Ontwikkelen van een project checklist voor de sterk uiteenlopende projecten bij Thirdwave.

Robin Bastiaan

HBO-ICT DT

Onderwijsinstelling:	De Haagse Hogeschool
Opleiding:	Software Engineering
Student:	Robin Bastiaan
Studentnummer:	
Afstudeerbegeleider:	Rianne Bechet - Tjoonk
Expert examinator:	Alphons Vinkesteijn
Opdrachtgever:	Thirdwave B.V.
Datum:	2021-06-04
Versie:	1.0

Inhoudsopgave

Voorwoord

Tijdens mijn afstudeerperiode heb ik een prototype opgeleverd genaamd “Preflight” waarbij men een checklist heeft waarop je taken kan afvinken die gedaan moeten worden voordat een project wordt opgeleverd. Hierdoor is het duidelijker en overzichtelijker bij de verschillende developers wat er allemaal gedaan moet worden, waardoor er minder vergeten wordt en fouten worden voorkomen.

Deze opdracht voer ik uit binnen de opleiding HBO-ICT aan De Haagse Hogeschool. Ik vind het fijn om kwaliteit op te leveren voor klanten en kleine foutjes die voorkomen hadden kunnen worden ook daadwerkelijk op te lossen voordat deze een probleem zouden worden. Door structuur aan te brengen in het proces weet je waar je aan toe bent en dat geeft ook een zekere rust en vertrouwen. Ik ben er van overtuigd dat dit ook zeker juist bij Thirdwave de developers kan ondersteunen, omdat er veel kleine websites worden opgeleverd met veel maatwerk die allemaal net wat anders zijn en dus andere dingen hebben waar op gelet moet worden.

Dit verslag is geschreven voor de examinatoren die uiteindelijk een beoordeling moeten uitbrengen voor mijn werkzaamheden. In dit document zal ik dus trachten een zo duidelijk mogelijk beeld te geven van zowel het proces, de gemaakte keuzes, en de verschillende (deel)producten.

Graag wil ik mijn collega’s en specifiek mijn bedrijfsmentor Guido Gautier bedanken voor hun hulp. Zonder hen was dit document niet mogelijk. De ontvangen hulp bestaat onder andere uit het meewerken met interviews, en het geven van feedback op het prototype. Ik heb met hen op effectieve wijze kunnen sparren over mijn scriptie. Mijn dank gaat ook specifiek uit naar de proeflezers die tussentijds feedback hebben gegeven op dit verslag.

Tenslotte wil ik Patricia Moscoso bedanken voor haar steun en toeverlaat tijdens onze wekelijkse *stand up*. Zij is een mede-afstudeerder die werkt aan een ongerelateerd onderwerp bij een ander bedrijf en door elkaars afstudeer ervaringen te delen kon ik deze opdracht in een beter perspectief zien.

Ik wens je veel leesplezier toe.

Robin Bastiaan
Leiden, 4 juni 2021

Samenvatting

De probleemstelling heb ik als volgt verwoord: “Thirdwave heeft geen overzicht welke stappen en productkwaliteitscontrole er voor de diverse projecten genomen moeten worden om een project op te leveren.”

De aanpak bevat elementen van het Scrum raamwerk zoals het werken aan de hand van sprints. Er vonden ‘sprints’ met een duur van 2 weken per sprint plaats, en gezien de duur van het afstudeertraject betekent dit 8 sprints. De precieze invulling van de sprints is afhankelijk van het definitieve verloop en voortschrijdend inzicht. Zo’n sprint start met een planning en wordt afgesloten met een evaluatie. Tussen de sprints door kan er op deze manier grip worden gehouden op de globale voortgang en bijgestuurd worden indien nodig.

De uitvoering vond plaats tussen februari van 2021 en juni 2021. Dit was gedurende een Corona pandemie waardoor er enkel digitaal met collega's gecommuniceerd kon worden. Ook is gedurende de uitvoering van het project zowel mijn opdrachtgever als mijn begeleidend examinator weggevallen. Ter vervanging van de opdrachtgever is de bedrijfsmentor een dubbele taak gaan vervullen en ook en ter vervanging van de begeleidend examinator is mijn expert examinator de rol van begeleidend examinator gaan vervullen en is er een nieuwe expert examinator aangesteld.

Het resultaat is een prototype. Dit prototype is gebouwd met gebruik van het PHP Framework Symfony en is meerdere malen getest op bruikbaarheid en uit de evaluaties zijn verbeteringen gekomen die zijn doorgevoerd in een nieuwe versie van het prototype.

1 Inleiding

Het waarborgen van de kwaliteit van een product is niet iets nieuws; zeker niet binnen de ICT. Ook het internetbureau Thirdwave heeft hier mee te maken. Tijdens mijn afstuderen heb ik een op maat gemaakte oplossing voor Thirdwave gemaakt welke in dit thema op oplossing biedt.

In de verschillende hoofdstukken wordt de uitvoering van de afstudeerscriptie verwoord. Ter introductie begin ik met iets te vertellen over het bedrijf waarbinnen ik deze opdracht heb uitgevoerd. Daarna vervolg ik met het beschrijven van de opdracht inclusief de doelstelling.

Vervolgens ga ik de diepte in en behandel ik in chronologische volgorde de verschillende genomen stappen. Deze stappen worden grotendeels in chronologisch volgorde doorlopen, maar soms ook gedeeltelijk synchroon om voorbereidend werk te verrichten. Bij iedere stap zet ik uiteen welke keuzes ik heb gemaakt en waarom.

Ten slotte sluit ik af met een evaluatie. Ik evalueer over het opgeleverde prototype, de beroepstaken en het gehele proces in het algemeen.

Helemaal aan het einde van dit verslag is er een begrippenlijst te vinden waar sommige technische termen staan toegelicht voor het geval deze tijdens het lezen van het verslag niet duidelijk genoeg beschreven zijn. Ook kan je hier een literatuurlijst, tabellenlijst en de bijlagen vinden.

2 Thirdwave; een fullservice internetbureau

Thirdwave heeft meerdere klanten en deze klanten bestaan hoofdzakelijk uit beroepsverenigingen, maar ook diverse andere bedrijven kunnen gebruik maken van de expertise van Thirdwave voor het bouwen van custom websites en apps op basis van de unieke requirements van de desbetreffende klant.

Thirdwave is een klein bedrijf met zo'n 12 werknemers. De collega's vormen een divers team en bestaat uit designers, front-enders, back-enders, een security officer en een projectmanager. Zoals te zien in figuur 1 is het kantoor ruim opgezet. Toch is het zo dat tijdens het uitvoeren van deze opdracht vrijwel alle collega's digitaal vanuit huis werken.

Figuur 1: Thirdwave kantoor

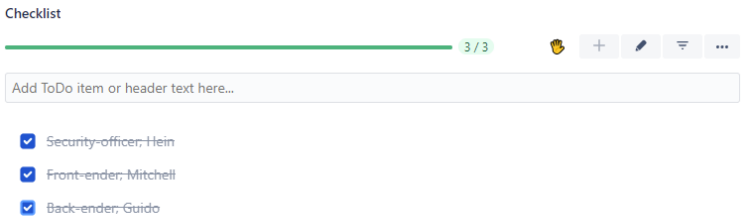
Door de grote diversiteit van de op maat gemaakte websites die Thirdwave oplevert voor zijn klanten, is vrijwel ieder project net wat anders. Voorbeelden zijn een ander e-mailsysteem, een ander CMS voor de beheerder van de website en een andere vorm van hosting.

Dit geeft aanleiding voor het aankaarten van het probleemdomein dat door de grote hoeveelheid aan stappen het overzicht te klein wordt gevonden, wat zich uit in het feit dat gemakkelijk fouten gemaakt worden, te laat bepaalde stappen genomen worden, kennis niet voldoende gedeeld wordt en er ad hoc te werk gegaan wordt. Dit probleem wordt versterkt door het feit dat de stappen per project sterk verschillen. Een andere belangrijke factor bij het beschrijven van het probleem is dat de stappen vaak niet opgeschreven worden, waardoor er in mindere mate een overzicht is en er afhankelijkheden worden gecreëerd van bepaalde betrokken personen. De stappen worden namelijk voornamelijk uit het hoofd gedaan op basis van ervaring.

In de huidige werkwijze wordt Jira gebruikt als project management software en deze is niet voldoende, omdat Jira niet de benodigde functionaliteiten bevat, welke in het volgende hoofdstuk verder zal worden toegelicht.

3 Opdrachtomschrijving; Preflight

Thirdwave kampt al langere tijd met het hiervoor beschreven probleemdomein en hier is tot op heden geen adequate oplossing gevonden. Eerdere periodieke overleggen (ook wel *retrospectives* genoemd) hebben dit probleem al aangekaart en hierop zijn ook al eerdere stappen ondernomen. Er is momenteel hierdoor een Jira *plugin* in gebruik genomen welke een vooraf instelbare checklist kan inladen en welke op deze manier stap voor stap kan worden afgewerkt. Zie figuur 2 om een indruk te krijgen van de UI van deze Jira *plugin*. De huidige ervaring met dit systeem is dat het lastig is om deze correct op te stellen, dat de regels van de checklist te weinig extra informatie kunnen bevatten en dat er geen acties direct vanuit de plugin geïnitieerd kunnen worden waardoor het niets kan automatiseren.



Figuur 2: Jira Plugin

De uitdaging en dus de complexiteit van het ontwikkelen en afsluitend opleveren van een product is de grote verscheidenheid van de verschillende projecten en klanten van Thirdwave. Daarom is het belangrijk om zo generiek mogelijk te werk te gaan en diverse configuraties mogelijk te maken, zodat er flexibel kan worden aangepast aan een andere opzet via configuratie en er zo met veel verschillende mogelijkheden rekening kan worden gehouden.

De belanghebbenden bij deze situatie zijn alle medewerkers van Thirdwave, als mede indirect de klanten van Thirdwave. Concreet kunnen de volgende rollen worden benoemd:

- Developer
- Klant
- Project manager
- Security officer

Samengevat wordt de probleemstelling als volgt verwoord: “Thirdwave heeft geen overzicht welke stappen en productkwaliteitscontroles er voor de diverse projecten genomen moeten worden om een project op te leveren.”

Zie ook de bijlage “1. Afstudeerplan HBO-ICT Robin Bastiaan”.

3.1 Doelstelling en Resultaat

Na overleg met de opdrachtgever wordt er voor het gemak vanuit gegaan dat de developers baat hebben bij een systeem waarin via een “project checklist” overzicht van de te nemen stappen wordt gecreëerd en er een gestandaardiseerde kwaliteitscontrole plaatsvindt. Uiteraard zal in de praktijk blijken of deze veronderstelling juist is. Met de term “project checklist” wordt het volgende bedoeld:

- **Project:** Een op te leveren product welke vaak een website is maar soms ook een app. In de regel wordt dit project gemaakt voor een klant, maar kan in theorie ook een intern product zijn.
- **Checklist:** Een lijst met uit te voeren taken die gedurende de ontwikkeling van een project gebruikt kan worden om de status van de te nemen stappen bij te houden en af te vinken na voltooiing.

Vervolgens kan er gekeken worden of er repeterende stappen geïdentificeerd kunnen worden, zowel tussen projecten als binnen één project, die geautomatiseerd kunnen worden wat voor een snellere oplevering van een project kan zorgen.

Door middel van een prototype formuleer ik een oplossing geformuleerd op de probleemstelling welke in het vorige hoofdstuk is beschreven. Het afstudeerproject wordt vanuit het oogpunt van de developer aangevangen, die de uitvoerder is van het bouwen en opleveren van een website. Het prototype is een hulpmiddel voor de developer om overzicht te creëren en de kwaliteit van een project beter te waarborgen.

Met dit prototype is Thirdwave in staat om sneller en beter websites te realiseren voor de klant, wat is vast te stellen door een kortere ontwikkelingstijd en een kleiner aantal aan support tickets om problemen op te lossen. De doelstelling is dat er minder menselijke fouten worden gemaakt door het vergeten van stappen, en dat repetitieve stappen waar mogelijk door middel van automatisering worden uitgevoerd.

Tijdens de afstudeerperiode wordt het prototype steeds verder uitgewerkt en uitgebreid. Hierin worden de gebruikerservaringen en wensen van de ontwikkelaars centraal gesteld. Vanzelfsprekend is het hierbij van belang om het prototype gedurende de ontwikkeling van het prototype al in gebruik te nemen zodat er steeds feedback ontvangen kan worden.

4 Aanpak & Algemene Planning

De aanpak bevat elementen van het Scrum raamwerk zoals het werken aan de hand van sprints. Er zullen ‘sprints’ met een duur van 2 weken per sprint plaatsvinden, en gezien de duur van het afstudeertraject betekent dit 8 sprints. De precieze invulling van de sprints is afhankelijk van het definitieve verloop en voortschrijdend inzicht. Zo’n sprint start met een planning en wordt afgesloten met een evaluatie. Tussen de sprints door kan er op deze manier grip worden gehouden op de globale voortgang en bijgestuurd worden indien nodig.

In de tabel hieronder zet ik een globale planning uiteen waarbij ik een aantal taken formuleer die voor het uitvoeren van mijn afstudeerproject aan bod komen. Uit de tabel kan je aflezen dat ik er vanuit ga dat er in totaal 52 dagen voor het uitvoeren van de afstudeeropdracht en 10 dagen voor het opbouwen van het afstudeerdossier genomen zal worden.

# dagen	Taken	Toelichting
5	Requirementsinventarisatie maken van een actueel project.	Op basis van interviews.
4	Uitvoeren van een requirementsanalyse voor het prototype.	Vaststellen van requirements.
4	Projectplanning in Jira vastleggen.	Inclusief categoriseren dmv labels.
15	Ontwerpen (inclusief ER diagram) en bouwen van de eerste versie van het prototype.	Voor het snel mogelijk maken van iteratief testen.
4	Review van ontwerpdocumenten opstellen op basis van gebruikerservaringen met behulp van een productietestplan.	Dit vormt de basis voor de volgende versie van het prototype.
10	Verbeteringen en nieuwe functionaliteiten implementeren in het prototype	Iteratief process die vorm geeft aan het eindproduct.
2	Reflectie op basis van de STAR-methodiek en leerstijl evaluatie	Gedurende het gehele project meerdere korte momenten voor pakken.
8	Opbouw afstudeerdossier	Algemene vastlegging en documentatie van werkzaamheden.

In hoofdstuk “4.1 Planning in Detail” ga ik verder in detail in op de planning en zet ik per sprint de te verrichten werkzaamheden uiteen. Ik merk alvast op dat sommige taken elkaar op logischerwijze opvolgen en dus in een bepaalde volgorde uitgevoerd zullen worden. Daarnaast houd ik er rekening mee dat ik ruimte zal moeten vrijmaken om de evaluaties en de bevindingen van tests te verwerken in een volgende versie van het prototype.

Zoals eerder vermeld zullen er in totaal 8 sprints plaatsvinden. Per sprint heb ik de volgende doelen voor ogen:

- **Sprint 1:** Exploratie probleemdomein
- **Sprint 2:** Oplevering eerste versie prototype
- **Sprint 3:** PoC Automatisering Prototype
- **Sprint 4:** Pilot Prototype 1
- **Sprint 5:** Doorontwikkeling Prototype 1
- **Sprint 6:** Pilot Prototype 2
- **Sprint 7:** Doorontwikkeling Prototype 2
- **Sprint 8:** Afronding Afstudeeropdracht

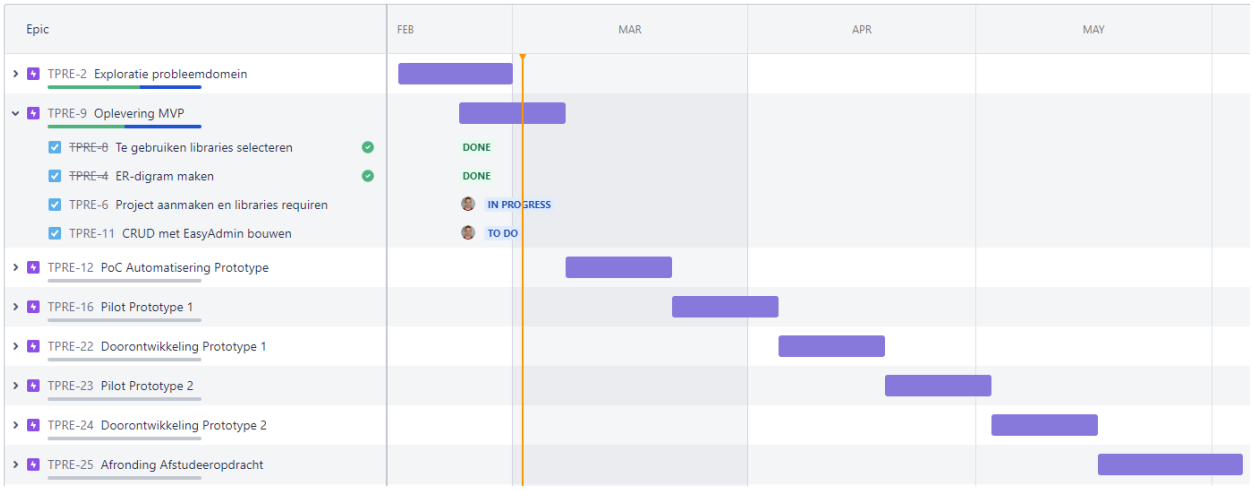
Waar de opgestelde taken uit voorgaande tabel vooral inzicht in de globale scope en te verrichten taken weergeeft, tracht ik met deze indeling van de sprints het afstudeerproject in losstaande doelen te formuleren waarbinnen deze taken uitgevoerd kunnen worden.

In de bijlage “2. Plan van Aanpak” ga ik verder in op de taken en verantwoordelijkheden, de risicofactoren, en ligt ik het activiteitenplan toe. Met dit document tracht ik op deze manier nog verder voorbereid te zijn op het succesvol voltooien van het afstuderen.

4.1 Planning in Detail

Deze requirements zijn gedocumenteerd en vervolgens in Jira opgenomen als *stories* (verhalen; deze beschrijven een functionaliteit vanuit het oogpunt van een gebruiker van het systeem) welke vervolgens weer bestaan uit *tasks* (taken) zodat het project overzichtelijk blijft.

Zoals in figuur 3 te zien is maak ik gebruik van de *roadmap* functionaliteit van Jira om een grote mate van overzicht te houden op de voortgang. Hier heb ik de tweewekelijkse sprints ingezet met de verschillende taken daaronder. Bij iedere taak geef ik door middel van *labels* aan voor welke beroepstaak die taak van belang is, wat voor mij handig te gebruiken is tijdens de verantwoording van de beroepstaken.



Figuur 3: Jira Planning 2020-03-02

Na afloop van iedere sprint evalueer ik hoe die sprint is verlopen en noteer ik dit in het voortgangsrapport. Ook neem ik vervolgens de tijd om de planning aan te passen indien nodig.

Het daadwerkelijke verloop van de planning houd ik ook globaal bij in de bijlage “3. Voortgangsrapport” met als doel om mijn bestede uren te loggen en mezelf de mogelijkheid te geven om een stukje reflectie toe te passen na iedere week.

5 Requirements

De eerste inhoudelijke stap is het inventariseren van *requirements*. Deze requirements heb ik op verschillende manieren verzameld. Ten eerste aan de hand van een literatuurstudie. Vervolgens ook door het houden van interviews met collega's die de eerder genoemde belanghebbende rollen vervullen. De resultaten van deze twee onderzoeken heb ik vervolgens samengebracht in een requirementsanalyse.

5.1 Interviews

Om waardevolle en complete informatie te verzamelen door middel van interviews, vind ik het belangrijk om gestructureerd te werk te gaan. Ik heb gekozen voor een interview waarbij ik van te voren de interviewvragen heb opgesteld. Hier heb ik voor gekozen omdat ik hierbij in kon gaan op eventuele vervolgvragen en zo dieper op de vraag in kon gaan en zo sneller een compleet beeld had van de relevante informatie. Tijdens het uitvragen van de requirements heb ik geprobeerd mee te denken met de geïnterviewde om zo alle mogelijk blinde vlekken te ontdekken.

In eerste instantie had ik twee interviews afgenomen, namelijk met de security officer en team front-end. Echter had ik nog summiere informatie vergaard over het *deploy process* zelf. Hiervoor heb ik een extra interview gepland en gehouden met een back-end collega die zich hier voor een groot gedeelte mee bezig houdt.

De transcripties van deze interviews zijn na te lezen in de bijlage "4. Requirements Interviews", evenals de complete werkwijze voor het afnemen van het interview en de gestelde interviewvragen. Tijdens de interviews kwam ook de de bijlage "6. Release Checklist" naar boven welke een lijst is met taken zoals deze voor aanvang van dit afstudeerproject door Thirdwave is opgesteld.

5.2 Requirementsanalyse

Daarna heb ik een *requirementsanalyse* opgesteld zodat duidelijk is wat de precieze scope van de applicatie is. Dit alles aan de hand van de literatuur en de interviews. Hierbij is het van belang de diversiteit van de verschillende lopende projecten helder in kaart te brengen, alsmede de

haalbaarheid van de verschillende functionaliteiten van de applicatie. Op die manier komen alle requirements tot hun recht en bevat het prototype de juiste features.

Hiervoor heb ik de requirementsanalyse in de MoSCoW prioritiserings methode gegoten en dit als leidraad gebruikt voor het invullen de items van mijn sprints. Het voordeel van deze prioritiserings methode vind ik het gemak waarmee deze kan worden opgesteld. Het maakt het voor iedereen visueel overzichtelijk hoe belangrijk ieder item wordt gevonden. En hoewel het soms niet geheel duidelijk is bij welke priorisering groep een item behoort, zorgt de discussie zelf veelal voor een juiste plaatsing. Zie “Figuur 4: De MoSCoW methode” voor een grafische weergave van dit acroniem: Must, Should, Could en Won’t.

Figuur 4: De MoSCoW methode

En die discussie was er wel, omdat mijn collega’s met verschillende rollen naar de requirements keken. Uiteindelijk heb ik de opdrachtgever natuurlijk het laatste woord gegeven, omdat die persoon er vanuit een algemene rol naar kijkt en goed afweegt wat voor het bedrijf het meest van toegevoegde waarde zal zijn. Alle requirements heb ik beschreven vanuit het oogpunt van de (algemene) developer.

Merk op dat in de lijst van de requirements er een enige vergelijking kan worden getrokken met *User Stories* zoals deze worden gebruikt binnen *Agile*. Binnen *User Stories* wordt namelijk normaliter een requirement uiteengezet in drie onderdelen: *Als een [..], wil ik [...], omdat […]*. Het tweede en derde gedeelte uit deze constructie komen grofweg overeen met de kolommen “Beschrijving” en “Waarom” respectievelijk. Omdat ik de requirements vanuit de rol van de algemene developer beschrijf, welke ook de uiteindelijke gebruiker is, laat ik die achterwege.

De complete lijst met opgestelde requirements is te vinden in de bijlage “5. Requirements Analyse”.

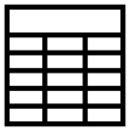
5.2.1 Requirement thema's

Uit de lijst met requirements heb ik een drietal thema's gedestilleerd. Dit heb ik gedaan om uit de lange lijst met onderdelen een algemene samenvatting te maken en zo nog meer overzicht te realiseren. Op deze manier heb ik bovendien ook meer houvast en focus tijdens het realiseren van het prototype, omdat ik bij iedere aanpassing die ik doe deze kan toetsen aan de thema's of deze voldoende bijdraagt. Deze thema's heb ik ook gedeeld met mijn collega's om te bevestigen dat dit inderdaad de thema's zijn die zij belangrijk vinden.

Op conceptueel niveau zijn de volgende drie thema's van wensen uit de interviews te benoemen.

Flexibiliteit

Door de diversiteit van de projecten moet er voldoende ruimte zijn om projecten op hun eigen unieke manier in te richten.



Standaardisatie

Door het prototype moet er standaardisatie in het uitvoeren van taken mogelijk zijn, waardoor deze niet vergeten kunnen worden en er zo inzicht wordt verkregen op datgene dat vereist is, en kennis in het proces wordt verweven.

Gebruiksgemak

Door het toepassen van automatisering en het voorkomen van meer administratie dan noodzakelijk, is er grotere kans dat het prototype niet als last ervaren wordt maar van toegevoegde waarde is, waardoor deze Met meer passie gebruikt wordt.

Hoewel ik mij eerst had voorgenomen sterk in te zetten op automatisering, heb ik dit omwille van de tijd en priorisering naar achter in de planning laten vieren. Het derde thema van gebruiksgemak heb ik vervolgens in een iets andere manier uitgewerkt. Ik heb namelijk invulling gegeven aan gebruiksgemak door een makkelijk te gebruiken front-end te bouwen. Dit thema kan in een later stadium dus verder aangevuld worden met automatisering, en op dit moment heb ik dus alsnog veel invulling kunnen geven aan dit thema.

5.2.2 Definition of Done

Een requirement zal kunnen worden afgestreept als gedaan wanneer er aan de onderstaande punten is voldaan. Deze punten vormen samen de Definition of Done (DoD).

- De code is gecommit in Git en er vindt een code review plaats.
- Indien mogelijk zijn er op redelijke wijze unittests toegepast. De unittests moeten van toegevoegde waarde zijn en niet iedere mogelijke situatie hoeft getest te worden.
- De functionaliteit is op de testomgeving gedeployed.
- Door collega's en/of de opdrachtgever is de functionaliteit bekeken en getest. Eventuele bugs zijn verholpen.

Ik kies er voor om de werken met een DoD tijdens het ontwikkeltraject zodat iedereen weet waar aan voldaan moet worden om er zo voor te zorgen dat een bepaalde functionaliteit kan worden afgestreept. Op deze manier doe ik aan verwachtingsmanagement en kan ik wat verwachten van mijn collega’s en andersom.

Als voetnoot wil ik graag opmerken dat met het gebruik van een DoD een enige vergelijking te trekken valt op de checklist zoals deze wordt vormgegeven in het prototype. Bij beide situaties controleer je namelijk of de van te voren bedachte en afgesproken punten (correct) aanwezig zijn. Dit is een toevallige bijkomstigheid.

6 Ontwerp van het Prototype

Aan de hand van de requirements heb ik vervolgens een ontwerp gemaakt. Dit ontwerp bestaat uit een *entity-relationship diagram* (ERD), het gebruik van een bepaalde architectuur, en tenslotte *design patterns*. In dit hoofdstuk zal ik deze stuk voor stuk verder toelichten en begin ik bewust met het ERD omdat deze het meest toegespitst is op deze specifieke opdracht.

Bij deze ontwerpen wordt steeds bekeken of het ontwerp aan de eisen voldoet. Daardoor zijn er van de verschillende diagrammen op itererende wijze meerdere verbeterde versies ontstaan. Ik zal alleen de meest belangrijke wijzigingen met de bijbehorende argumentatie in dit verslag beschrijven.

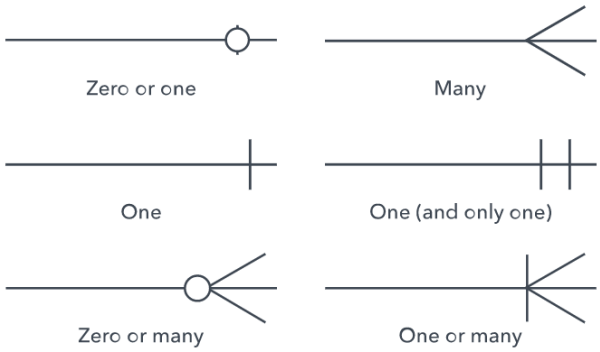
6.1 ER diagram

Om het prototype te realiseren heb ik de requirements vertaald naar verschillende entiteiten die in een database opgeslagen kunnen worden. Om een eenduidige opstelling van deze entiteiten te bewerkstelligen, heb ik de volgende definities gehanteerd:

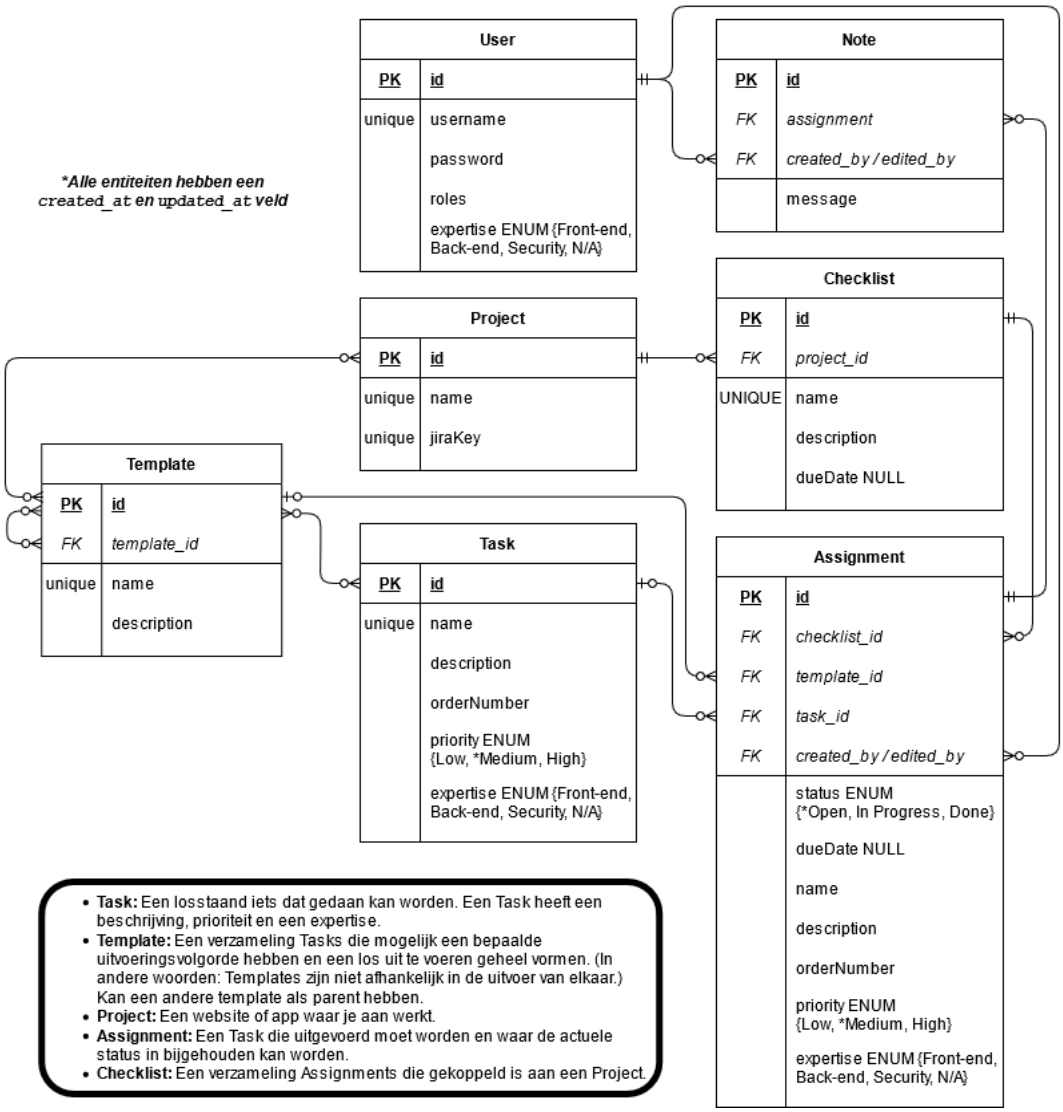
- **Task:** Een losstaande taak die uitgevoerd moet worden. Een `Task` heeft een beschrijving, prioriteit, expertise en volgorde nummering.
- **Template:** Een verzameling `Tasks` die mogelijk een bepaalde uitvoeringsvolgorde hebben en een los uit te voeren geheel vormen. In andere woorden: `Templates` zijn niet afhankelijk in de uitvoer van elkaar. Een `Template` kan (een) andere template(s) als parent hebben.
- **Project:** Een website of app die voor een klant (of intern) wordt ontwikkeld.
- **Assignment:** Een Task die uitgevoerd moet worden en waar de actuele status in bijgehouden kan worden.
- **Checklist:** Een verzameling Assignments die gekoppeld is aan een Project.

Voor het overzicht heb ik deze entiteiten gevisualiseerd in onderstaande figuur: “Figuur 6: ERD versie 7”. In dit diagram wordt op deze manier ook inzichtelijk welke velden bij elke entiteit aanwezig zijn en wat de precieze onderlinge relatie is tussen deze entiteiten.

Bij alle ER diagrammen heb ik er voor gekozen om de kardinaliteit weer te geven met de Hanepoot notatie (in het Engels: Crow's Foot Notation) omdat het een gangbare notatie is en ik de symbolen die je daarbij kan gebruiken heel intuïtief vind. Zie "Figuur 5: Hanepoot notatie van de ERD" voor een legenda van deze notatie.



Figuur 5: Hanepoot notatie van de ERD



Figuur 6: ERD versie 7

Er zijn hier een aantal interessante opmerkingen bij te maken die ik nu zal toelichten.

Merk bijvoorbeeld op dat een template een andere template als parent kan hebben en er hier sprake is van een veel-op-veel relatie naar zichzelf.

Ook zijn de entiteiten `Assignment`, `Checklist` en `Note` niet onafhankelijke entiteiten omdat hierbij verplicht andere entiteiten aan gekoppeld moet worden. Dit is het geval omdat zonder deze gegevens de entiteit niet meer van toegevoegde waarde zou zijn voor de applicatie en dus niet een logische waarde is.

Merk vervolgens op dat alle entiteiten een `created_at` en `updated_at` veld hebben. Omwille van de overzichtelijkheid van het diagram heb ik deze velden voor het gemak even weggelaten. Deze velden worden geïmplementeerd door gebruik te maken van de `TimestampableEntity` *trait* waar ik in hoofdstuk “7.1 Gekozen libraries” verder op in ga.

Alleen bij de entiteiten `Note` en `Assignment` leg ik door middel van de velden `created_by` en `updated_by` vast wie een rij heeft toegevoegd of voor het laatst heeft aangepast. Je zou kunnen zeggen dat het de traceerbaarheid van wijzigingen in de database ten goede komt door dit overal in te zetten. Toch heb ik er voor gekozen dit niet te doen en mijn keuze hiervoor is dat ik voor de andere entiteiten niet van plan ben deze via het prototype te tonen aan de eindgebruiker. Alleen wanneer men direct in de database zou kijken om dit op te zoeken zou dit dan op te zoeken zijn, maar ik kan mij geen reële situatie bedenken waarbij dit zou kunnen gebeuren en dus zou dit een overbodig veld zijn. Voor de entiteiten die deze velden wel hebben worden deze geïmplementeerd door gebruik te maken van de `BlamableEntity` *trait* waar ik in hoofdstuk 8.1 ook verder op in ga.

Zoals te zien wordt er zowel bij `User` als bij `Task` en `Assignment` gebruik gemaakt van de enum veld `expertise`. Enum, wat staat voor enumeratie, is een datatype waarbij in de configuratie van de database een lijst van mogelijke waarden wordt vastgelegd. Hierdoor is het niet mogelijk dat een gebruiker meerdere expertises tegelijkertijd toegewezen krijgt. Hoewel dit op zicht wel een logische mogelijkheid zou zijn, heb ik er voor gekozen dit toch niet zo te modelleren omdat het toch vaak voorkomt binnen Thirdwave dat een collega zich in één bepaalde expertise heeft gespecialiseerd. Daarnaast is een belangrijk gegeven dat dit veld alleen een voorkeur vastgelegd en dus nooit een gebruiker ergens in limiteert. Iedere gebruiker kan nog steeds alle handelingen uitvoeren ondanks zijn ingestelde expertise, en dit was ook een belangrijke mogelijkheid welke in de requirementsanalyse naar boven kwam.

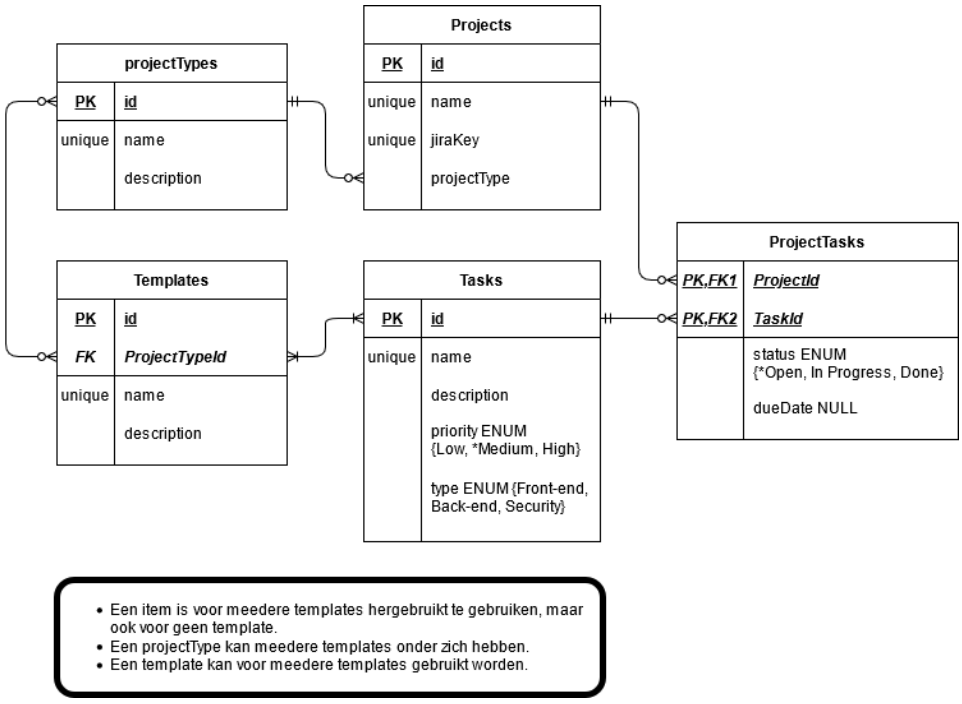
Ten slotte heb ik er bij de entiteit `Task` er bewust voor gekozen om de naam uniek te maken. Dit doe ik om aan de gebruikers van het systeem mee te geven dat de naam de taak volledig zou moeten omschrijven. In de praktijk zal blijken of deze keuze door de gebruikers niet te beperkend wordt ervaren. De productevaluatie is te lezen in hoofdstuk “8.1 Productevaluatie”.

6.1.1 Wijzigingen aan de hand van voortschrijdend inzicht

Bewust heb ik het hier gehad over versie 7 van het ERD. Er is namelijk meerdere keren iets in gewijzigd op basis van feedback uit de gehouden pilots. De belangrijkste veranderingen in het diagram zal ik kort benoemen en de vorige diagrammen zijn in de figuren 7 en 8 te bekijken.

De grootste wijziging is het feit dat ik initieel een extra entiteit had gemodelleerd. Deze entiteit heette `ProjectTypes` welke ik later had hernoemd naar `TemplateGroup` en had als doel een verzameling templates te ondervangen. Echter bleek het met deze opzet niet mogelijk om direct een `Template` onder een `Project` te hangen, wat het minder flexibel maakte. In het nieuwe ontwerp is dit wel mogelijk door het gebruik van een veel-op-veel relatie met zichzelf bij de entiteit `Template`.

Tenslotte was de naam `ProjectTasks` als naam voor een entiteit wat verwarrend en dus heb ik deze hernoemd naar `Assignments`.

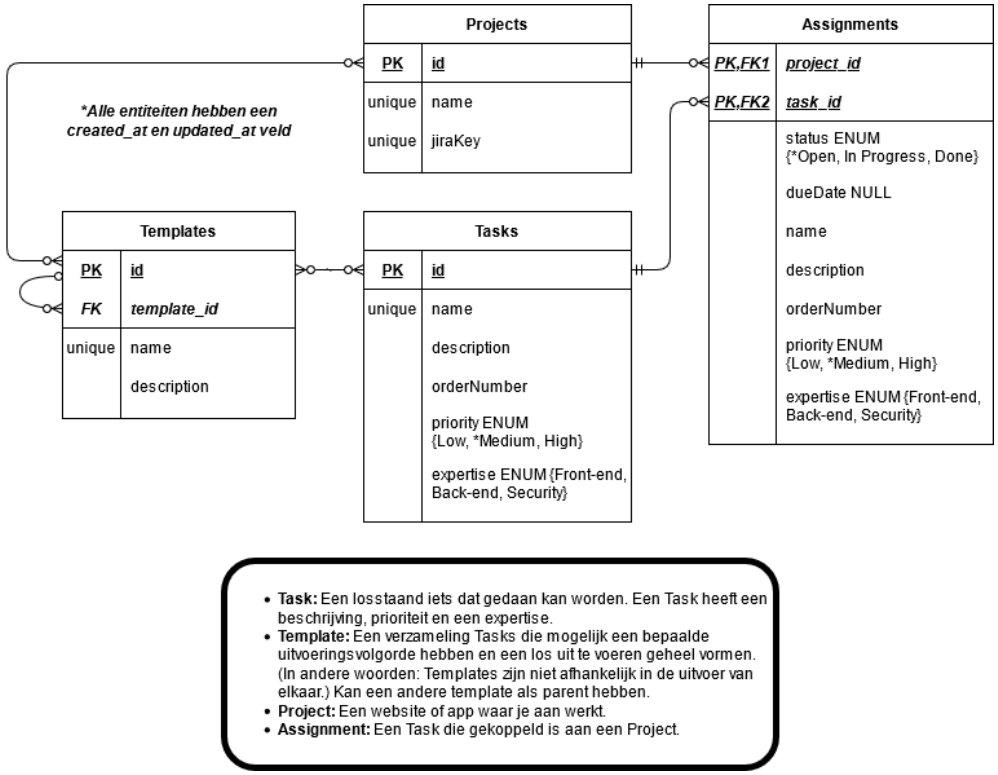


Figuur 7: ERD versie 1

Hieronder in figuur 8 is versie 5 van het ERD te zien. De voorgaande genoemde wijzigingen zijn hierin al verwerkt en de grootste verschillen tussen dit ontwerp ten opzichte van het uiteindelijke ontwerp is de toevoeging van entiteiten en databasevelden.

Hierbij wil ik nog even specifiek benoemen dat de entiteit `Checklist` hierna is toegevoegd. Het nadeel van het ontwerp in ERD versie 5 is namelijk dat er een directe koppeling van Assignments naar Projects is wat het niet mogelijk maakt om per project meerdere af te werken lijsten tegelijkertijd te hebben. Hoewel het te verwachten is dat het tegelijkertijd hebben van twee checklists niet voor zal komen, is deze aanpassing wel erg handig. Bij de release van een nieuwe versie van een project wil je namelijk wel opnieuw een checklist kunnen doorlopen zonder de vorige checklist te moeten verwijderen.

In het laatste en definitieve ontwerp zijn overigens alle entiteiten in het enkelvoud aangeduid. Dit vind ik een verbetering omdat uiteindelijk ook de modellen in de code en ook de tabellen in de database deze enkelvoudige naamgeving aanhouden.



Figuur 8: ERD versie 5

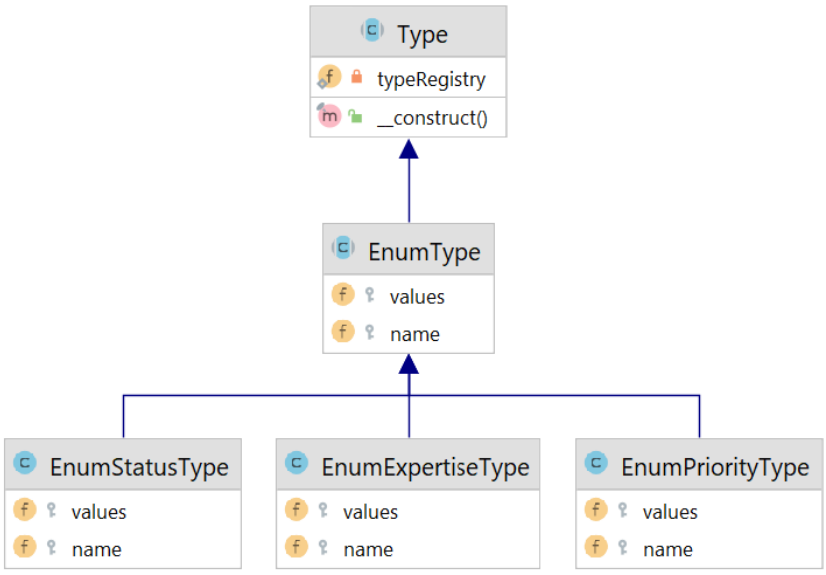
6.2 UML diagram

In dit hoofdstuk zal ik door het gebruik van UML diagrammen wat meer inzicht geven in de samenhang en daarmee de structuur van de verschillende objecten in de code. De diagrammen maken het ontwerp visueel wat mij op deze manier beter in staat stelt het ontwerp naar anderen te communiceren en het brengt de mogelijkheden tot het gebruik van abstracties en *best practices* aan het licht.

De diagrammen zijn gerealiseerd met gebruik van een tool van mijn editor PHPStorm. Ik hanteer bij het maken van de diagrammen geen vast format en bepaal steeds zelf welke informatie ik aan de diagram toevoeg aan de hand van de situatie.

6.2.1 UML van Enum

Zoals af te lezen in het ER diagram maak ik gebruik van ENUM types om een vaste lijst aan mogelijkheden in de database te configureren. Om dit op een generieke en uitbreidbare manier te realiseren heb ik een abstracte klasse genaamt `EnumType` geïmplementeerd, welke een kind is van de abstracte Doctrine klasse `Type`. In deze klasse heb ik een paar methoden en *properties* staan waar ik vervolgens handig gebruik van kan maken in de concrete kind-klassen. Dit alles is af te lezen in figuur 9. De ENUM types zijn hierdoor heel eenvoudig op slechts één plek aan te passen en wordt vervolgens in de gehele applicatie hergebruikt.

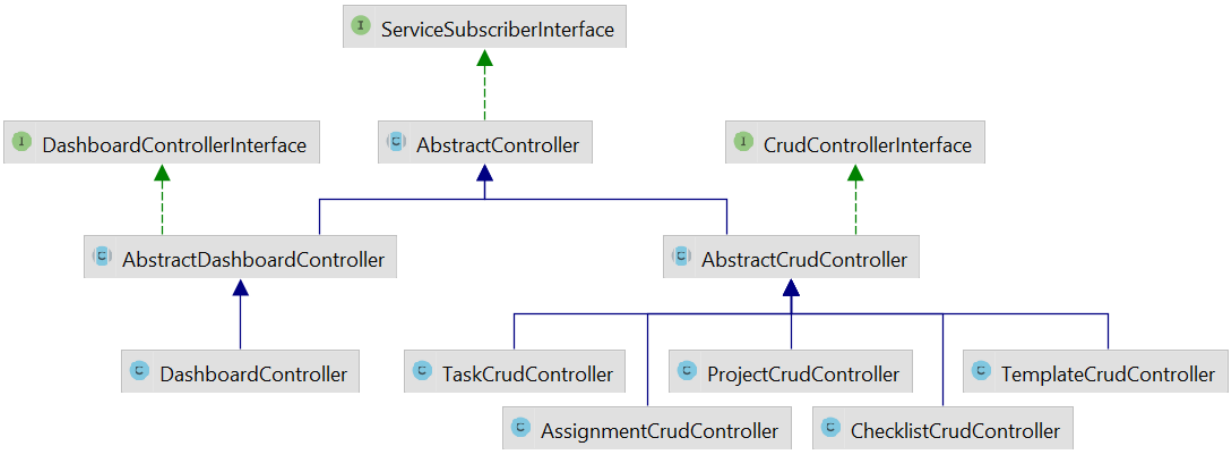


Figuur 9: UML van DBAL EnumTypes

6.2.2 UML van EasyAdmin

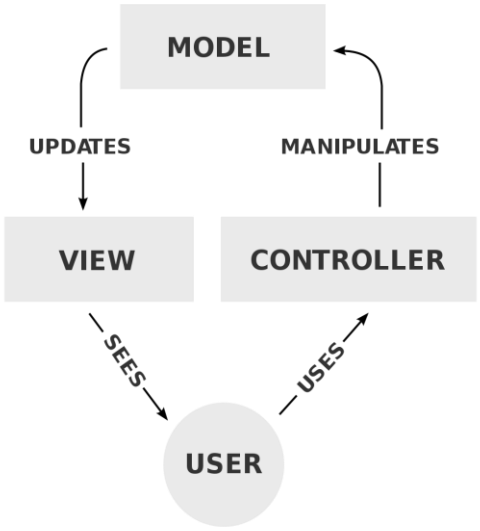
In een later stadium van het ontwikkeltraject heb ik gekozen voor het gebruik van de library genaamd EasyAdmin. In hoofdstuk “7.1 Gekozen libraries” ga ik verder op deze keuze in. In dit hoofdstuk wil ik graag benoemen dat de entiteiten in combinatie met het gebruik van deze *library* de onderstaande figuur hebben gerealiseerd: “Figuur 10”.

Zoals te zien breng ik de entiteiten in verschillende Controllers onder. De *Crud* entiteiten zijn ondergebracht onder de `AbstractCrudController`. Daarnaast maak ik ook gebruik van een `DashboardController` om verschillende zaken zoals de invulling van de dashboard pagina en het menu te configureren. Alleen de concrete klassen heb ik zelf moeten programmeren en de rest van de klassen en interface worden door de library voorzien. Dit vind ik de kracht van de library, want hierdoor kan er veel sneller een complete admin gedeelte worden gerealiseerd.



Figuur 10: UML van Admin Controllers

6.3 Architectuur



Figuur 11: Model-View-Controller Architectuur

Zoals eerder vermeld bouw ik het prototype met behulp van het Symfony framework. Dit framework maakt gebruik van een MVC architectuur. MVC staat voor *Model-View-Controller*. In figuur 11 is een globaal overzicht te zien van hoe dit werkt. De *model* bestaat uit objecten die de data vasthouden en de data kan hieruit worden opgevraagd in de *view* om deze aan de gebruiker te tonen. Wanneer de gebruiker een *request* aan het systeem maakt, wordt dit opgevangen en verwerkt door de *controller*. Via de controller is het op deze manier mogelijk de data de manipuleren.

Het voordeel van dit *pattern* is dat de code makkelijker te bouwen en te onderhouden wordt. Ieder stukje code is namelijk te isoleren en heeft slechts één verantwoordelijkheid. Het is op deze manier ook gemakkelijker om een stukje code te vervangen mocht dat nodig zijn, omdat het het mogelijk maakt abstracter en dus meer generiek een oplossing te implementeren. Symfony is een framework die voor het gebruik van MVC is gebouwd en het erg gemakkelijk maakt hierin te programmeren. Dit realiseert Symfony door configuratie gemakkelijk te maken en de verbindende stukjes code die voor ieder project hetzelfde zijn al te hebben geschreven.

Dit is niet de enige design pattern die aanwezig is binnen Symfony, maar wel de belangrijkste omdat dit *pattern* de structuur van de code voor het grootste gedeelte bepaalt. Vandaar dat ik de MVC structuur liever een architectuur noem dan een *pattern*. De M, de V en de C van MVC zal ik wel in het volgende subhoofdstuk verder in detail uitleggen, hoewel deze niet voorkomen in de klassieke *patterns* van de Gang of Four (GoF).

Van oorsprong is PHP geen object georiënteerde taal (OO), maar inmiddels is er wel een goede OO implementatie in te vinden. Voor een goede implementatie van MVC is een OO eigenlijk wel noodzakelijk omdat het ondersteund in het afbakenen van code (*encapsulatie*), het de code leesbaarder maakt en het helpt voorkomen van code duplicatie.

6.4 Design Patterns

Tijdens het realiseren van het prototype heb ik meerdere design patterns gebruikt om de code herbruikbaar en onderhoudbaar te maken en code duplicatie te voorkomen. De meest interessante patterns benoem ik hieronder.

Dit eerste drie *patterns* zijn zo fundamenteel voor webapplicaties in Symfony dat het bijna niet kan dat je deze *patterns* niet gebruikt. En dat wil je ook niet, aangezien Symfony hier veel functionaliteiten om heen heeft gebouwd.

6.4.1 Model pattern (de M van MVC)

Dit *pattern* heeft als doel data te manipuleren. In het geval van het prototype is er een PostgreSQL database op versie 13 die wordt benaderd door het gebruik van de library Doctrine. Doctrine is een Object Relational Mapper (ORM) en verzorgt de vertaling van data van PHP naar de database en terug. In de broncode van het prototype definieer ik in entiteit-objecten de tabellen en hun relaties, en vervolgens werk in in de *repository* objecten de database queries voor de database uit. De queries schrijf ik aan de hand van DQL (Doctrine Query Language); een dialect van SQL

Deze queries naar de database kan ik zowel via de `createQuery()` functie als via de `createQueryBuilder()` functie implementeren. Het eerste is gemakkelijker te lezen omdat het erg lijkt op SQL. Als je de query niet hoeft te wijzigen aan de hand van een set parameters, is dit waarschijnlijk de beste keuze. Query Builder is een api/interface om query's te maken. Het is makkelijker in gebruik als je een query dynamisch moet bouwen, zoals bij het itereren over een set parameters of filters. Bij de Query Builder hoef je geen string-bewerkingen uit te voeren om de query te maken; zoals samenvoegen, splitsen, enzovoorts.

Uiteindelijk leveren beide opties het hetzelfde resultaat op, en is het dus puur een keuze van de developer om te bepalen wat qua leesbaarheid in een specifieke situatie de voorkeur heeft. Belangrijk is de juiste use-case te identificeren om het jezelf makkelijker te maken.

6.4.1 View pattern (de C van MVC)

Dit *pattern* heeft als taak het genereren van iets aan de eindgebruiker. Dit kan een webpagina zijn, maar ook een PDF of zelfs een e-mail. Binnen Symfony is het gebruikelijk dit te realiseren door middel van de PHP pre-processor genaamd Twig, maar dit is geen vereiste.

Twig maakt het makkelijker en overzichtelijker om code te schrijven door gebruik van allerlei helper functies die de code korter en modulair maken. Daarnaast is het flexibel in het toevoegen van extra functies zoals ik ook heb gedaan voor het prototype met het object `ColorExtension` waar ik de filter `due_date_color` heb geïmplementeerd. Deze filter pakt de tekstuele representatie van een datum en geeft daarvoor een bijpassende kleur voor terug. Specifiek gebruik ik deze

6.4.1 Controller pattern (de C van MVC)

Dit *pattern* heeft objecten met het doel om verzoeken van de gebruiker te ontvangen en juist af te vangen. Dit kan zich uiten in het ophalen of zelfs veranderen van data om vervolgens een *view* terug te geven. De logica voor deze beslissingen wordt allemaal beschreven binnen deze *controllers* en ze bepalen zelfs zelf op welke verzoeken ze reageren.

Het handige van controllers in Symfony is de mogelijkheid van het zogenoemde “*autowiring*”. Hierbij kan ik *dependencies* automatisch definiëren aan de hand van de argumenten van de methoden. Door het gebruik van *type hinting* (het opgeven van een data-type voor een variabele) maak ik expliciet bekend welke services ik nodig heb en Symfony regelt de rest waardoor ik minimale configuratie nodig heb.

6.4.2 Factory pattern

Dit *creational pattern* maakt het mogelijk om met gemak meerdere objecten aan te maken zonder hierbij het exacte objecttype mee te geven en hier dus strikt aan verbonden te zijn. Het aanmaken van objecten is zo *decoupled* van de objecten zelf, wat in lijn staat met de SOLID principes. Dit implementeer ik in de `DataFixtures\BaseFixture` klasse met de `createMany()` methode en gebruik ik om de database gemakkelijk te laden met testdata. Deze testdata gebruik ik om de correcte werking van de applicatie te testen. Doordat ik het prototype de entiteiten met tientallen rijen vul is het gebruik van de *pattern* wel zo makkelijk en robuust. Ik heb namelijk minder rijen code nodig en kan minder makkelijk fouten maken.

6.4.3 Listener pattern

Dit *behavioural pattern* maakt het mogelijk om bepaalde code op het juiste moment voor de geabonneerde objecten uit te voeren. In het prototype maak ik gebruik van dit patroon tijdens het aanmaken van de `Assignment` entiteit. Om er namelijk voor te zorgen dat aanpassingen mogelijk blijven voor tasks zonder dat alle assignments hiermee direct retrospectief aangepast zouden worden, heb ik er voor gekozen de details van een assignment die uit een task zouden komen, tijdens het aanmaken van een assignment te kopiëren. Hierdoor blijft het zowel mogelijk generieke als specifieke details te hebben. Dit doe ik door middel van een zogenoemde “Lifecycle Event” en dan specifiek de “`PrePersist()`” event, welke alleen uitgevoerd wordt wanneer een nieuwe regel aan de database toegevoegd moet worden.

Een andere plek waar ik dit *pattern* toepas is die bij de zelf geïmplementeerde klasse `EasyAdminListener`. Zoals de naam wellicht doet vermoeden zorgt deze klasse er voor dat wanneer er iets aangepast wordt binnen het admin gedeelte er een zogenoemde *flash message* wordt gegenereerd. Een *flash message* is een bericht op een webpagina welke slechts eenmalig zich aan de gebruiker laat zien. De EasyAdmin *library* heeft er voor gekozen om standaard geen *flash messages* te genereren om de hoeveelheid berichten te beperken. Omdat ik echter expliciet aan de gebruiker wil tonen dat iets is uitgevoerd, heb ik deze *pattern* toegepast.

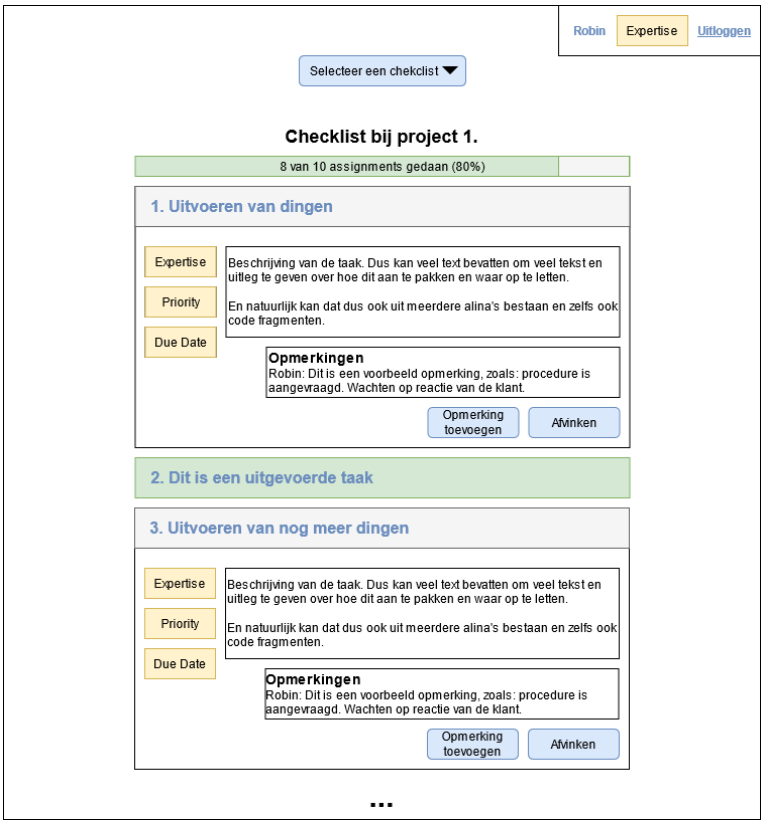
6.5 Wireframe

Doordat het voor de eindgebruiker belangrijk is een duidelijke en gebruiksvriendelijke applicatie te hebben, heb ik een front-end gerealiseerd. Voordat ik die user interface (UI) ging bouwen, heb ik deze eerst geschetst in een wireframe. Zie het ontwerp hieronder in figuur 12. Doordat de UI van het admin gedeelte al voor een grote gedeelte vast stond heb ik hiervoor geen wireframe gemaakt.

Doordat het hier gaat om een *wireframe*, geeft dit slechts weer welke elementen waar geplaatst. Dat betekend dat ik door middel van het gebruik van kleuren een bepaalde betekenis daar aan mee kan geven.

Toelichting bij de wireframe

- Met de blauwe kleur geef ik aan dat iets klikbaar is.
- Met de gele kleur geef ik aan dat hier door middel van het gebruik van kleuren de uiteindelijke waarde van dat onderdeel wordt aangepast.
- Met de groene kleur geef ik aan dat een element is die te maken heeft met het afvinken of zelfs al is gemarkeerd als gedaan.



Figuur 12: Wireframe Checklist

7 Prototype

Met behulp van het voorwerk dat gedaan is in de vorige hoofdstukken kon ik vervolgens een prototype bouwen. BIj het bouwen van het prototype is er gebruik gemaakt van het *PHP backend framework* genaamd Symfony, waarmee dynamische websites gebouwd kunnen worden.

Op basis van de opgestelde requirements heb ik in de uitwerking van het prototype keuzes gemaakt in het gebruik van bepaalde *libraries*, welke ik in dit hoofdstuk verder zal toelichten. De code test ik welke ik daarna zal beschrijf en ten slotte deploy ik het geheel naar een testomgeving.

7.1 Gekozen libraries

Zoals eerder vermeld zijn er libraries gebruikt in het prototype. Binnen het Symfony Framework worden externe libraries beschikbaar gesteld door middel van bundles. In tabel 1 is de complete lijst van de gebruikte libraries te vinden.

Er is een belangrijke tweedeling in de gekozen libraries op te merken. Het is namelijk zo dat sommige libraries alleen op de development omgeving van het project nodig zijn. In dat geval zorg ik er voor dat deze libraries om performance en security redenen alleen op de development environment te gebruiken zijn door deze aan het project toe te voegen met de `--dev` optie.

De volgende belangrijke gekozen libraries licht ik kort toe.

7.1.1 easycorp/easyadmin-bundle

De reden voor het kiezen van deze bundle is het feit dat er snel en gemakkelijk een complete CRUD gerealiseerd kan worden. Met CRUD bedoel ik de vier basis operaties op duurzame gegevens, te weten: Create, Read, Update en Delete. Dat is fijn, want hierdoor is het prototype snel beschikbaar. Nadeel is dat er in mindere mate veranderingen aan de UI toegepast kunnen worden. Dit betekent voor het prototype vooral dat het gedeelte via de *bundle* minder gebruiksvriendelijk is.

7.1.2 stof/doctrine-extensions-bundle

Door het gebruik van deze bundle kan ik dankbaar gebruik maken van verschillende doctrine *traits*. Belangrijkste hierbij zijn de *traits* `TimestampableEntity` en `BlameableEntity` waardoor ik niet opnieuw deze functionaliteit opnieuw uit hoef te vinden. Dit voegt namelijk door middel van event listeners de juiste afhandeling van de databasevelden toe die bijhouden wanneer en door wie een entity is aangemaakt en aangepast. De noodzaak voor het bijhouden van deze databasevelden kwam voort uit de requirements.

7.1.3 Niet gekozen library: symfony/workflow

Hoewel ik de library symfony/workflow bewust niet heb gekozen wil ik deze library toch graag benoemen omdat het een belangrijke keuze is geweest. Deze library (of component) wordt namelijk gebruikt om een eindigetoestandsautomaat te implementeren. Dit modelleert het gedrag van een systeem waarbij het model bestaat uit een eindig aantal toestanden, overgangen tussen die toestanden en acties.

Je zou kunnen denken dat voor het prototype dit gebruikt zou kunnen worden om de volgorde van uitvoer in de Assignments te implementeren en zo af te dwingen. Doordat het echter voor het prototype van belang is dat deze heel flexibel is opgesteld, is deze optie juist niet van belang en dus niet van toegevoegde waarde.

7.2 Unittesten

De code van het prototype heb ik voorzien van *unittesten*. Deze *unittesten* stel ik op door middel van de *library* PHPUnit, welke een veel gebruikte *library* is en een uitstekende integratie biedt met het Symfony framework.

Voor het handig testen maak ik waar mogelijk gebruik van een *data provider* methode. Dit zorgt er voor dat ik in de ene methode de te testen data kan opgeven, en vervolgens in een andere methode over deze data kan itereren om deze stuk voor stuk te testen op de juiste werking. Een voorbeeld hiervan is de `getJiraKeyProjectNameTests()` *provider* waarbinnen ik Jira keys opgeef welke op de juiste manier gevalideerd moeten worden. Dit wordt vervolgens gedaan in `testJiraKeyProjectName()`.

Andere onderdelen die ik door middel van PHPUnit test is mijn Twig *extension*, de Doctrine Enum Type klassen, en de verschillende services.

In “Figuur 13: PHPUnit Testing” laat ik een screenshot zien van het resultaat wanneer ik handmatig het *unittest* script uitvoer.

```
PS C:\development\projects\preflight> symfony php bin/phpunit
PHPUnit 8.5.15 by Sebastian Bergmann and contributors.

Testing Project Test Suite
............                                      12 / 12 (100%)

Time: 254 ms, Memory: 6.00 MB

OK (12 tests, 12 assertions)
```

Figuur 13: PHPUnit Testing

7.2.1 Code Coverage

Bij het schrijven van de tests let ik er op dat ik de verschillende onderdelen van mijn code allen benader. Echter heb ik er omwille van de tijd nog geen *code coverage runner* op los gelaten. Het voordeel van zo’n *runner* is dat deze aangeeft hoeveel procent van de code daadwerkelijk getest wordt. Dit geeft extra informatie over de staat van de tests die in het project aanwezig zijn en daarmee meer zekerheid over de correcte werking van het prototype. In het vervolg ontwikkeltraject van het prototype zal ik hier verder aan werken om deze toe te voegen.

7.3 Realisatie front-end

Voor een hoge mate van gebruiksvriendelijkheid voor de eindgebruiker heb ik een front-end gerealiseerd. Hier heb ik verschillende technieken gebruikt die er samen voor zorgen dat ook in deze code de beste practices worden gehandhaafd om de code onderhoudbaar te maken.

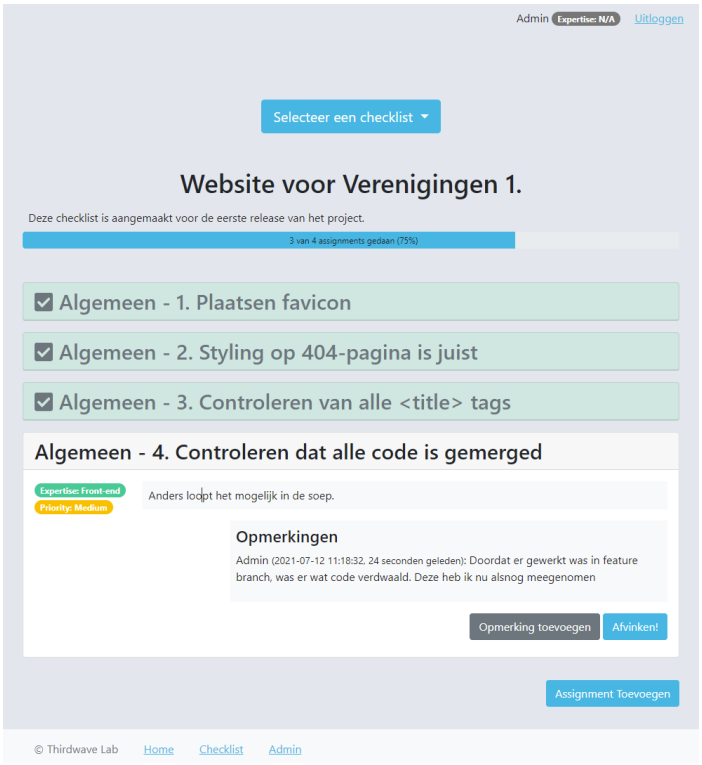
Zoals eerder gezegd bouw ik de templates met behulp van de template engine Twig. In deze templates maak ik handig gebruik van `blocks` om op die manier code her te gebruiken. De verschillende templates orden ik vervolgens ook via een overzichtelijke mappenstructuur waardoor ik snel kan vinden waar welke template voor gebruikt wordt. Gedeeltes van een pagina verzamel ik in zogenoemde `partials`.

Voor het inladen van de *stylesheets* en *javascripts* maak ik gebruik van de library `symfony/webpack-encore-bundle`. Deze library comprimeert mijn code en voegt alle losse bestanden samen tot een enkel bestand die vervolgens aan de gebruiker van de website getoond wordt. Bovendien schrijf ik de styling in SASS (dit staat voor Syntactically Awesome Style Sheets) en dit is een styling format die nog gecompileerd moet worden tot normale CSS (dit staat voor Cascading Style Sheets). Webbrowsers verstaan namelijk alleen CSS, maar als developer maak ik graag gebruik van SASS om efficiënter code te kunnen schrijven.

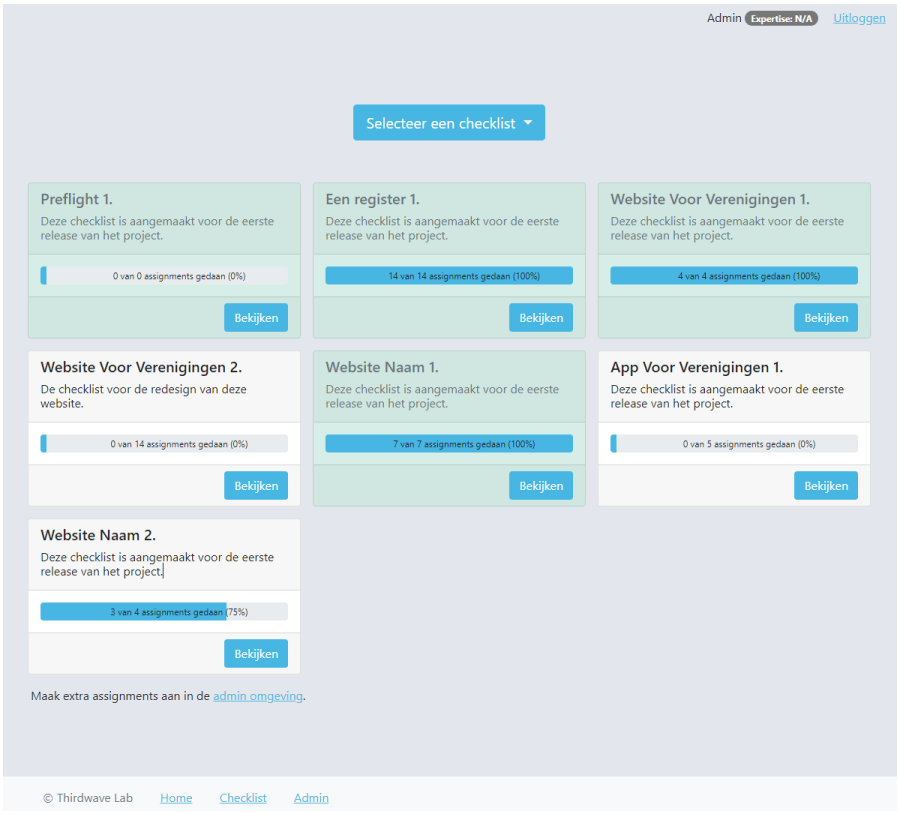
Om de styling nog makkelijker in elkaar te zetten maak ik gebruik van het front-end framework Bootstrap 5. Dit moderne en populaire framework stelt handige componenten beschikbaar welke ik kan gebruiken in de applicatie. Voorbeeld hiervan zijn de `modal`, `card` en `progress` componenten. Uiteraard heb ik de styling volledig responsive gemaakt. Dat wil zeggen dat de website ook op mobiel goed te gebruiken is. Bootstrap 5 ken daarnaast ook variabelen welke ik heb uitgebreid met mijn eigen variabelen. Op deze manier heb ik bijvoorbeeld de specifieke kleuren die worden gebruikt voor de *expertise* en *priority* op slechts één plek gedefinieerd.

Ten slotte wil ik benoemen dat ik mijn styles heb onderverdeeld in een *base* bestand, een *components* bestand en een *layout* bestand. Dit was handig om de code modulair en gestructureerd op te zetten. Omdat de applicatie niet nóg groter was, heb ik er voor gekozen om niet te gaan voor de *7-1 pattern* welke bij grote SASS applicaties wordt aangeraden.

Op de volgende pagina staan twee screenshots van de meest gebruikte pagina's in de front-end in de figuren 14 en 15 om een indruk te geven van de front-end die ik heb gerealiseerd.



Figuur 14: Screenshot Preflight Checklist



Figuur 15: Screenshot Preflight Overzicht

7.4 Deploy naar testomgeving

De ontwikkelingen aan het Preflight project worden gedeployed naar een testomgeving. Hiervoor heb ik een ontwikkelstraat in GitLab opgesteld welke is geconfigureerd door middel van een `gitlab-ci.yaml` bestand binnen het project. Deze configuratie is zo ingesteld dat wanneer er een *commit* (een aanpassing in de code) wordt gedaan naar de testomgeving deze automatisch een *gitlab runner* zal aansporen om allerlei scripts uit te voeren die de code eerst testen en vervolgens de code wegschrijft op een testomgeving.

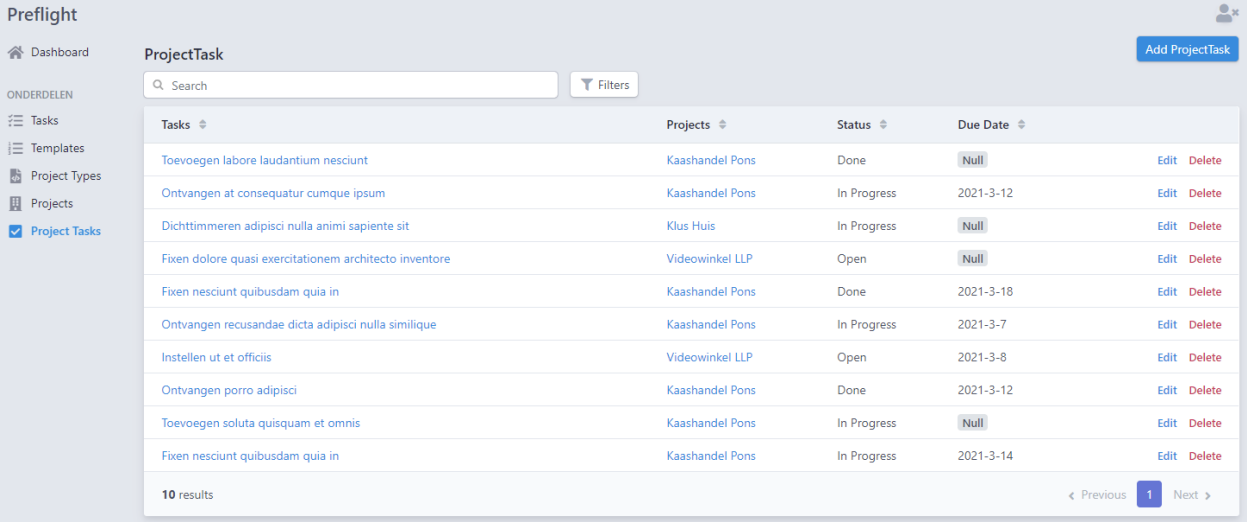
Onderdeel van het automatisch deployen is het aanmaken van een `.env.local` bestand die de database configuratie aan het project wegschrijft. Vervolgens wordt er een `composer install` gedaan om alle dependencies binnen te halen. Daarna worden aanpassingen in de database doorgevoerd en indien nodig de data binnen de tabellen geüpdate. Ook voer ik nu een scriptje uit genaamt “npm” om de styling en javascript te installeren. Tenslotte wordt de *cache* van de website verwijderd en opnieuw opgebouwd zodat alle wijzigingen te zien zijn.

Ook wordt er tijdens het deployen specifiek getest op eventuele beveiligingsfouten omdat deze testen makkelijk te automatiseren zijn en een belangrijke eis zijn voor sommige projecten binnen Thirdwave. Ook de PHPUnit test zoals beschreven in hoofdstuk “8.2 Unittesten” wordt hier automatisch uitgevoerd zodat dit niet kan worden vergeten. Dit is belangrijk omdat hiermee de kwaliteit van de code kan worden gegarandeerd.

Bij andere projecten binnen Thirdwave worden wel eens nog meer scripts uitgevoerd. Denk hierbij aan linters en scanners die de code van het project inspecteren. Bij dit project zou ik deze additionele scripts in een later stadium nog willen toevoegen maar helaas heb ik daar tot nu toe nog geen tijd voor gehad. Deze scripts zouden de code naar een nog hoger niveau kunnen tillen, maar door het gebruik van de hiervoor beschreven tests is dit voor nu alvast voor het prototype naar een acceptabel niveau gebracht. Bovendien wil ik opmerken dat de het programma waarbinnen ik en mijn collega’s code schrijven al zelf intern een linter hebben, waardoor de noodzaak voor een additionele linter lager ligt. Op dit moment is het echter zo dat er gecodeerd wordt in verschillende *editors* met mogelijk een andere configuratie waardoor niet alles (op de juiste of dezelfde manier) wordt gecontroleerd.

7.5 Pilot & Doorontwikkeling

Zodra een eerste prototype werkzaam was werd deze in gebruik genomen bij een te beginnen project om zo de geïntegreerde tool in de complete werkomgeving te gebruiken. De keuze viel hierbij op projecten met een korte doorlooptijd van maximaal een paar weken om korte-cyclische processen aan te houden. Een screenshot van hoe het prototype er in deze eerste versie uitzag is te zien in figuur 16.



Figuur 16: Screenshot Preflight Pilot

Tenslotte zijn de bevindingen van het testen gebruikt als input voor doorontwikkeling, om zo te verbeteren wat er goed ging en wat er beter kon. De bevindingen werden op deze manier verwerkt in een nieuwe versie van het prototype. Dit is expliciet ook met de opdrachtgever en de bedrijfsmentor besproken om alles goed af te stemmen.

7.5.1 Usability Tests & Code Reviews

Samen met een collega heb ik de applicatie doorlopen en deze uitgebreid besproken. Tijdens deze sessies ben ik mee gaan kijken bij mijn collega om samen het prototype te doorlopen. Door een gesprek aan te gaan bij de verschillende handelingen die gedaan moeten worden bij de uitvoer van een checklist komen er verwachtingen en wensen naar voren. Deze verwachtingen kon ik vervolgens verwerken in een nieuwe versie van het prototype. Dit kwam de kwaliteit en gebruiksvriendelijkheid van het prototype ten goede.

Tijdens de Code Reviews ben ik ook samen met mijn bedrijfsmentor door de code gelopen om mogelijke verbeteringen in de code te lokaliseren. Zie hiervoor de bijlage “7. Code Reviews”.

8 Evaluatie

In dit hoofdstuk evalueer ik op het product alsmede het proces. Ik geef aan wat goed ging en waar de verbeterpunten liggen. Op deze manier geef ik inzicht in het resultaat van mijn werkzaamheden, maar ook mijn persoonlijke inzicht in mijn eigen functioneren en leerproces.

8.1 Productevaluatie

Een van de opgeleverde deelproducten van deze afstudeeropdracht is het prototype. Hoewel het een vereiste requirement vanuit Thirdwave was om er een Symfony applicatie van te maken, zou ik zelf geen andere keuze gemaakt hebben. Symfony is een modern en krachtig framework welke ik ervaar als prettig mee te werken. Met oog op de toekomst wil ik mij hier komende jaren verder in doorontwikkelen.

Het prototype is in verschillende stadia in gebruik is genomen om op iteratieve wijze feedback te vergaren, maar dat wil niet zeggen dat er tijdens het ontvangen van de feedback niets meer te melden was. Uit de verschillende feedbackrondes zijn verschillende wijzigingen in het prototype uit gekomen welke op dit moment geïmplementeerd zijn. Bijvoorbeeld het toevoegen van de entiteit `Checklist`, maar ook de mogelijkheid tot het plaatsen van opmerkingen bij een `Assignment` en het door admins geheel af kunnen vinken van een `Checklist`. Het uniek maken van de naam van een `Task` werd niet als te beperkend bevonden en ontving zelfs waardering door mijn collega's. Door het uitvoeren van deze feedbackrondes weet ik dat het prototype naar behoren werkt, zijn mijn collega's meegenomen in het process, en werden eventuele bugs er gelijk uitgehaald.

Hoewel ik van de opgestelde requirements alle *Must Haves* en *Should Haves* heb kunnen realiseren, heb ik een aantal *Should Haves* niet kunnen realiseren. Belangrijkste hiervan is het laten uitvoeren van een automatische controle op een taak voor extra gebruiksgemak en tijdswinst. Maar ook het kunnen hangen van een taak aan een specifiek persoon om uit te voeren zodat verantwoordelijkheden expliciet kunnen worden uitgegeven. Doordat het slechts gaat om een handje vol *Should Haves* wordt het prototype wel als compleet en bruikbaar ervaren.

In sommige gevallen heb ik ook concrete verbeteringen aan de code (zoals performance verbeteringen) in de code aangegeven door middel van commentaar met het woord "`TODO`". Doordat mijn editor (en die van mijn collega's) hier op reageert en deze verzameld is het heel makkelijk deze openstaande punten te vinden. Uiteraard heb ik deze punten ook in Jira in de

backlog toegevoegd zodat deze nog minder de kans hebben vergeten te worden. Dit lijkt misschien enigszins dubbelop, maar dat is het niet omdat de "`TODO`" handig is voor de developer terwijl de Jira ticket in de *backlog* handig is voor de project manager. Ten slotte wil ik toevoegen dat het vermelden van een "`TODO`" alleen zin heeft als het uitvoeren van dat punt niet een hoge urgentie heeft. In dat geval is het namelijk raadzaam het punt direct uit te voeren.

Na deze afstudeeropdracht zal het prototype intensiever in gebruik genomen worden en verder worden ontwikkeld. Ik ben erg nieuwsgierig naar de toekomstige ervaringen met het gebruik van het prototype en kan me heel goed voorstellen dat er nieuwe wensen zullen ontstaan. Mijn verwachting is dat deze nieuwe wensen zich vooral richten op het verder gebruikersvriendelijk maken van het prototype. Dit kan gerealiseerd worden door het toevoegen van automatische controles welke dan ook automatisch de assignments hierop aanpassen.

8.1.1 Productevaluatie door opdrachtgever

De opdrachtgever geeft aan dat het prototype er goed uitziet en voldoet aan de verwachtingen. Daarnaast wordt door hem de code als onderhoudbaar ervaren door de juiste structurering van de code en het gebruik van best practices. Hij heeft aan dat ik veel verschillende componenten heb gebruikt en daardoor heb ik veel van de belangrijkste en veel voorkomende componenten gezien en juist toegepast. Dit is een goede voorbereiding op toekomstige werkzaamheden.

8.2 Reflectie beroepstaken

Met het uitvoeren van dit afstudeerplan kan ik de hierna te noemen beroepstaken aantonen. De met een ster gemarkeerde beroepstaken zijn verplichte beroepstaken die horen bij dit afstudeerprogramma.

*A1; Analyseren probleemdomein & opstellen probleemstelling

Deze beroepstaak kwam vooral aan bod bij het uitvoeren van de requirementsanalyse voor het prototype. De huidige situatie is grondig in kaart gebracht zodat het prototype het juiste probleem op de in de context van deze afstudeeropdracht het op het beste manier verhelpt. Dit werd gedaan door het houden van interviews met de belanghebbenden.

Product(en):

- **Requirementsinventarisatie op basis van interviews:** Zie hiervoor het document Requirements Interviews.
- **Requirementsanalyse op basis van interviews:** Zie hiervoor het document Requirements Interviews.

Voetnoot: Wanneer ik een volgende keer de requirements zou verzamelen, vind ik het interessant om te onderzoeken of bijvoorbeeld het Bulk en Gedonder-kwadrant van Basile Lemaire een hulpmiddel kan zijn in het verkrijgen van requirements om te tekortkomingen van de MoSCoW methode te ondervangen.

*C1; Ontwerpen software

Het prototype ontwierp ik op basis van de opgestelde requirements in een ER diagram die gedurende de looptijd van het afstuderen steeds aangepast wordt en dus als levend document beschouwd is geweest. Er werd hierbij object georiënteerd te werk gegaan gebruikmakend van een MVC architectuur. Waar nodig werden er ook *design patterns* toegepast, zodat het prototype makkelijk uitbreidbaar blijft.

Product(en):

- Zie hiervoor de vastlegging in hoofdstuk 6 inclusief de verschillende diagrammen die binnen dit hoofdstuk te vinden zijn.

*D1; Realiseren van software

Het prototype, welke de vorm heeft van een dynamische website, werd gerealiseerd binnen het PHP Symfony framework op basis van het ontwerp. Waar mogelijk werd er gebruik gemaakt van externe *libraries*, wat als doel had sneller te kunnen ontwikkelen en niet zelf helemaal opnieuw het wiel uit te hoeven vinden.

Product(en):

- Zie hiervoor de vastlegging in hoofdstuk 7 inclusief de verschillende afbeeldingen die binnen dit hoofdstuk te vinden zijn.

D2; Testen & Evalueren

In het domein van de probleemstelling stelde ik voor het ontwikkelen van het prototype een adequaat plan op en toetste ik deze aan de opgestelde *requirements*, welke werden vastgelegd in een *review* van ontwerpdocumenten. Ook werden er *unittest* aan het prototype toegevoegd zodat de correcte werking van geïsoleerde stukjes code kon worden getest. Op deze manier werd met behulp van een acceptatietest de ervaringen van het prototype juist en compleet vergaard welke het ontwikkelproces ondersteund.

Product(en):

- **Review van ontwerpdocumenten uit beroepstaak C1:** Zie hiervoor hoofdstuk 6.1.
- **Unittesten:** Zie hiervoor hoofdstuk 7.2.
- **Acceptatietest:** Zie hiervoor hoofdstuk 7.4.

D4; Configureren

Deze beroepstaak heb ik ingevuld door het maken en toepassen van scripts om deployments van het prototype automatisch te realiseren. Ook heb ik Symfony geconfigureerd om het prototype op de juiste manier te laten werken. Tenslotte zijn de checklists en de andere gerelateerde data compleet configureerbaar door de gebruiker van het prototype. De verschillende projecten worden namelijk op verschillende wijzen vooraf geconfigureerd op basis van de gebruikte technologieën van het desbetreffende project.

Product(en):

- **Configureerbaar ontwerp:** Zie hiervoor hoofdstuk 6.
- **Prototype in PHP Symfony:** Zie hiervoor hoofdstuk 7.

*Gc; Kritisch, onderzoekend & methodisch werken

Om dit project tot een succes te brengen, breng ik veel helder in kaart. Door kritisch te zijn kan ik mij de beste oplossing voorstellen, en ontwikkel ik een oplossing voor het juiste probleem. Ik reflecteer op de voortgang door gebruik te maken van de STAR-methodiek. Een solide planning is van belang om niet uit te lopen in de tijd welke ik in Jira vastleg. Dit systeem maakt het mogelijk om onder andere taken uit te werken, de complexiteit ervan in te schatten en deze te plannen.

Product(en):

- **Planning in Jira:** Zie hiervoor figuur 7; Planning in Jira.

*Gf; Leren leren: voorbereiden op volgende studiefase & beroep

Er zijn door mij nieuwe en onbekende technieken aangeleerd, met name het gebruik van het backend framework Symfony. Deze kennis is erg relevant voor het vakgebied omdat het een veel gebruikt en gevraagd framework is, en werkt volgens algemeen geaccepteerde en bewezen design patterns (oa MVC) welke relevant zijn en blijven. Mijn bekwaamheid in deze technologie heb ik verhoogd door het volgen van de leertrajecten op *symfonycast.com*, welke ik met certificaten heb geformaliseerd maar niet heb verwerkt in het eindverslag omdat ik dit niet als product voor dit project hoef op te leveren. Om mijn professionele ontwikkeling verder vorm te geven, reflecteer ik op deze leerwijze en leg ik deze naast mijn leerstijl.

Product(en):

- **Leerstijl evaluatie:** De ideale leerstijl voor mij is het combineren van theorie en praktijk. Graag doe ik theoretische kennis op zodat ik weet waar ik het over heb en weloverwogen keuzes kan maken. Juist door deze kennis vervolgens toe te passen breng ik het in de praktijk en doe ik ervaring op hoe dit in zijn werk gaat in een project.

8.3 Algehele reflectie

Samengevat vind ik dit afstudeerproject succesvol verlopen. Zoals blijkt uit de verslaglegging in de vorige hoofdstukken heb ik veel gedaan en geleerd, en heeft mijn bedrijf een werkzaam prototype opgeleverd gekregen waar zij profijt van heeft. Ik ben blij dat mijn bedrijfsmentor deze mening met mij deelt.

In de planning heb ik moeten schuiven. Initieel was het plan om meer de nadruk te leggen op automatisering van controles. Ook had ik de optie om mij sterk verder te verdiepen in een integratie met de security tools waar de security officer mee kwam. Aan de hand van complexiteit en haalbaarheid binnen de duratie van de opdracht heb ik op juiste wijze invulling van de scope van de opdracht kunnen geven. Dit betekende in dit geval dat ik de implementatie van deze additionele functionaliteiten bij het prototype heb moeten uitstellen tot na mijn afstudeerproject. Deze functionaliteiten waren namelijk niet van cruciaal belang voor de werking van het prototype.

Ook moet ik toegeven dat ik niet optimaal gebruik heb gemaakt van het werken met sprints. Doordat Jira-tickets langer dan één sprint open stonden kon ik niet netjes een sprint afsluiten en met een volgend onderwerp verder gaan. Hierdoor werkte ik aan verschillende tickets en doelen tegelijkertijd wat op zich niet erg is maar wel de structuur of in ieder geval de overzichtelijkheid van de werkwijze verminderd. Om hier beter mee om te gaan heb ik tickets toch nog kleiner in scope moeten formuleren en eerder het resultaat van een ticket als voldoende bestempelen.

8.3.1 Tegenslagen

In het algemeen vond ik dit wel een erg zware afstudeeropdracht. Zoals eerder vermeld werd deze afstudeeropdracht tijdens de Corona pandemie uitgevoerd waardoor iedereen online samenwerkt. Hierdoor is het erg lastig snel en efficiënt met elkaar te communiceren wat juist tijdens een afstudeeropdracht belangrijk is.

Daarnaast viel mijn begeleidend examinator gedurende het traject uit, en ook viel mijn opdrachtgever gedurende het traject uit. Ook deze feiten hebben het uitvoeren van deze opdracht niet makkelijker gemaakt, maar gelukkig waren er binnen een paar weken vervangende mensen voor deze rollen geregeld.

De conclusie is dat hierdoor de kwaliteit en intensiteit van de begeleiding lager ligt dan anders het geval was. Er werd een groot beroep gedaan op mijn zelfstandigheid en zelfredzaamheid. Hoewel ik hier moeite mee heb, zou ik willen zeggen dat ik mezelf ermee voldoende uit de voeten heb gemaakt. Ik weet van mezelf dat dit niet in mijn persoonlijkheid zit en met oog op de toekomst

is het dus belangrijk een rol te kunnen vervullen die mij veel meer houvast geeft. Dit geeft mij rust en hierdoor komt mijn technische creativiteit veel beter tot zijn recht.

8.3.2 Evaluatie van persoonlijke werkwijze

Wanneer ik dit project een volgende keer zou uitvoeren, zou ik mijzelf nog zelfverzekerder op willen stellen zodat ik sneller en standvastiger beslissingen kan nemen. Dit kan beter als ik het bedrijf, de collega's en de techniek beter ken. Met meer ervaring zal dit makkelijker verlopen.

Onderdeel hiervan is dat het mij houvast geeft wanneer ik een bepaalde structuur aanbreng in mijn werkzaamheden en deze opdeel in kleinere stukken. Hierdoor weet ik beter waar ik vandaan kom en waar ik naartoe wil, en word ik minder overweldigd door een grote berg taken. Dit helpt mij enorm in het tackelen van mijn onzekerheid, en is van belang om optimaal te kunnen functioneren. Deze structuur moet ik zelf vormgeven omdat dit een persoonlijke behoefte is. Hoewel ik vind dat ik op dit punt nog wel veel te leren, vind ik dat ik hier gedurende het traject steeds beter een eigen draai aan kunnen geven door bijvoorbeeld het gebruik van de *roadmap* functionaliteit binnen Jira.

Daarnaast is het belangrijk om niet in details te verzanden waar ik soms nog wel eens vatbaar voor ben doordat ik perfectionistische neigingen heb. Gelukkig weet ik hier inmiddels goed mee om te gaan doordat ik mij kan vinden in de *agile* werkwijze. Door het aanhouden van een flexibele strategie en het gebruik van korte cycli, kan ik er vrede mee hebben dat niet alles gelijk perfect hoeft en kan ik het project in perspectief zien. Door structurele reflectie weet ik mij te verdedigen tegen de "*sunken cost fallacy*", wat voor mij betekend dat ik mij niet te lang stukbijt op iets waar ik op vastloop maar op tijd overschakel naar een andere optie. Het concept van iteratie spreekt mij hierbij erg aan. Deze werkwijze heb ik ook tijdens de uitvoering van deze afstudeeropdracht aan kunnen houden en dit heeft mij zeker geholpen het tot een succes af te ronden.

Met de term "technische creativiteit" zou ik mezelf echt willen samenvatten. Ik vind het namelijk erg leuk en zelfs belangrijk in mijn werk om met gebruik van mijn technische creativiteit iets te maken. Het geeft mij een goed gevoel ergens hard over na te denken en vervolgens de juiste implementatie te kiezen om een probleem op te lossen. Wanneer ik op die manier iemand structureel kan helpen geeft mij dat een ultiem gevoel van voldoening. Doordat het prototype bij Thirdwave structurele ondersteuning kan bieden heb ik dit positieve gevoel hierbij ook.

Begrippenlijst

Het jargon dat voorkomt in dit document wordt hieronder nader toegelicht ter ondersteuning van de duidelijkheid en begrijpelijkheid van de tekst. Merk op dat de begrippen handig op alfabetische volgorde staan.

Begrip	Betekenis
Bundle	Een bundel is symfony flex-abstractie voor het bieden van eenvoudige modulariteit, inclusief configuraties en automatiseringen.
CLI	Staat voor Command Line Interface; bekend als het zwarte schermpje waar programmeurs allemaal computer dingen in doen. Wordt vaak gebruikt tijdens development van applicaties door zijn uitgebreide mogelijkheden en precieze commando uitvoering.
CRUD	Met CRUD wordt er gedoeld op de vier basis operaties op duurzame gegevens, te weten: Create, Read, Update en Delete.
DBAL	DBAL betekent “Database Abstraction Layer” en zoals de naam doet vermoeden is dit een abstractie laag die verzorgd dat er gebruik gemaakt kan worden van een flexibele en intuïtieve API voor het communiceren met de database. Voor dit project gebruik ik de library genaamd Doctrine om deze functionaliteit te realiseren.
DoD	Een Definition of Done is een lijst met eisen waaraan een functionaliteit moet voldoen voordat deze kan worden afgestreept als gedaan. Dit begrip komt uit de Agile methodiek.
ERD	Een Entity Relationship Diagram waarbij “entiteiten” binnen een systeem met elkaar verbonden zijn. Dit wordt vaak gebruikt om relationele databases te ontwerpen.
GoF	Staat voor Gang of Four en refereert naar de vier schrijvers van het zeer invloedrijke boek “Design Patterns: Elements of Reusable Object-Oriented Software” uit 1994.
MoSCoW methode	Een prioriserings methode waarbij requirements worden ingedeeld in Must Haves (verplicht aanwezig voor de werking), Should Haves (niet vitale, maar wel belangrijke requirements die van significante waarde zijn), Could Haves (handige requirements die slechts van klein belang zijn) en Won’t Haves (requirements die niet oplevert zullen worden tijdens dit traject).
OO(P)	Staat voor Objectgeoriënteerd (Programmeren) en is een paradigma binnen programmeertalen waarbij code wordt geschreven aan de hand van objecten welke zowel data kunnen bevatten als data kunnen bewerken. In deze werkwijze kan men een bepaalde werking afschermen voor de buitenwereld en door het gebruik van overerving code hergebruiken zonder deze te dupliceren.

ORM	Dit staat voor Object Relational Mapping en is een techniek binnen de software engineering waarbij een OO object wordt gebruikt voor de vertaling van data. In het geval van het prototype is dit een vertaling van data in PHP naar data die de Postgresql database verstaat, en andersom.
OTAP	Dit staat voor OTAP: Ontwikkeling-Test-Acceptatie-Productie en hiermee worden de verschillenden omgevingen aangeduid in een ontwikkelstraat. Tijdens het gebruik hiervan gaat code van links naar rechts in dit acroniem en wordt de kwaliteit van de software gewaarborgd.
PoC	Staat voor Proof of Concept; een werkende prototype die de mogelijkheden demonstreert. Hierbij is het vooral belangrijk vast te leggen hoe het werkt en wat de complexiteit is van aanpassingen die in een vervolgstadium gemaakt kunnen worden. Het is voor een PoC nog niet belangrijk dat alles er gelikt uit ziet.
SASS	Dit staat voor Syntactically Awesome Style Sheets en is een styling format die nog gecompileerd moet worden tot normale CSS. SASS stelt de developer in staat sneller en modulairder styling te coderen.
SOLID	Dit acroniem is erg bekend binnen de object oriented software development en bestaat uit een vijftal regels die het waarschijnlijker maken dat de developer onderhoudbare en uitbreidbare code zal opleveren. `S` - Single-responsibility principle `O` - Open-closed principle `L` - Liskov substitution principle `I` - Interface segregation principle `D` - Dependency Inversion Principle
Trait	Een PHP-specifieke manier om horizontale compositie van code te realiseren. Dit is een handig mechanisme binnen PHP omdat de taal slechts overerving op één *parent* per object toelaat. Het biedt dus een granulaire en consistente manier om methoden tussen objecten met elkaar te delen.

Literatuurlijst

The 2020 Scrum GuideTM. Geraadpleegd in januari 2021.
https://scrumguides.org/scrum-guide.html

Symfony The Fast Track. Geraadpleegd op 10 februari 2021.
https://symfony.com/doc/current/the-fast-track/en/index.html

Symfony documentatie. (z.d.). Symfony. Geraadpleegd op 3 februari 2021.
https://symfony.com/doc/current/index.html

Symfonycasts. (z.d.). Symfonycasts. Geraadpleegd op 3 februari 2021.
https://symfonycasts.com/

Doctrine PHP Open Source Project. Geraadpleegd op 16 februari 2021.
https://www.doctrine-project.org/index.html

Using PHPLOC for quick code quality estimation door Matthias Noback. Geraadpleegd op 24 maart 2021.
https://matthiasnoback.nl/2019/09/using-phploc-for-quick-code-quality-estimation-part-1/

Best Practices - Doctrine Object Relational Mapper (ORM). Geraadpleegd op 2 april 2021.
https://www.doctrine-project.org/projects/doctrine-orm/en/2.8/reference/best-practices.html

Best Practices for Reusable Bundles. Geraadpleegd op 16 maart 2021.
https://symfony.com/doc/current/bundles/best_practices.html

Model-View-Controller Pattern. Geraadpleegd op 10 maart 2021.
https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

Factory method pattern. Geraadpleegd op 13 april 2021.
https://en.wikipedia.org/wiki/Factory_method_pattern

Observer Design Pattern. Geraadpleegd op 2 april 2021.
https://refactoring.guru/design-patterns/observer

Design Patterns: Elements of Reusable Object-Oriented Software door Erich Gamma, Richard Helm, Ralph Johnson en John Vlissides. ISBN 0-201-63361-2 Geraadpleegd op 14 mei 2021.

Sass Guidelines door Kitty Giraudel. Geraadpleegd op 3 mei 2021.
https://sass-guidelin.es/

Bulk en Gedonder-kwadrant kan uitkomst bieden door Basile Lemaire. Geraadpleegd op 3 juni 2021.
https://www.computable.nl/artikel/opinie/development/3969591/1509029/bulk-en-gedonder-kwadrant-kan-uitkomst-bieden.html

Tabellenlijst

Tabel 1: Gekozen libraries

```
composer create-project symfony/skeleton test
composer require roave/security-advisories --dev
composer require symfony/maker-bundle --dev
composer require symfony/profiler-pack --dev
composer require ergebnis/composer-normalize --dev
composer require orm-fixtures --dev
composer require fzaninotto/faker --dev
composer require test-pack --dev
composer require security
composer require form
composer require knplabs/knp-time-bundle
composer require symfony/orm-pack
composer require doctrine/doctrine-bundle
composer require doctrine/doctrine-migrations-bundle
composer require stof/doctrine-extensions-bundle
composer require easycorp/easyadmin-bundle
composer require alterphp/easyadmin-extension-bundle
composer require symfony/webpack-encore-bundle
```

Bijlagen

Hier worden alle opgeleverde tussenproducten als bijlage van het afstudeerverslag toegevoegd.

1. **Afstudeerplan HBO-ICT Robin Bastiaan**
2. **Plan van Aanpak**
3. **Voortgangsrapport**
4. **Requirements Interviews**
5. **Requirements Analyse**
6. **Release Checklist**
7. **Code Reviews**