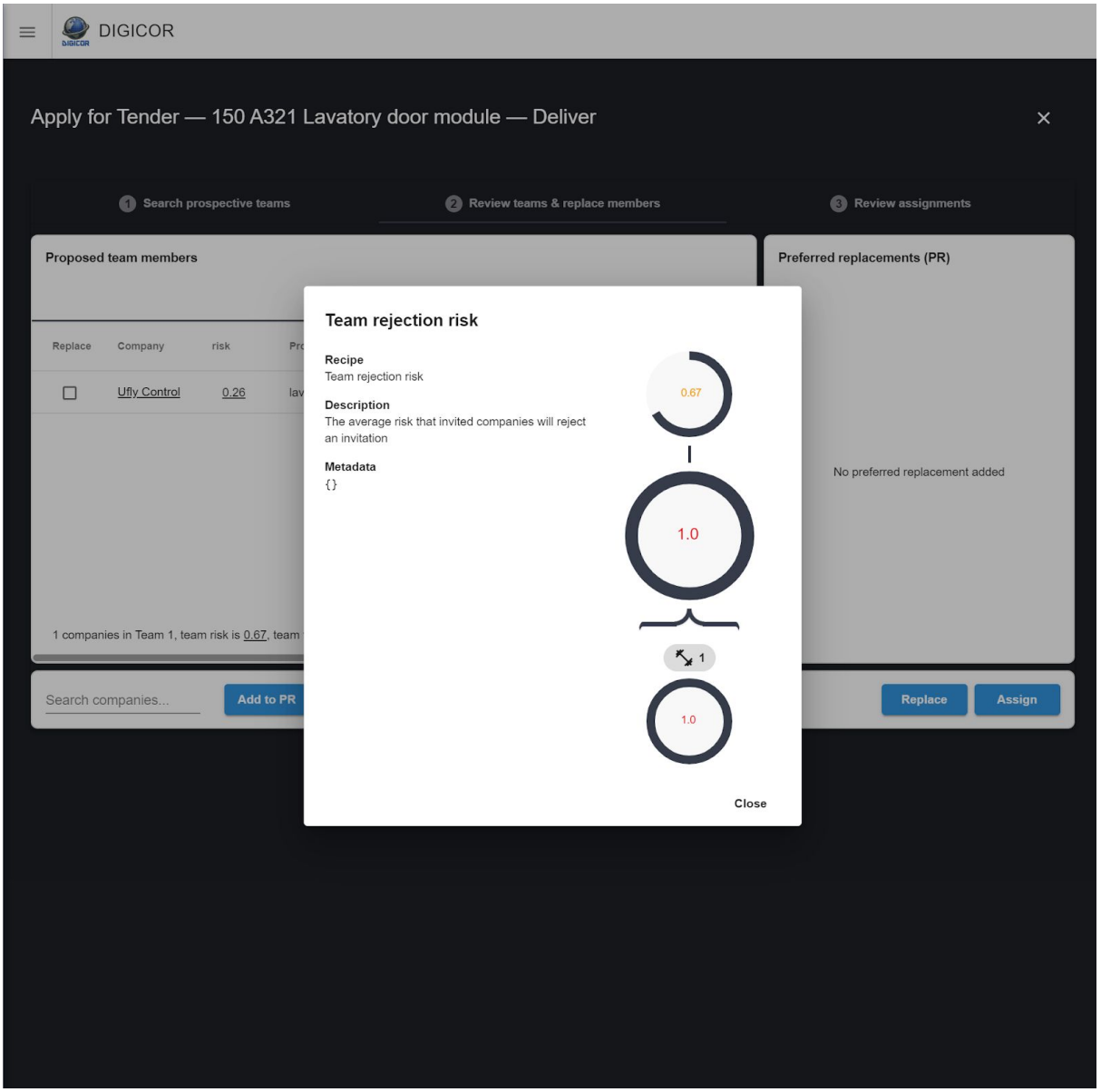
The back-end service source code is available at:
https://git.digicor-platform.eu/digicor/java-services/risk-analysis

- The project consists of a REST/agent api and a test project.
- The registration of services and agents can be done in BackendConfiguration.java
- Configuration of database connection strings, the URL for agents and other environment variables can be set in the resources/application.properties at each level accordingly.

Screenshots front-end

In onderstaande figuren zie je enkele screenshots waarin visualisatie van risico's te zien zijn. Een risico kan zijn opgebouwd uit meerdere subrisico's. Deze staan onder het huidige risico weergeven met een aanpasbaar gewicht. Met de gewichten kan de weging van de subrisico's aangepast worden. Als een risico onderdeel is van een overkoepelend risico staat deze boven het huidige risico weergeven.




DIGICOR
Apply for Tender — 150 A321 Lavatory door module — Deliver
1 Search prospective teams
2 Review teams & replace members
3 Review assignments
Proposed team members
Replace
Company
risk
Ufly Control
0.26
1 companies in Team 1, team risk is 0.67, team
Search companies...
Add to PR
Preferred replacements (PR)
No preferred replacement added
Replace
Assign
Team rejection risk
Recipe
Team rejection risk
Description
The average risk that invited companies will reject an invitation
Metadata
{}
0.67
1.0
1
1.0
Close