DE HAAGSE HOGESCHOOL

Het ontwikkelen van een geautomatiseerd self-assessment systeem bij het CBS

Centraal Bureau voor de Statistiek

Auteur:	Milo Kastablank
Opleiding:	Software Engineer | Internet of Things
Afdeling:	Beleids- en Managementondersteuning
Datum:	10-05-2021 t/m 8-10-2021
Versie:	1.00 nu

Voorwoord

Voor u ligt het afstudeerverslag ‘Het ontwikkelen van een geautomatiseerd self-assessment systeem bij het CBS’. Het onderzoek beschreven in dit verslag is uitgevoerd binnen het CBS bij de afdeling Beleids- en Managementondersteuning.
Dit afstudeerverslag is geschreven in het kader van mijn afstuderen aan de opleiding HBO-ICT aan de Haagse Hogeschool. Vanaf Mei 2021 tot en met Oktober 2021 ben ik bezig geweest met het uitvoeren van het onderzoek en het schrijven van dit verslag.

Binnen dit document geef ik een omschrijving van de opdracht en de context waarbinnen deze opdracht wordt uitgevoerd. Ook beschrijf ik hoe ik tot een aanpak ben gekomen en hoe ik deze aanpak heb uitgevoerd. Tijdens het afstudeertraject kon ik mijzelf verder ontwikkelen in de Software Development sector. Hierdoor heb ik nieuwe programmeertalen geleerd, het bedenken en uitvoeren van conceptoplossingen, en het onderzoeken naar alternatieven.

Nederlands is niet mijn eerste taal hierdoor was het schrijven van deze scriptie een uitdaging. Echter heb ik mijn afstudeertraject beschouwd als een leerzaam proces dat een positief effect heeft gehad bij het ontwikkelen van mijn Nederlandse taalvaardigheden.

Bij deze wil ik graag mijn bedrijfsmentoren Dhr Wentink en Dhr van de Schoor bedanken voor de fijne begeleiding en hun ondersteuning tijdens dit traject. Ook wil ik mijn collega Dhr Declerck bedanken die mij had geholpen met het leren en werken met Python Django en andere platformen dat CBS gebruikt voor het ontwikkelen van applicaties.

Tevens wil ik mijn examinatoren Dhr de Neef en Dhr van de Beek bedanken voor hun tips rondom mijn eerdere versie tijdens het tussentijds assessment. Hierdoor kon ik de beste versie van mijn afstudeerverslag opleveren.

Ten slotte wil ik nog mijn vriendin Isabella bedanken voor het 24/7 steun tijdens mijn afstudeertraject en natuurlijk mij vertellen dat ik paar keer naar buiten moest voor frisse lucht.

Ik wens u veel leesplezier toe.

Milo Kastablank

Den Haag, 7 Oktober 2021

Organisatie

Naam: Centraal Bureau voor de Statistiek
Afdeling: Beleidstaf EBN
Postcode : 2490 HA
Adres: Henri Faasdreef 312, Den Haag

Mentoren:
Mentor #1 Naam: Dhr F. Wentink Mentor
#2 Naam: Dhr JH. van de Schoor Mentor
#3 Naam: Dhr SH. Declerck

Referaat

Naam	Omschrijving
ASP.NET	Open-source C# web applicatie framework ontworpen voor webontwikkeling om dynamische webpagina's te produceren.
Blaise	Software-platform binnen CBS ten behoeve van het bouwen en afnemen van computer ondersteunde vragenlijsten.
Django	Open-source Python web applicatie framework ontworpen voor webontwikkeling om dynamische webpagina's te produceren.
Library	Geheel van gegevens en programmeercode die wordt gebruikt om softwareprogramma's en -toepassingen te ontwikkelen.
MS SQL	Relationele database manager ontwikkelt door Microsoft. Een onderdeel van Transact-SQL
PostgreSQL	Open-source relationele database manager.
PyCharm	Ontwikkel omgeving applicatie voor het ontwikkelen van Python projecten

Python	Programmeertaal voornamelijk gebruikt voor het bouwen van web frameworks en scripts.
SQLite3	“Lightweight” relationele database manager.
Transact-SQL	Uitbreiding van SQL gemaakt door Microsoft.

Inhoud

Leeswijzer

Dit afstudeerverslag geeft een beschrijving van de ontwikkeling van een applicatie dat het self-assessment systeem binnen het CBS automatiseert. De ontwikkeling van deze applicatie wordt uitgevoerd als een afstudeeropdracht van het Centraal Bureau voor de Statistiek binnen de afdeling Beleids- en Managementondersteuning.

Gedurende mijn afstudeertraject heb ik gebruik gemaakt van de werkmethode Incrementele Kanban.

- **3. Onderzoeksopzet**: In dit onderdeel is het probleemdomein, probleemstelling en de doelstelling geanalyseerd en opgesteld. Daarnaast wordt er ook bepaald welke hulpmiddelen ik nodig heb om mijn doelstelling te bereiken. Ten slotte wordt ook mijn keuze voor Kanban toegelicht.
- **4. Requirements:** In dit onderdeel worden de eisen van zowel de stakeholders als opdrachtgevers opgesteld. Hierin worden ook mijn methodieken voor het vergaren en opstellen van de eisen besproken.
- **5. Scope van de opdracht**: Nadat de requirements, methodieken en benodigde onderdelen bekend waren, worden de werkzaamheden, randvoorwaarden en risico's bepaald door middel van een plan van aanpak. Daarnaast wordt ook mijn gebruik van Kanban en software architectuur toegelicht.
- **6. Deelproduct 1: Ontwerp Database en Code**: Aan de hand van de opgestelde requirements, aanpak en architectuur worden de database en de interne opbouw van de software ontworpen en uitgewerkt volgens de Model-View-Template architectuur. Bovendien worden ook voorbeelden geschetst om te laten zien hoe de gebruikersinterface van de software eruit komt te zien.
- **7. Deelproduct 2: Realiseer Database en Code**: Na het ontwerpen van de software wordt de software vervolgens gerealiseerd in code. De software is ontwikkeld met de Python programmeertaal en het Django framework.
- **8. Deelproduct 3: Testen**: Tijdens mijn realisatiefase werd mijn software periodiek getest. Tijdens het testen wordt er nagelopen of de software correct functioneert en visualiseert. Voor het testen werden verschillende testscripts ontwikkeld waarmee een ontwikkelaar de software automatisch kan testen.
- **9. Overdracht**: De laatste fase van dit project. Hierin word de applicatie overgedragen aan de stakeholders en opdrachtgevers zodat ze de software kunnen testen met behulp van twee handleidingen voor het gebruik van de software.

Het uiteindelijk resultaat van deze werkzaamheden is een webapplicatie dat CBS kan gebruiken om het invullen en beheren van een self-assessment te automatiseren. Aan het einde van dit verslag word de evaluatie van het opgeleverde product en uitvoering van het project beschreven.

1.Inleiding

In de periode van mei tot oktober 2021 heb ik mijn afstudeertraject uitgevoerd bij het overheidsorgaan Centraal Bureau voor de Statistiek (CBS). Het CBS verzamelt en publiceert statistische gegevens over de Nederlands samenleving. Het is daarom van belang dat de gegevens die worden gebruikt binnen de werkprocessen leiden naar accurate en kwalitatief hoogwaardige informatie.

Het probleem dat CBS heeft is dat proceseigenaren handmatig een self-assessment moeten invullen om te toetsen of de statistische processen overeenkomen met de kwaliteitsrichtlijnen van het CBS. Hierdoor gaat het invullen en beheren van een self-assessment moeizaam. Omdat het handmatige proces zo tijdrovend is, demotiveert het de gebruiker om het self-assessment nauwkeurig in te vullen, waardoor de kwaliteit van de statistische gegevens die uit statistische processen naar voren komt afneemt. Door het afnemen van de kwaliteit neemt ook de betrouwbaarheid van de statistische gegevens af. Een voorbeeld is dat het CBS statistieken heeft gepubliceerd waaruit blijkt dat de prijzen van bestaande koopwoningen in Nederland zijn gedaald, terwijl zij in werkelijkheid zijn gestegen. Wanneer het onderzoek uit de media blijkt dat dit niet klopt, schaadt dit de goede naam van het CBS.

Mijn afstudeeropdracht betrof het ontwerpen en realiseren van een web applicatie dat het invullen en beheren van een self-assessment automatiseert, om de bovenstaande problemen op te lossen. Deze applicatie moet draaien op een virtuele machine die gebruikt maakt van een server binnen het intranet van het CBS, waardoor alleen bevoegden hier gebruik van kunnen maken. Het CBS heeft ook de wens dat de applicatie het mogelijk maakt om de gegevens die in het self-assessment worden ingevuld te analyseren. Hiermee kunnen ze dan bestuderen wat er gedaan moet worden on de kwaliteit van de statistische processen te bevorderen. Een tweede wens is dat de applicatie ook mogelijk maakt om beheerders zelf self-assessments en overige vragenlijsten kan maken en beheren .

De opdracht betrof met het bestuderen van de huidige situatie en het bepalen van het scope. Vervolgens wordt er gekeken naar de benodigde hulpmiddelen om de web applicatie te realiseren naar de wensen van de opdrachtgevers en stakeholders. Daarna wordt de webaplicatie ontworpen, gerealiseerd en getest om te kijken of de kwaliteit naar behoren is. Ten slotte zal er een evaluatie plaatsvinden van de werkzaamheden en beroepstaken die zijn verricht tijdens het afstudeertraject.

2. Huidige Situatie

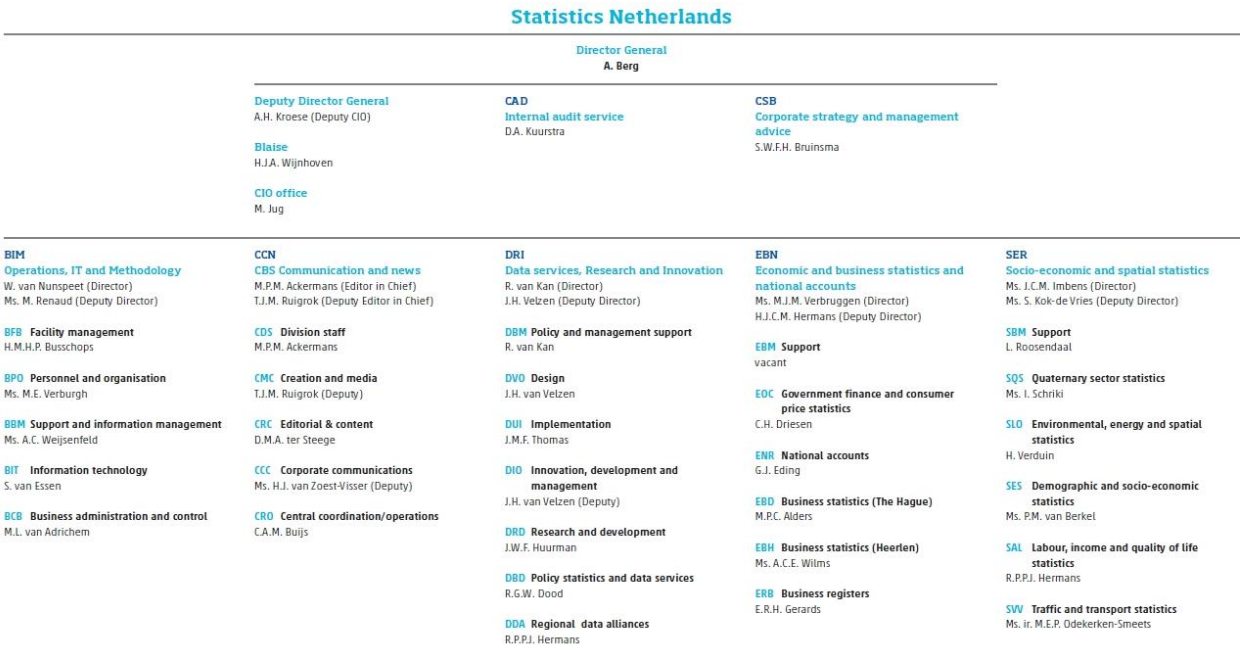
2.1 Bedrijf

Het Centraal Bureau voor de Statistiek (CBS) is een Nederlandse overheidsinstantie die informatie verzamelt over de Nederlandse samenleving. Op basis van deze informatie produceert het CBS statistische gegevens. De informatie die het CBS verzamelt heeft voornamelijk betrekking op economische en maatschappelijke onderwerpen in Nederland. Daarnaast publiceert het CBS haar resultaten en statistieken. De informatie is openbaar en daardoor voor iedereen tegelijkertijd toegankelijk, zodat nieuwsbedrijven niet eerder toegang krijgen tot haar publicaties.

Het CBS opereert binnen Nederland vanuit Den Haag en Heerlen. Daarnaast heeft het CBS ook een kantoor op Bonaire voor de verzameling van gegevens en de publicatie van de statistieken met betrekking tot Caribisch Nederland. De verdere verwerking van de data die verzameld worden, vindt plaats in Den Haag en Heerlen.

Binnen het CBS bevinden zich verschillende hoofddirecties die verschillende subafdelingen bevatten. Elke hoofddirectie heeft zijn eigen directeuren en managers, hierdoor is CBS opgebouwd in een lagenstructuur. Alle hoofddirecties worden gezamenlijk bestuurd door de directeur-generaal.

In figuur 1 is het organogram van CBS weergegeven [1]. Alle CBS hoofddirecties staan onder toezicht van de Directeur Generaal van het CBS, terwijl elke hoofddirectie wordt beheerd door een specifieke directeur. Dat houdt in dat een directeur toezicht houdt op projecten en processen binnen zijn/haar hoofddirecties. Onder elke hoofdirectie bevinden zich ook meerdere afdelingen die tevens worden beheerd door een specifieke directeur of hoofdmanager. Tijdens mijn afstudeertraject kwam ik voornamelijk in contact met collega’s vanuit EBN en SER, aangezien de twee hoofddirecties vaak samenwerken. Elke afdeling bevat teams die bestaan uit statistische onderzoekers, ontwikkelaars, proceseigenaren en kwaliteitsbeheerders.



Figuur 1: Organogram

2.2 Beleidstaf EBN

Mijn afstudeertraject wordt uitgevoerd binnen de beleidstaf van Economie, bedrijven en nationale rekeningen, oftewel Beleidstaf EBM.

De heer Fred Wentink is zowel mijn opdrachtgever als bedrijfsmentor tijdens dit traject. De heer Wentink is de hoofdmanager van zowel Beleidstaf EBM als Centrale Overheids Financiën binnen CBS. De Beleidstaf EBM is een afdeling die beleid en managementondersteuning biedt aan EBN. Daarnaast organiseert de beleidstaf het relatiebeheer bij EBN door het stroomlijnen en onderhouden van de contacten tussen de voor EBN relevante externe partijen en het CBS. Het externe relatiebeheer wordt afgestemd met andere divisies binnen EBN. De heer Jac van de Schoor is mijn tweede opdrachtgever. Hij is werkzaam als projectleider bij de afdeling Socio-economic en Ruimtelijke Statistieken (SER) en werkt samen met de Beleidstaf EBM. Omdat ik een afstudeerstudent ben, word ik beschouwd als een decentrale ontwikkelaar. Dit houdt in dat mijn werkzaamheden niet vallen onder een agile team maar meer als een éénmans team met begeleiding van een divers team.

2.3 Aanleiding

Tijdens mijn afstudeeropdracht focus ik mij op het proces van self-assessment. Dit proces omvat de handelingen en de instrumenten waar gebruik van wordt gemaakt door de proceseigenaren binnen EBN voor het toetsen van zijn/haar proces. Het doel van deze self-assessment is om vast te stellen in hoeverre het proces aan de relevante wet- en regelgeving (kwaliteitsrichtlijnen) voldoet.

De wet- en regelgeving die worden getoetst middels het self-assessments veranderen jaarlijks. De sjablonen die worden gebruikt voor het self-assessment moeten hierop worden aangepast. Momenteel gebeurt dit echter niet waardoor deze niet meer volledig overeenkomen met de kwaliteitseisen. Dit resulteert in self-assessments met een grote hoeveelheid verouderde of onjuiste informatie. Ook is het niet mogelijk om een samenvatting te maken van de overzichten van het ingevulde self-assessment ondanks dat hier wel behoefte aan is.

Er is geen database of een ander soortgelijk instrument aanwezig, waardoor zowel het beheren als het invullen van het self-assessment proces veel tijd kost.

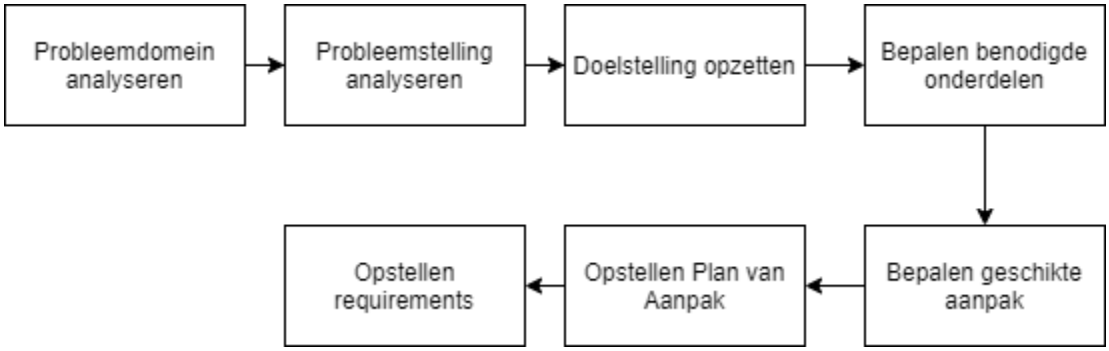
2.4 Self-assessment

Het self-assessment is een invullijst die wordt gebruikt door proceseigenaren en statistische onderzoekers. Een proceseigenaar is een persoon die toezicht houdt op statistische processen binnen het CBS. Een statistische onderzoeker is een persoon die zich voornamelijk bezighoudt met het onderzoeken van de statistische processen van het CBS. Om na te lopen of een proces voldoet aan de wet- en regelgeving en kwaliteitsrichtlijnen van het CBS, vult een proceseigenaar of een onderzoeker een self-assessment in.

Voordat ik begon met mijn afstudeerproject werd een self-assessment handmatig ingevuld middels een Microsoft Word bestand. Het CBS wil hier echter van af en wil het invulproces laten automatiseren met een op maat gemaakte applicatie voor een betere gebruikservaring en analyse doeleinden. Daarnaast behoort een self-assessment tot de categorie Voorschrift Informatiebeveiliging Rijksdienst (VIR) documenten. Dat zijn documenten waarin alle statistische processen op een standaard manier worden beschreven en getoetst.

3. Onderzoeksopzet

De volgende werkzaamheden heb ik uitgevoerd tijdens deze fase. Hieronder is visueel weergeven hoe ik te werk ben gegaan.



3.1 Probleemdomein

De afstudeeropdracht zal uitgevoerd worden binnen de afdeling EBN (Economie, bedrijven en nationale rekeningen). Dit is een afdeling binnen CBS die statistieken zoals de nationale rekeningen, de consumptieprijsindex en de internationale handel onderzoekt en publiceert. Dit wordt gedaan door het uitvoeren van statistische processen. Binnen deze afdeling bevindt zich een beleidsstaf die beleid- en managementondersteuning biedt aan EBN. Mijn werkzaamheden zullen plaatsvinden binnen deze beleidsstaf. Als afstudeerstudent zal ik mij bezig gaan houden met het self-assessment systeem dat momenteel gebruikt wordt door proceseigenaren binnen EBN.

3.2 Probleemstelling

Voor het toetsen van de processen maken de proceseigenaren van EBN gebruik van self-assessments. Het uitvoeren van een self-assessment is zo complex en tijdrovend dat het niet uitnodigt tot een zorgvuldige volbrenging ervan. Hierdoor ontstaan er kwaliteitsproblemen met die statistische data die voortkomt uit de getoetste processen waarop beslissingen en beleid worden gebaseerd. Wanneer er uit ander of aanvullend onderzoek blijkt dat de statistische data niet klopt, dan schaadt dat de goede naam van het CBS.

3.3 Doelstelling

Het CBS streeft naar een geautomatiseerd prototype systeem voor self-assessment als een webapplicatie. Binnen dit prototype zal een database worden ontwikkeld, hierdoor wordt het voor proceseigenaren en beheerders makkelijker om efficiënt gebruik te maken van het self-assessment lijsten. Daarom is het doel van mijn opdracht om een geautomatiseerd prototype te realiseren en te implementeren.

Mijn opgeleverd eindproduct moet vanuit de opdrachtgever voldoen aan een aantal eisen. Allereerst moet het prototype het mogelijk maken om de ingevulde vragen van het self-

assessment op te nemen in een database. Hierdoor kunnen de proceseigenaren eenvoudig een overzicht verkrijgen van de verschillende onderdelen binnen de ingevulde vragenlijsten, wat het analyseren vereenvoudigt.

Vervolgens moet het prototype het mogelijk maken voor de beheerders om het self-assessment sjabloon te kunnen aanpassen op basis van de kwaliteitseisen die met enige regelmaat veranderen. Dit houdt in dat het self-assessment een 1-op-1 relatie heeft met de huidige kwaliteitsrichtlijnen en dat het sjabloon van het self-assessment up-to-date blijft.

Het prototype kan verder ontwikkeld worden naar een volledig systeem na afloop van mijn afstudeertraject. Naast mijn prototype wil het CBS een globale beschrijving van de nieuwe input/ output processen van het prototype.

3.4 Hulpmiddelen om het doel te bereiken

Voordat ik kan beginnen met het realiseren van het prototype moet er eerst gekeken worden naar software tools en hulpmiddelen die moeten worden gebruikt om dit waar te maken. Normaliter kijk je pas na het vergaren van de software requirements de benodigde tools en hulpmiddelen. Maar omdat het CBS al in het begin duidelijkheid wilde over welke programmeertaal er gewerkt zal worden moest ik jammer genoeg al voorafgaand een keuze hiervoor maken.

Aangezien het prototype een webapplicatie wordt, was mijn oorspronkelijke keuze om gebruik te maken van de programmeertaal C# ASP.NET vanwege zijn flexibiliteit, prestatie met grote applicaties en onderhouds vriendelijkheid. Daarnaast is C# ASP.NET een compiled taal waarin een compiler meer tijd heeft om van te voren optimalisatie stappen toe te passen. Dat is niet mogelijk met Python aangezien het een script taal is. Het CBS gaf echter een voorkeur aan Python of Blaise aangezien tijdens mijn afstudeertraject nog niet bekend was welke afdeling het prototype zal gaan beheren na mijn afstudeerperiode. Daarnaast is Python een standaardtaal bij alle ICT afdelingen binnen CBS.

Blaise is een programmeertaal ontwikkeld in C# binnen CBS dat voornamelijk wordt gebruikt voor het maken van vragenlijsten. Aangezien het self-assessment een soort vragenlijst is, kan Blaise een solide keuze zijn voor dit prototype. Er is echter maar beperkte documentatie beschikbaar ten opzichte van Python. Daarnaast zijn enkele vastgestelde software eisen in hoofdstuk 4 niet toepasbaar met Blaise.

Uiteindelijk koos ik ervoor om mijn prototype te gaan ontwikkelen in Python met PyCharm Professional als ontwikkelprogramma. Niet alleen maakt deze keuze het makkelijker voor een CBS informatica afdeling om mijn gerealiseerde prototype te beheren, maar ook omdat Python een soepeler database integratie heeft dan Blaise. Daarnaast is het ook mogelijk om design patterns of ontwerpprincipes toe te passen. Dit is met Blaise haast niet mogelijk vanwege de werking van Blaise’s code architectuur.

Voor het gebruik van een database was het oorspronkelijke plan om gebruik te maken van Microsoft SQL Server vanwege zijn makkelijke integratie met C#, syntax en de mogelijkheid om gebruik te maken van functies zoals views en stored procedures. Tijdens een gesprek met een ontwikkelaar binnen CBS bleek PostgreSQL echter een vereiste database tijdens het ontwikkelen van software volgens de CBS ontwikkelrichtlijnen. Aangezien zowel PostgreSQL en Microsoft SQL Server allebei gebruik maken van de Transact-SQL syntax, zijn ze bijna identiek in gebruik. Hierom had ik geen bezwaar om gebruik te maken van PostgreSQL voor het bouwen van de database tijdens mijn afstudeertraject.

Tot slot heb ik voor versiebeheer en het deployment van de applicatie gebruik gemaakt van Cloud Foundry, een standaard binnen CBS. Op Cloud Foundry kan men zowel de gehele code als database deployen, dit maakt het mogelijk om een speciale Uniform Resource Locator (URL) te genereren zodat een reguliere gebruiker gebruik kan maken van de webapplicatie.

3.4.1 Afweging web applicatie frameworks

Voor het bouwen van een webapplicatie in Python moet er dan gebruik worden gemaakt van een web framework. Aangezien ik zowel een front-end als back-end moet ontwikkelen heb ik gekozen om een full-stack web framework voor Python te kiezen. Daarom werd een populaire micro web framework als Flask niet in overweging genomen. Er zijn tal van verschillende full-stack web frameworks maar op het einde was de keuze tussen Django en Turbogears. Uiteindelijk koos ik voor Django omdat er veel documentatie en tutorials over dat framework beschikbaar zijn[2][3]. Dit biedt betere ondersteuning tijdens het ontwikkelen ten opzichte van Turbogears. Daarnaast bevat Django meer library/package support dan Turbogears, wat het programmeren enigszins kan versoepelen.

3.5 Werkwijze

Bij dit project koos ik ervoor om gebruik te maken van Kanban, een werkmethode dat ook wordt gebruikt binnen de Beleidstaf. Hierbij wordt er eerst een backlog met taken gecreëerd waarin elke taak stap voor stap wordt uitgevoerd met behulp van een gevisualiseerde boord. Elke taak heeft zijn eigen tijdsduur gebaseerd op zijn tijdsintensiviteit[4].

De keuze voor een combinatie van Incrementele Kanban is gemaakt om de volgende redenen:

- Mocht er iets veranderen, dan kan dit makkelijk worden aangepast met een Kanban bord vanwege zijn maximale aanpassingsvermogen. Want er is geen voorgeschreven duur van de fasen en de prioriteiten worden voortdurend opnieuw beoordeeld op basis van de meest recente informatie.
- Een Kanban bord is een visuele representatie van de uit te voeren taken, hierom is het makkelijk te analyseren en toezicht op te houden.
- Aangezien ik tijdens de zomerperiode mijn opdracht uitvoerde, moest ik mijn contactmomenten met de stakeholders goed plannen. Met een Kanban bord kon ik expliciet afspraken maken met stakeholders en dus afspraken aanpassen.

- Omdat het prototype in de vorm van een web-app in Python Django wordt gemaakt moet het worden opgedeeld in meerdere onderdelen, bijvoorbeeld modellen, webpagina's, logica en databases. Daarom is het handig om elk onderdeel op te leveren per iteratie. Dat kan ik makkelijk visualiseren met een Kanban bord.
- De afdeling waar ik werkzaam ben maakt gebruik van Kanban, hierdoor vind ik het ook gepast om Kanban te gebruiken om beter bij mijn begeleiders aan te sluiten. Daarnaast werd het ook sterk aangeraden door mijn opdrachtgever om gebruik te maken van Kanban.

Deze keuze werd gemaakt na het overwegen van meerdere werkmethodes. De methodes die ik had overwogen waren: Waterval, Scrum en Rapid Application Development (RAD). In de volgende paragrafen zal er beschreven worden waarom er niet werd gekozen voor deze werkmethodes.

Waterval is een lineair werkmethode waarin de software in een stuk door wordt ontwikkeld. Echter is dit niet handig, aangezien er weinig contactmomenten zijn met mijn stakeholders en opdrachtgever na het vergaren van de requirements periode, in verband met de zomervakantie waardoor hun beschikbaarheid beperkt is. Stel dat de code structuur moet worden aangepast vanwege wijzigende requirements, dan zou er veel tijd worden besteed om de verandering aan te passen aangezien waterval zeer rigide is.

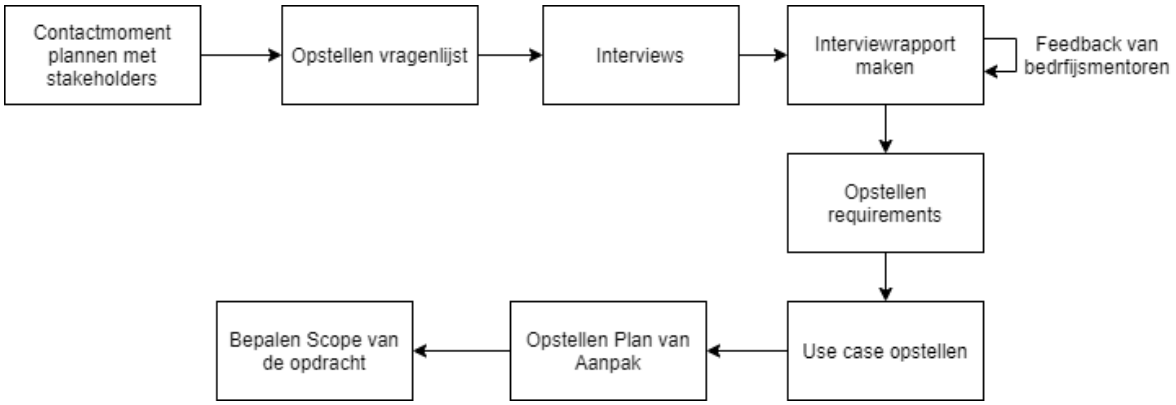
Omdat de Watervalmethode enigszins vergelijkbaar is met Kanban qua workflow bestond een gedeelte van mijn onderzoek uit het stellen van vragen binnen de Beleidstaf over waarom ze hadden gekozen voor Kanban. De Beleidstaf maakt gebruik van Kanban vanwege zijn agile bord dat werkflow overzichtelijk maakt vergeleken met Waterval.

Scrum is een agile werkmethode waarbij per sprint (korte werk cycli) taken worden geprioriteerd met behulp van planning poker met een scrum team. Echter is het voor mij niet mogelijk aangezien ik in een ontwikkelteam behoort en Scrum een team discipline is[5].

RAD vereist dat er eerst een prototype gemaakt wordt waarna er feedback van de gebruikers wordt verzameld. De feedback kan bruikbaar zijn voor het bepalen of de vragen goed waren opgesteld. Echter is het risico groot dat meerdere gebruikers niet reageren aangezien mijn traject door gehele duratie van de zomervakantie loopt. Dat risico kon ik mij niet veroorloven omdat mijn opdracht binnen de gestelde afstudeerperiode moest worden afgerond.

4. Requirements

Hieronder is een visueel overzicht gemaakt van de werkzaamheden die ik heb uitgevoerd bij het opstellen van de requirements van mijn afstudeeropdracht. Verderop is visueel weergeven hoe ik te werk ben gegaan.



Om mijn afstudeertraject goed te laten verlopen was het belangrijk om eerst software eisen te gaan opstellen. De requirements waren opgesteld op basis van de wensen en eisen van de stakeholders, die bestaan uit twee CBS proceseigenaren, vier statistische onderzoekers en twee kwaliteitsfunctionarissen. Voor elk type stakeholders had ik een specifieke vragenlijst gemaakt, het doel hiervan is om een beter beeld te krijgen van hoe het self-assessment wordt gebruikt door de verschillende rollen.

4.1 Requirements

Tijdens mijn interviews stelde ik vragen over hoe de eindgebruikers omgaan met het invullen van een self-assessment en wat er verbeterd kan worden. De vragen die gesteld zouden moeten worden waren voorafgaand aan het interview al opgesteld op half gestructureerde wijze. Dit houdt in dat alle onderwerpen die ik ga behandelen al op papier staan en dat ik tijdens mijn interviews doorvraag totdat een onderwerp voldoende is behandeld. Ik heb voor dit soort interview wijze gekozen omdat ik niet veel weet over self-assessments en de wensen en eisen voor het prototype. Tijdens de interviews kreeg ik een beter beeld over wat voor soort gebruikers gebruik maken van het self-assessment. Met de gegeven kennis kon ik een beter beeld schetsen over hoe ik de wensen en eisen van de stakeholders kon realiseren.

De gegevens van alle interviews werden vervolgens verwerkt in verschillende interview transcripts die gebruikt zullen worden als basis voor het opstellen van requirements. De interview transcripts bestaan voornamelijk uit antwoorden van de stakeholders op basis van mijn opgestelde vragen. Alle interviews werden gehouden via Zoom, aangezien het door COVID-19 vrijwel onmogelijk was om fysiek interviews uit te voeren. Voorafgaande aan de interviews heb ik toestemming gevraagd aan de stakeholders om het gesprek op te nemen, met als argument het optimaal verweken van de transcripts.

Aan de hand van de informatie die ik had vergaard tijdens de interviews begon ik met het opstellen van de requirements. Voor het indelen van requirements besloot ik om gebruik te maken van de MosCoW methode[6]. Ik heb hiervoor MosCoW gekozen omdat deze methode erg efficiënt is voor het classificeren van requirements op basis van de wensen en eisen van de stakeholders. Hierdoor kon ik makkelijker aangeven welke requirements een hogere prioriteit krijgen.

Ik had ook overwogen om gebruik te maken of MosCoW te combineren met het SMART-principe. Met SMART kan je bepalen of de haalbaarheid en meetbaarheid van individuele requirements realistisch zijn. Een voordeel is dat je een goede structuur kan bieden voor het bereiken van de doelen gesteld in de requirements. Echter uit eigen ervaring wordt het juist ongestructureerd en bijna onoverzichtelijk wanneer er veel requirements beschikbaar zijn. Hierdoor kan de focus op de requirements overal liggen, wat mijn acties kan beïnvloeden met betrekking tot het bereiken van mijn SMART-doelstellingen. Omdat ik meer dan 15 requirements had opgesteld besloot ik om toch geen gebruik te maken van SMART.

Zoals benoemd in de vorige alinea, werd er gebruik gemaakt van MosCoW. Deze methode is een afkorting van de volgende termen:

- **Must Have:** Minimale eisen waaraan het eindproduct moet worden voldaan.
- **Should Have:** Zeer gewenste eisen die geen invloed geven voor een functioneel eindporduct. Vaak worden ze alsnog geïmplementeerd op een relatief korte termijn.
- **Could Have:** Eisen die gewenst zijn maar niet zo’n hoge prioriteit hebben als Should Have’s. Het is wel fijn om deze eisen te implementeren als er nog genoeg tijd over is.
- **Won’t Have:** Eisen met de laagste prioriteit. Het zijn eisen die niet gewenst zijn maar wel mogen worden geleverd met het eindproduct

Het belangrijkste term van MosCoW is de Must Have. Ik had mijn Must Have’s bepaald op basis van de vaakst genoemde eisen vanuit de stakeholders en tijdens een gesprek met mijn opdrachtgevers, waarbij we de door mij opgestelde requirements doornamen.

Na het uitwerken van de requirements heb ik een document opgesteld met een overzicht van requirements. Met een overzicht bedoel ik de wijze waarop ik elk individueel requirement en de bijbehorende geformuleerde doelstellingen worden geprioriteerd. In dit document geef ik aan hoe mijn requirements tot stand zijn gekomen met de bijbehorende acceptatiecriteria. Binnen dit document heb ik ook use cases gemaakt dat de interactie tussen de gebruikers en de software beschrijft. De interactie is gebaseerd op de nieuwe functies die met de afstudeeropdracht komen vanuit de requirements.

Het document en verder uitleg van MosCow is te vinden in bijlage : Requirements.

4.2 Conclusie requirements

Voor het opstellen van de requirements heb ik een half-gestructureerd interviews uitgevoerd met acht stakeholders. Deze acht stakeholders maken voornamelijk gebruik maken van het self-assessment, zowel voor het invullen als beheren van dat document. Ook maken ze soms gebruik van de resultaten voor auditing en review doeleinden. Tijdens mijn interviews had ik tot doel om de onderstaande primaire vragen te kunnen beantwoorden:

- Waarom loopt het afnemen van het Self Assessment achter?
- Wat zijn de knelpunten van zowel het invullen als analyseren van de huidige Self Assessment documenten?
- Zien ze het Self Assessment als een nuttige instrument binnen het kwaliteitszorg?
- Hoe kan het Self Assessment meer toegevoegde waarde opleveren?
- Wat wordt er gedaan met de rapportage na het invullen van een Self Assessment?

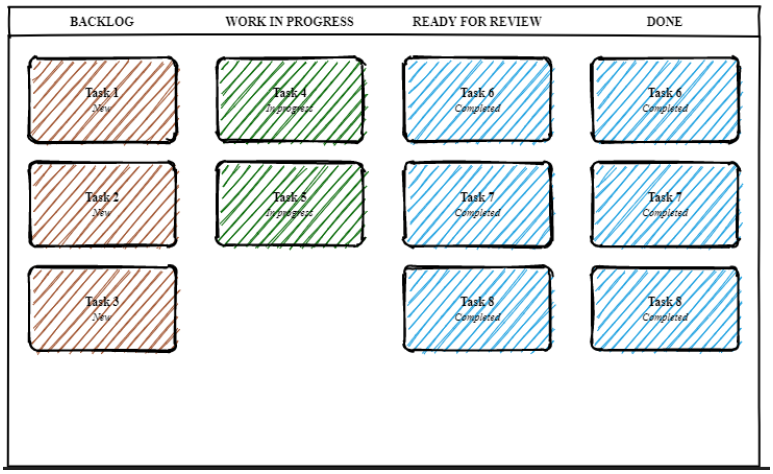
Na afloop had ik voldoende informatie om de requirements op te stellen. De bedrijfsmentoren hadden de opgestelde requirements vervolgens goedgekeurd. Ik heb de opgestelde requirements meegenomen in mijn Kanban bord zodat deze kunnen worden toegepast en bestudeerd en vervolgens een plan van aanpak kan opstellen.

5. Scope van de opdracht

Hieronder wordt globaal de scope van de opdracht beschreven.

5.1 Kanban board

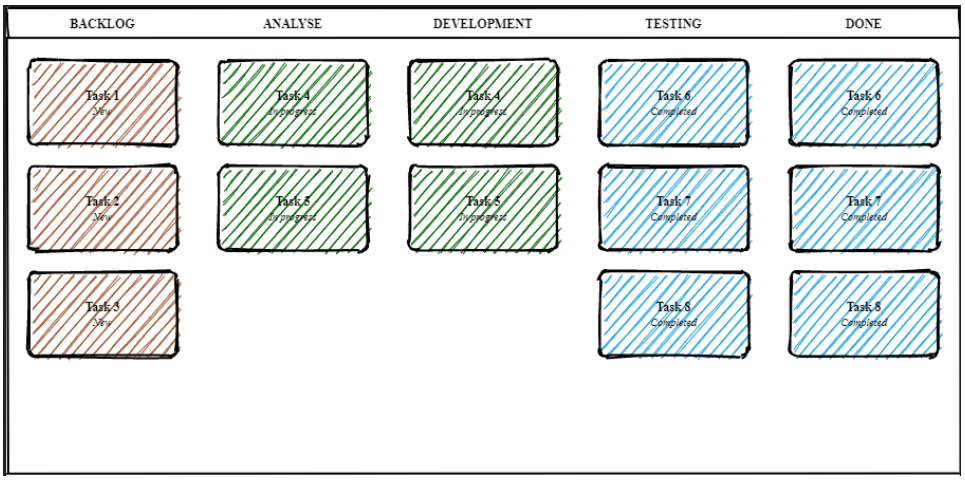
Voor Kanban wordt er gebruik gemaakt van twee verschillende Kanban borden: Een bord voor de onderzoeksfase en tenslotte een bord voor de ontwikkelfase. Het eerste bord komt er zo uit te zien:



Figuur 2: Onderzoeksfase Kanban board

Het bovenstaande bord wordt gebruikt tijdens mijn onderzoeksfase. Dat houdt in vergaren van informatie, interviews en opstellen van documentatie. Als eenmaal het uitvoeren van een taak klaar is, wordt het gereviewd door mijn opdrachtgevers of een senioren ontwikkelaar binnen CBS mits het over een technische taak gaat, bijvoorbeeld het opzetten van een ontwikkelplatform. Deze fase wordt beëindigd als eenmaal de volgende documenten zijn goedgekeurd door mijn bedrijfsmentoren: Requirements Rapport, Proces Flow Diagram en Analyse diagram voor het toekomstige prototype tool.

Als de onderzoeksfase is afgerond, schakel ik over naar een nieuw Kanban bord dat is ontworpen voor software ontwikkeling. Het ontwikkelbord komt er zo uit te zien:



Figuur 3: Ontwikkelfase Kanban board

Hierboven wordt er aangegeven dat dit Kanban bord andere swimlanes bevat ten opzichte van het vorige bord. Dit zijn allemaal swimlanes die betrekking hebben op het ontwikkelen van een product volgens het Agile principe, aangezien Kanban onderdeel is van Agile. Daarom vond ik het handig om met een apart bord te beginnen als mijn onderzoeksfase werd afgerond. Tijdens de ontwikkelfase worden alle componenten ontworpen en geanalyseerd (Analyse) die uiteindelijk worden gerealiseerd in code (Development). Vervolgens wordt elk gerealiseerde component getest in verband met unit testen en acceptatietesten (Testing). Als eenmaal het testen is afgerond, wordt een component beschouwd als compleet (Done) waardoor er verder kan worden gewerkt met het volgende component (Backlog) op een incrementeel cyclus.

De ontwikkelfase wordt beëindigd wanneer alle eisen volgens het requirements rapport voor het product succesvol worden geïmplementeerd en getest met een bijbehorend document dat dient als een handleiding.

5.2 Opstellen Plan van Aanpak

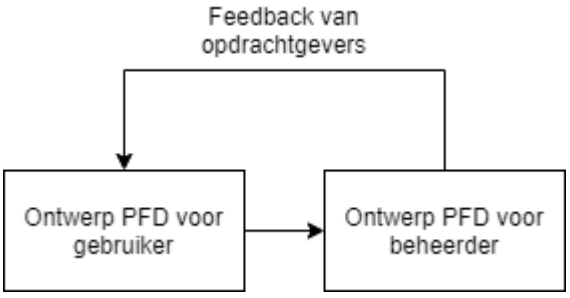
Nadat de requirements, onderzoeks- en werkmethodes en de benodigde onderdelen bekend waren, moest ik mijn werkzaamheden bepalen voor mijn opdracht. Hiervoor stelde ik een plan van aanpak op. Binnen het plan van aanpak werden alle technieken, tools, onderzoeksmethoden en platforms globaal toegelicht. Daarnaast werd de opdracht omschreven met de bijbehorende randvoorwaarden en risico's met een globale proces planning.

Het plan van aanpak deelde ik met mijn opdrachtgevers zodat zij het konden doornemen en hun feedback geven. Voor mijn eerste versie moest ik mijn planning aanpassen omdat de visualisatie niet duidelijk genoeg was. Na het uitwerken van een duidelijke planning waren mijn opdrachtgevers tevreden over mijn plan van aanpak.

Het document is te vinden in de bijlage: Plan van Aanpak.

5.3 Proces Flow Diagram en werking

De volgende werkzaamheden heb ik uitgevoerd voor deze taak:



Voordat er kan worden begonnen met de ontwikkelfase, is het belangrijk om de globale procesbeschrijving van het product in kaart te brengen. De beste manier om dit aan te pakken was om een visueel diagram te maken voor elke stap die moest worden doorlopen om gebruik te maken van het product. Hiervoor heb ik gebruik gemaakt van een Proces Flow Diagram (PFD).

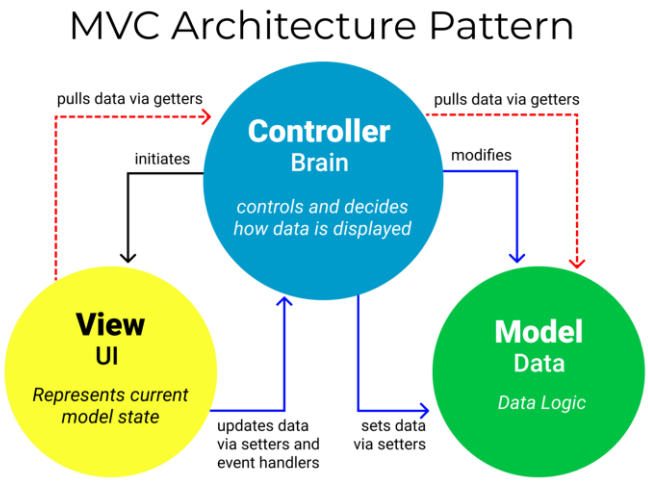
Redenering hierachter is dat met een PFD duidelijk aangegeven kan worden hoe een proces loopt met behulp van swimlanes en de bijbehorende objecten die als processen, inputs en outputs kunnen worden weergeven. Ik heb deze techniek geleerd tijdens mijn onderwijs en met behulp van voorbeeld PFD´s op het CBS Intranet [7] kon ik een PFD schetsen dat duidelijk aangeeft hoe het product werkt. In het PFD wordt er dan aangegeven hoe de algemene processtroom voor zowel een invuller als kwaliteitsbeheerder eruit zal zien. Eenmaal het PFD gemaakt, wordt het dan gedeeld met mijn bedrijfsmentoren voor goedkeuring en eventuele feedback.

Het PFD met zijn beschrijving is te vinden in de aparte bijlage Proces Flow Diagram.

5.4 Uitleg MVC en MVT

Aangezien ik ervoor had gekozen om met Python Django te werken voor mijn afstudeertraject moest ik ook een software architectuur kiezen. Normaliter wanneer je een software gaat ontwerpen kies je eerst het architectuur en dan pas de programmeertaal/software. Voor mij was het echter andersom omdat ik al tijdens het begin van mijn opdracht al een keuze moest maken over de programmeertaal. Hierdoor moest ik een software architectuur kiezen gebaseerd op de gekozen programmeertaal. Aangezien ik een webapplicatie moest ontwerpen is Model View Controller (MVC) de meest gebruikte architectuur vanwege zijn loskoppeling met zijn drie hoofdcomponenten[8]. Allereerst het Model, dat dient als brug tussen de applicatie en database. De View, welke functioneert als een interface waarbij een eindgebruiker verschillende acties kan ondernemen, bijvoorbeeld het weergeven van een formulier op een webpagina. Tenslotte heb je de Controller die zowel het Model als View beheert met updates en gebruikers acties. Het wordt voornamelijk gebruikt met de programmeertalen C# en PHP.

Een visuele werking van MVC ziet er uit als volgt:



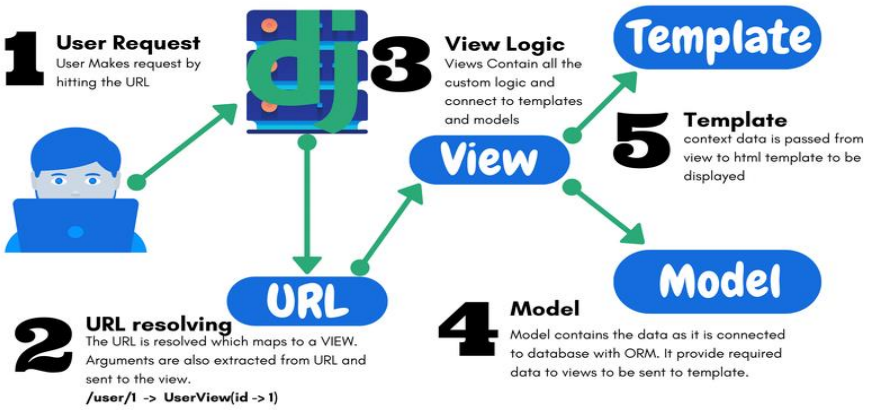
Figuur 3: Globale werking MVC

Kan je MVC optimaal toepassen op Django? Het antwoord is zowel ja als nee. Het MVC patroon pleit voor de ontkoppeling van de presentatie laag van de applicatie logica. Bijvoorbeeld, bij het ontwerpen van een online game website API, zou je de highscores tabel van een game kunnen presenteren als een HTML, XML, of CSV bestand. Echter, de onderliggende model klasse zou worden ontworpen onafhankelijk van hoe de gegevens uiteindelijk zouden worden gepresenteerd.

MVC is zeer rigide over wat modellen, views, en controllers doen. Echter heeft Django een veel praktischer kijk op web applicaties. Vanwege de aard van het HTTP-protocol, is elk verzoek om een webpagina onafhankelijk van elk ander verzoek. Django's framework is ontworpen als een pijplijn om elk verzoek te verwerken en een antwoord voor te bereiden.

Als een alternatief biedt Django het software architectuur Model View Template (MVT) aan[9]. MVT is een architectuur waarin elke oproep voor een webpagina onafhankelijk is van andere oproepen. In MVT wordt Model gebruikt voor het database interface. View wordt gebruikt voor het verwerken van oproepen en Template biedt een presentatie laag. Deze is vergelijkbaar met een View van MVC. Omdat MVT specifiek is ontworpen voor Django, zou ik dan veel tijd besparen dan wanneer ik MVC zou proberen toe te passen op Django.

Een visuele werking van MVT ziet er uit als volgt:



Figuur 4: Globale werking MVT

5.5 TO-DO Punten

Vervolgens heb ik voor mijzelf een overzicht gemaakt waarin de verschillende onderdelen staan die ontwikkeld moeten worden voor de applicatie. Dit overzicht kon ik uiteindelijk gebruiken tijdens het ontwerpen en realiseren van de applicatie.

Het overzicht ziet er als volgt uit:

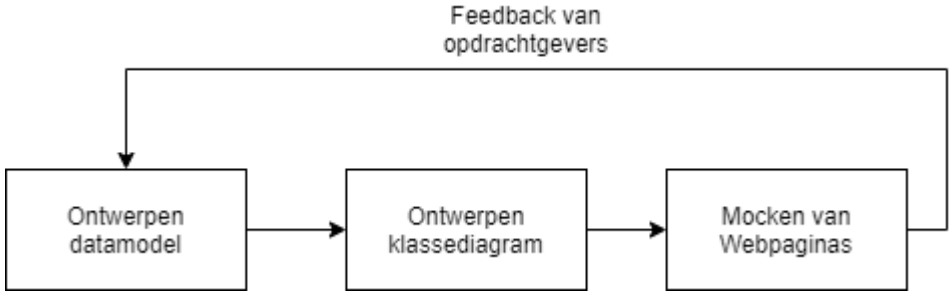
TODO: MODEL	Req	Omschrijving		TODO: V	Req	Omschrijving		TODO: TEMPLAT	Req	Omschrijving
Model concept	/	Modellen ontwerpen		Views concept ontwerpen	R14	Het ontwerpen over het algemene werking van alle Views		Index template	R6	Weergeven van het index view
Python omgeving installeren	R8	Het computer taal		Index View	R12	Weergeven van een lijst van beschikbare vragenlijsten		vragenlijst template	R6	weergeven van het Detail View
PostgreSQL installeren	R9	Het database		Detail View ontwikkelen: Vragen	R5,R6	Opbouwen van vragenlijst door vragen vanuit het database te weergeven op een pagina		rapportage invuller template	R6	weergeven van het Rapportage Invuller View
Vraag Model ontwikkelen	R2	Het basis model voor het aanmaken van vragen		Detail View: Categorie	/	Vragen te weergeven en sorteren per categorie		rapportage admin template	R6	Weergeven van het Rapportage Admin View
Antwoord Model ontwerpenontwikkelen	/	Het basis model voor het aanmaken van ingevulde antwoorden		Detail View: Antwoorden	R10,R11	Antwoord opties weergeven per vraag en voor zorgen dat het gekozen antwoord wordt opgeslagen		templates urls mappen	/	Link opleggen tussen het tempalte URL's met de Views
VirDocument Model ontwikkelen	R2	Het basis model voor vir documenten		Detail View: Opslaan	R15	Een ingevulde vragenlijst moet worden verzonden met een POST Request en worden opgeslagen in een Collectie model met een unieke code		Bootstrap toepassen	R6	Toepassen van CSS Bootstrap voor styling
Collectie Model ontwikkelen	/	Het basis model voor al ingevulde vir documenten		Rapportage Invuller View	R18	Maakt het mogelijk om een overzicht van ingevulde antwoorden van het huidige sessie te bekijken		Inlog scherm	R7	Pagina weergeven waarin een gebruiker de optie heb om in te loggen
Categorie model ontwikkelen	R2	Het basis model voor het aanmaken van vragen categorieen		Rapportage	R2,R18	Maakt het mogelijk om informatie op te halen en filteren van ingevulde vragenlijsten				
Model migreren naar Database	/	Voor het aanmaken van alle modellen in het database		Detail View: Toelichting knop	R1	Met een knop kan je een hulp text van een vraag weergeven tijdens het invullen van een vragenlijst				
				Detail View: Vragen skippen	R10,R11	Eenmaal een vraag binnen een categorie met Nee is beantwoord, krijg het gebruiker een melding of datgene persoon alle vragen binnen het categorie met Nee automatisch mag beantwoorden				

Figuur 5: To Do lijst

Hiervoor had ik mijn overzicht in 3 tabellen gesorteerd: Model, View en Template. In elke tabel wordt weergegeven aan welke taken moet worden voldaan om de verschillende componenten af te ronden. Ook geeft elke tabel voor bepaalde taken aan bij welke requirements(s) ze horen. Daarnaast bevat elke taak ook zijn eigen korte beschrijving. Deze taken nam ik vervolgens op in het backlog van een Kanban bord. Hiermee kon ik mijn taken beter prioriteren en vervolgens delen met mijn opdrachtgevers.

6. Deelproduct 1: Ontwerp Database en Code

In deze fase zijn de volgende werkzaamheden uitgevoerd. Hieronder is visueel weergeven hoe ik te werk ben gegaan.



6.1 Ontwerpen Data model

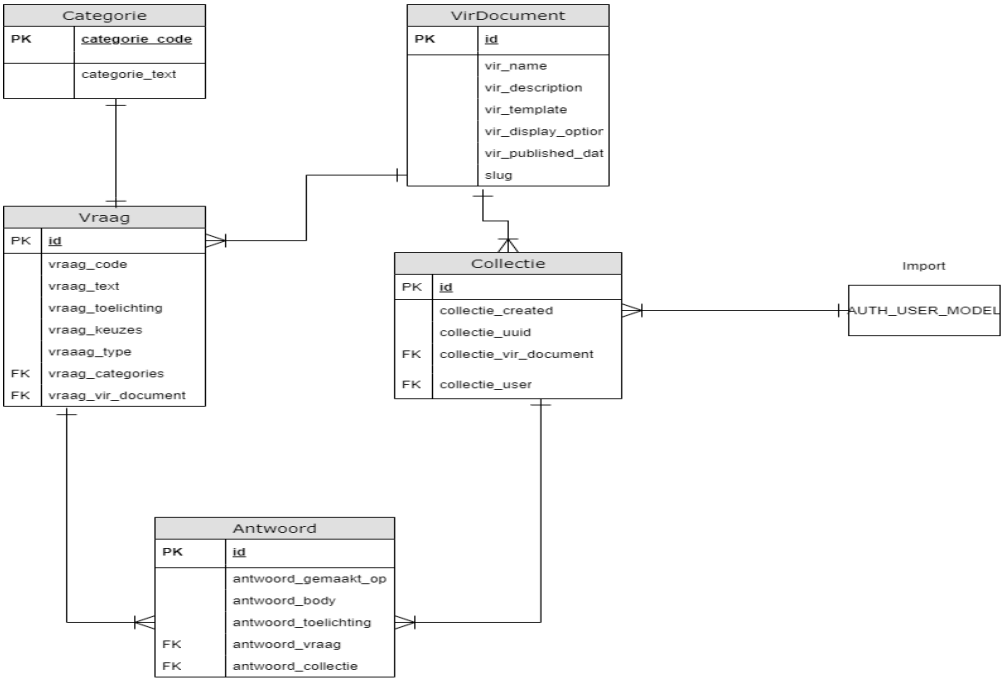
Aangezien mijn stakeholders behoefte hadden aan een database was het van groot belang dat ik zorgvuldig een database model ontwierp dat een verbinding kan maken met de te ontwikkelen software. Ondanks dat het mogelijk is met Python Django om direct een datamodel gebaseerd op de Models te genereren, is het toch belangrijk om een datamodel te ontwerpen voor de modellen omdat het makkelijk is om een datamodel aan te passen indien het noodzakelijk is.

Na nader onderzoek met benodigde informatie vanuit zowel het internet als CBS's intranet besloot ik om gebruik te maken van een Entity-Relationship Diagram (ERD) voor het maken van een datamodel. Redenering hierachter is dat je met een ERD makkelijker alle entiteiten, in dit geval modellen kan weergeven met hun relaties. Hierdoor is het makkelijker om de logische structuur van een database uit te leggen.

Ten eerste moet er een model worden ontworpen voor het beheren van vragen, zoals vraag code, id, antwoordopties en toelichting. Ten tweede moet een Vraagmodel ook antwoorden bevatten, daarom wordt er ook een antwoordmodel ontworpen. Omdat een self-assessment behoort tot een vir-document (zie hoofdstuk 2.4) en zelf vragen bevat, is het dan ook verstandig om een model genaamd VirDocument te maken dat één-op-één meer relatie bevat met het vraag model. Elk vraag moet uniek zijn voor elk individueel VirDocument volgens mijn opdrachtgever.

Ten slotte moet een ingevulde VirDocument op een database worden opgeslagen, anders kan een ingevuld VirDocument niet worden geanalyseerd door de kwaliteitsbeheerders. Voor deze taak wordt er dan een Collectie model ontworpen. Telkens wanneer een ingevulde VirDocument wordt ingediend, wordt het dan opgeslagen in een enkel Collectie model met de datum van aanmaak en de gebruiker die het heeft ingevuld.

Hieruit volgt het volgende ERD:



Figuur 6: ERD Diagram

6.2 Ontwerpen klassendiagram

Zoals eerder beschreven heb ik gebruik gemaakt van het Model-View-Template voor het ontwerpen van de software. In dit project wordt het Model gehanteerd als een database laag, de View als het business logic terwijl het Template wordt gebruikt als de presentatielaag. De reden waarom er gebruikt wordt gemaakt van lagen is omdat MVT is gebaseerd op het ontwerp principe Domain Driven Design (DDD) [10]. Hierin wordt een applicatie gebouwd in lagen met de benodigde logica omheen om de gewenste applicatie te ontwikkelen. De gehele applicatie neemt inspiratie van een open source project genaamd “django-survey” op github[11]. Hierdoor wordt mijn ontwerpstructuur nauw overgenomen van dat project.

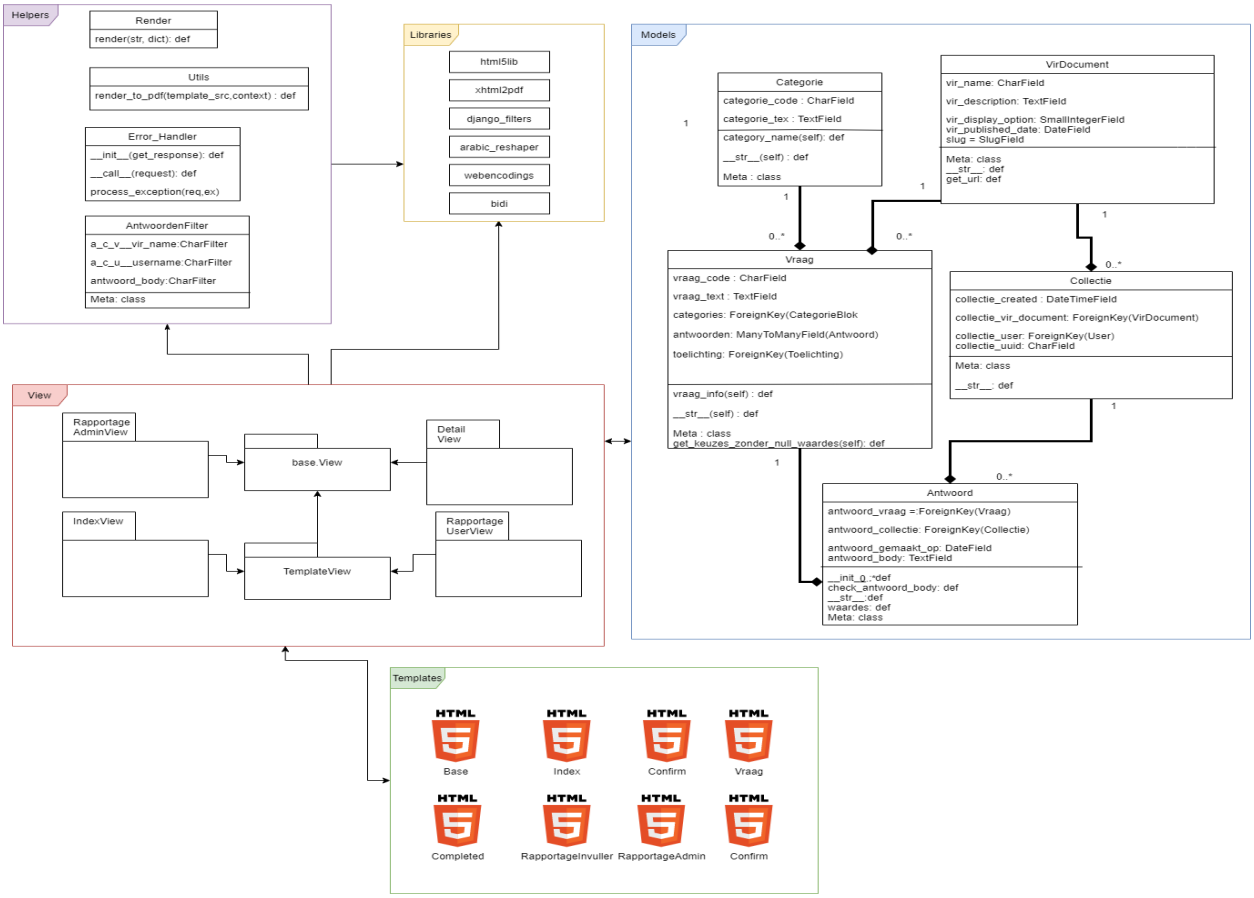
Het Model, ofwel de database laag bevat verschillende klassen dat samen werkt met de View voor het transporteren van data en het renderen van het Template. Elke klasse model heeft een interactie met de database. Dit gaat voornamelijk voor het beheren van vragen, vir-documenten, zijn antwoorden en categorieën. Wanneer een vir-document/self-assessment wordt ingevuld dan wordt al de ingevulde informatie opgeslagen in een apart model genaamd “Collectie”. Hierdoor wordt het dan makkelijker om informatie over ingevulde assessments te beheren vanuit het Collectie model. Alle models kunnen worden beheerd via Django’s eigen interface, Django Admin Interface. Verdere verdieping is terug te lezen in hoofdstuk 7.2.

De View, ofwel de business logic, bevat klassen die een interactie maken met het Model en Template. Zonder View klassen kan er geen verbinding ontstaan tussen het Model en Template, waardoor de applicatie niet zal werken. Het gaat voornamelijk om het weergeven van vragen, antwoorden vanuit het Model naar een Template en uiteindelijk het weergeven van een rapportage gebaseerd op het ingevulde vragen.

Om bepaalde Views te ondersteunen, had ik bedacht om bepaalde hulp klassen te ontwerpen. Uiteindelijk zouden ze functioneren als soort lokale libraries. Naast de helper klassen maak ik ook gebruik van verschillende externe python libraries om bepaalde requirements te realiseren, onder andere het generen van een PDF bestand via een View.

De Template, ofwel de presentatie laag bevat HyperText Markup Language (HTML) bestanden dat uiteindelijk de datamodellen rendeert naar een user interface met behulp van de View. Deze laag zorgt voor het weergeven van webpagina's met vragen en zijn antwoordopties, rapportage en speciale webpagina's waarin kwaliteitsbeheerders de database kan beheren.

Het ontwerp ziet er als volgt uit, deze diagram is tevens te vinden in de aparte bijlage Klassendiagram



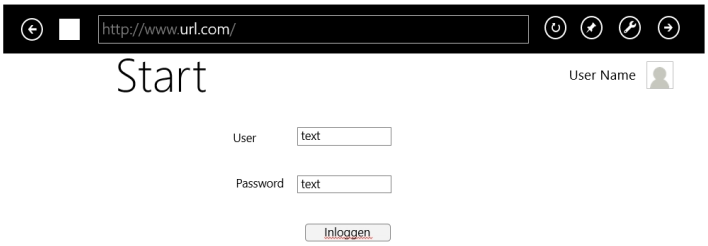
Figuur 6: Klassendiagram

6.3 Ontwerpen mocking

Een belangrijk onderdeel van de applicatie is de user interface (UI). Door middel van een UI kan een gebruiker een vragenlijst invullen en beheren vanuit een webpagina.

Omdat de webapplicatie ervoor zorgt dat het mogelijk is om een self-assessment weer te geven, is het belangrijk dat de visuele weergave van de layout zo veel mogelijk overeen komt met het sjabloon van een self-assessment. Aangezien de layout van een self-assessment en baseline sjabloon veel overeenkomsten hebben vond ik het praktisch om het UI's van het webpagina's te ontwerpen waarin de layouts overeenkomen met beide sjablonen.

Het voorbeeld van de complete UI's van elk webpagina ziet er als volgt uit:

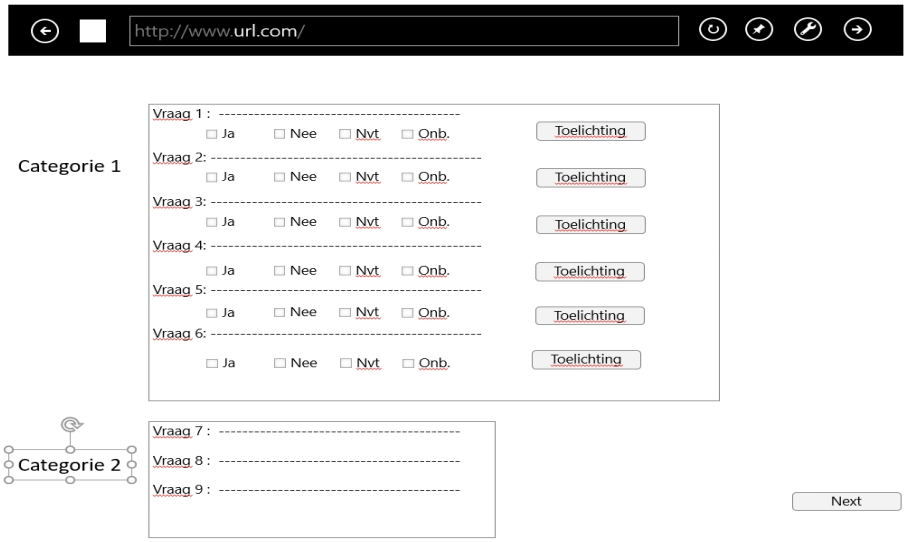


Wanneer een gebruiker voor het eerst de applicatie opstart, dan komt de gebruiker terecht in een inlog scherm. Op basis van zijn of haar functie wordt er dan bepaald of de gebruiker een invuller of kwaliteitsbeheerder is.

Figuur 7: Mock Login scherm

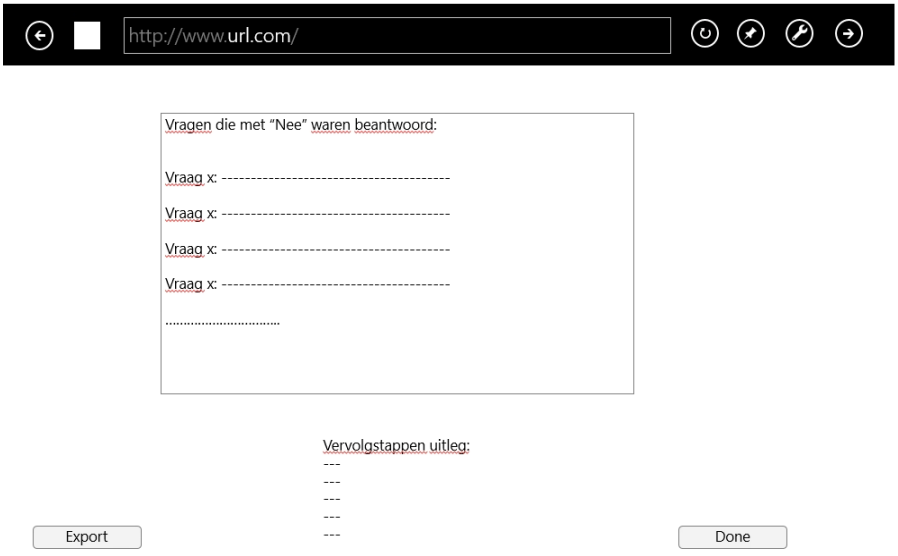
Na het inloggen kan zowel een invuller als kwaliteitsbeheerder kiezen welke vragenlijst hij wil gaan invullen. De lijst van vragenlijsten wordt dynamisch weergeven in een lijst in het backend van de applicatie. Dit houdt in dat het automatisch meerdere vir-documenten weergeeft indien ze aanwezig zijn in de database.

Figuur 8: Mock Index scherm



Figuur 9: Mock Vragenlijst scherm

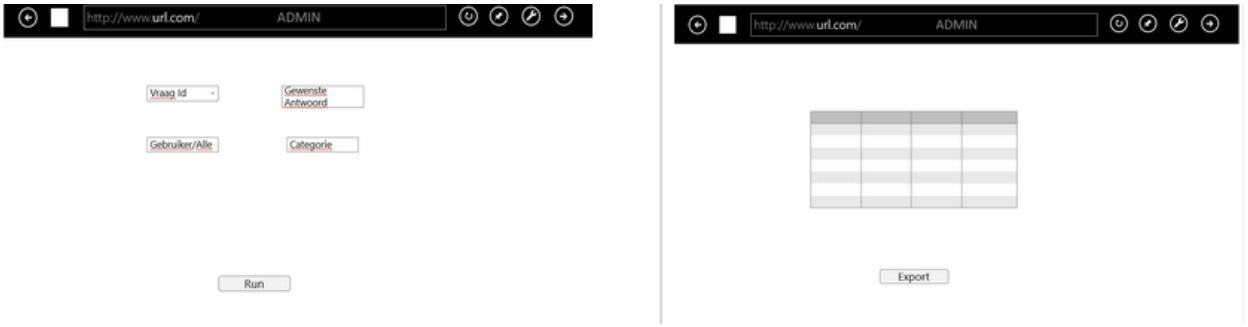
Op deze pagina kan een gebruiker een vragenlijst invullen die gesorteerd is per categorie. Elke vraag binnen een categorie bevat een vraagtekst en meerkeuze antwoorden opties. Indien een bepaalde vraag een toelichting bevat als hulptekst kan een gebruiker op de toelichting knop klikken voor een klein pop-up scherm dat een helptekst weergeeft.



Figuur 10: Mock Rapportage User scherm

Wanneer een gebruiker klaar is met het invullen van een self-assessment, komt hij terecht op een aparte webpagina dat dient als een evaluatie rapportage. Deze rapportage bevat een overzicht van alle vragen die met "Nee" worden beantwoord (aangezien er zoveel mogelijk

vragen niet met “Nee” mogen worden beantwoord vanwege de kwaliteitsrichtlijnen) en een lijst met vervolgstappen om verbeteracties uit te voeren. Daarnaast heeft een gebruiker ook de optie om deze resultaten te exporteren naar een PDF bestand.

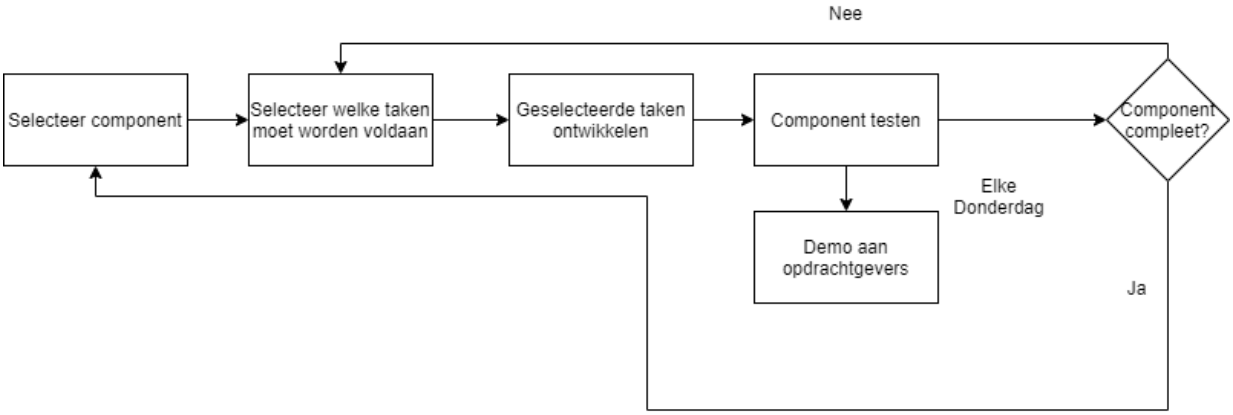


Figuur 11 en 12: Mock Rapportage Admin schermen

Indien de gebruiker een kwaliteitsbeheerder is, dan heeft men toegang tot een speciale webpagina waarin de resultaten van alle ingevulde self-assessments die zijn opgeslagen in een database kunnen worden gefilterd. Dit zal voornamelijk worden gebruikt voor het zoeken naar specifieke informatie of wat de meest voorkomende antwoorden zijn bij specifieke vragen of categorieën. Links ziet u de pagina met filteropties en rechts ziet u het resultaat dat wordt weergeven op een data tabel.

7. Deelproduct 2: Realiseer Database en Code

Deze fase dient voor het daadwerkelijk ontwikkelen van de verschillende componenten van de applicatie. Hieronder is visueel weergeven hoe ik te werk ben gegaan.



In totaal zijn er vier componenten die moesten worden uitgewerkt voor de Django applicatie: Models, Views, Templates en een Django-Admin interface voor kwaliteitsbeheerders en overige bevoegden. Met behulp van een Kanban bord koos ik per component een taak om uit te werken. Elke taak is een onderdeel van een software requirement volgens het requirements rapport. Nadat de taken van een component zijn uitgewerkt kies ik een volgende component. Elke donderdag geef ik een demo van mijn applicatie aan mijn opdrachtgevers of bespreken we mijn voortgang. We maken hierbij gebruik van Zoom.

Ondanks dat de ontwerp- en analyse werkzaamheden al werden uitgevoerd, werd de ontwikkelfase incrementeel uitgevoerd. Telkens wanneer een to-do taak compleet is, wordt het component stuk op een iteratieve manier ontwikkelt. Als er iets fout gaat tijdens het ontwikkelen van een component, dan kan ik het terugkoppelen naar een vorige iteratie om de fout ongedaan te maken. Hierdoor hoef ik niet helemaal opnieuw beginnen met het ontwikkelen van de software.

In hoofdstuk 3: onderzoeksopzet heb ik aangeven dat ik gebruik zal gaan maken van Python met het web-framework Django voor het realiseren van dit product. Op het intranet van het CBS zijn verschillende python standaard libraries beschikbaar die ik kon gebruiken, aangezien ik geen externe internettoegang had op mijn virtuele ontwikkeling. Echter zijn een paar websites door CBS op een “whitelist” geplaats mocht ik een specifieke python library nodig heb. Voordat ik begon met het realiseren van dit product volgde ik een Udemy cursus over Python en bestudeerde ik Python/Django boeken die ik had aangeschaft.

7.1 Models

Allereerst heb ik de Models component uitgewerkt. Het leek mij een goed idee om met deze component te starten omdat er voor de meeste modellen niet veel logica moet worden geprogrammeerd maar vooral declaraties. Daarnaast moeten de relaties tussen de modellen

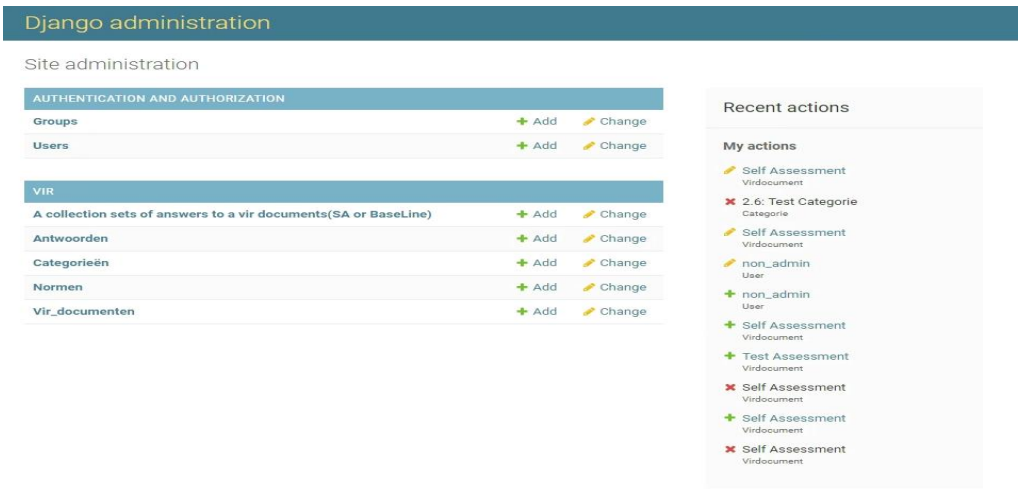
ook correct zijn. Zonder models kan ik niet optimaal de Views programmeren aangezien een View grotendeels afhankelijk is van een of meerdere Models.

Het uitwerken van de models verliep enigszins soepel. Elk model bevat hulp functies om bijvoorbeeld attribuutnamen op het Django-Admin-Interface correct te weergeven. Voor het Antwoord model had ik een functie geschreven om te kijken of een bepaalde Vraag objecten de optie hebben om een vraag te beantwoorden. Daarnaast bevat het Vraag model ook een functie welke het niet mogelijk maakt om een vraag te maken/publiceren dat geen antwoord opties bevat (denk aan Ja, Nee, Nvt etc).

Na het realiseren van alle Modellen zijn deze gemigreerd naar een SQLite3 database, dit maakte het mogelijk om de modellen te beheren vanuit een database. Ik had gekozen om eerst te gaan ontwikkelen op een SQLite3 database omdat het standaard meekomt met Python Django, terwijl er een heel proces is om PostgreSQL te installeren op Django. Daarnaast is SQLite3 zo ontworpen dat voor het migreren naar PostgreSQL bijna geen extra codeer werk wordt vereist. Tijdens de overdracht werd de SQLite3 database vervolgens gemigreerd naar PostgreSQL. Hiermee kan dan alle modellen worden beheerd door een administrator met behulp van de Django Admin Interface, dat in het volgende paragraaf globaal wordt toegelicht.

7.2 Django Admin Interface

Een van de meest krachtigste onderdelen van Python Django is de automatische beheerinterface genaamd Django Admin Interface (DAI) welke benaderbaar is vanuit een webbrowser. Dit onderdeel leest metadata van de Models door middel van een model-centrische interface voor vertrouwde beheerders waardoor ze de metadata kunnen beheren. Hierin kan een beheerder vragen, vir-documenten, gebruikers en categorieën maken en beheren. Daarnaast worden alle Collectie objecten en ingevulde antwoorden daarin opgeslagen vanwege zijn directe verbinding met een relationele database.

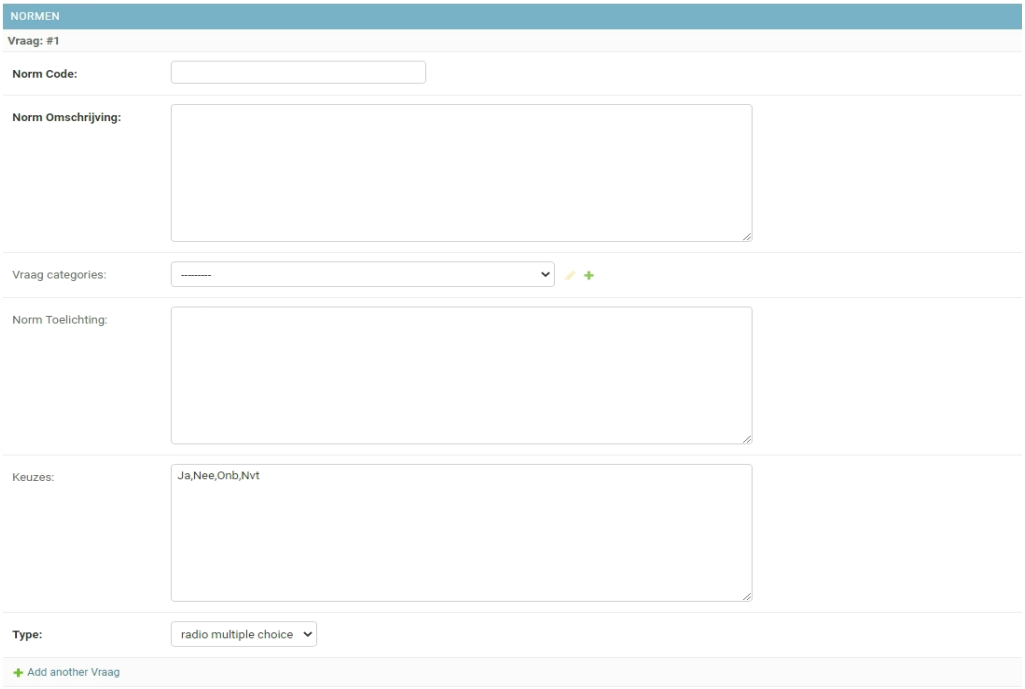


Figuur 13: De thuispagina van een Django Admin Interface

Op de pagina welke wordt weergeven in de onderstaande afbeelding kan een beheerder vanuit het overzicht bestaande vragenlijsten aanpassen of verwijderen. Als er een nieuwe vragenlijst sjabloon moet worden gemaakt dan heeft die beheerder ook de mogelijkheid om een nieuwe vragenlijst sjabloon te maken door middel op het “ADD VIRDOCUMENT +” knop rechtsboven te klikken.

Figuur 14: Het VirDocument pagina op de Django Admin Interface

Als er een nieuwe vragenlijst wordt gemaakt, moet de beheerder eerst een naam en omschrijving geven van de nieuwe vragenlijst. Het unieke ID wordt al automatisch gegenereerd maar er is ook een optie om het ID aan te passen. Vervolgens kan een vraag handmatig worden gemaakt door de benodigde attributen in te vullen dat te zien is op het onderstaande figuur.



Figuur 15: Het vragen pagina op de Django Admin Interface

Een nadeel is echter dat het niet mogelijk is om meerdere vragen in één keer in te vullen. Dat komt omdat Django Interface geen mogelijkheid biedt voor dat soort functionaliteit. Daarom kan er maar één vraag tegelijkertijd worden aangemaakt in het vragenlijst creatie scherm. Als het gaat om een grote vragenlijst (denk aan meer dan 200 vragen) dan kan het snel onoverzichtelijk worden. Als oplossing voor het laatstgenoemde probleem heb ik een optie gemaakt waardoor een beheerder eerst vragen apart kan maken en vervolgens te laten doorverwijzen naar een vragenlijst/vir-document.

Wanneer een invuller klaar is met het invullen van een vragenlijst, dan wordt al de ingevulde informatie opgeslagen in een Collectie model. Met behulp van Django Admin Interface is het ook mogelijk om alle opgeslagen Collectie modellen daarin terug te zien met alle belangrijke informaties zoals invuller, datum en alle ingevulde antwoorden.

Figuur 16: Een opgeslagen ingevulde vragenlijst op de Django Admin Interface

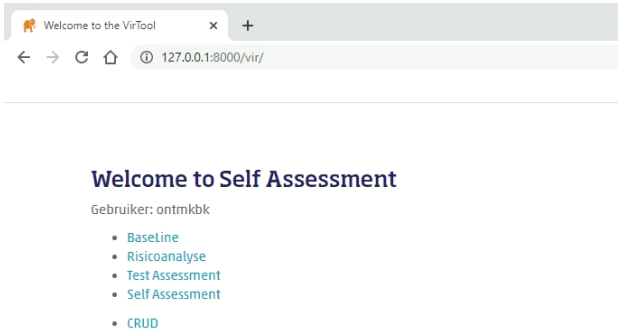
Een meer gedetailleerd uitleg en het gebruik van de Django Admin Interface is te vinden in de bijlage: Handleiding Admin.

7.3 Views en Templates

Na de Modellen ging ik aan de slag met de Views en Templates. Dit component bevatte vier klassen. Rapportage View voor Gebruikers, Rapportage View voor Kwaliteitsbeheerders, Index en Details View. Ik zal uitleggen waarom de bovenstaande klassen nodig zijn en een omschrijving geven van de geïmplementeerde functionaliteit.

7.3.1 Index

De Index View zorgt ervoor dat de thuispagina van de webapplicatie een lijst van alle beschikbare VirDocumenten weergeeft door ze allemaal op te halen vanuit de database. Dit is gesorteerd op basis van hun publicatiedatum. Als de gebruiker een administrator of een kwaliteitsbeheerder is zal er via de Indexpagina een extra optie worden gegeven naar de Rapportage Admin pagina.



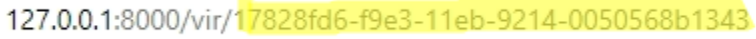
Figuur 17: Index scherm

7.3.2 Details

Nadat er op een VirDocument is geklikt wordt de Details View in werking gebracht. De Details View is zogenaamd “het hart” van de hele applicatie omdat deze de belangrijkste logica’s bevat. Dit omvat het verkrijgen, verwerken en verzenden van vragen en antwoorden binnen een VirDocument naar een Collectie. Dit wil zeggen dat deze view de logica voor het weergeven en het invullen van een vragenlijst regelt.

Ten eerste wordt het geselecteerde VirDocument vanuit de Index lijst opgehaald uit de database. Elk opgehaald VirDocument bevat vragen gesorteerd op hun identificatienummers en categorieën. Ik koos ervoor om per pagina een categorie met vragen weer te geven voor prestatie doeleinden. Dit omdat een self-assessment meer dan 200 vragen bevat en de kans groot is dat de hele webapplicatie vast zal lopen als alles op een pagina wordt weergeven.

Echter het verzenden van een ingevuld VirDocument is anders dan het ophalen van dat document. Wanneer een invuller klaar is met het invullen van een vragenlijst, wordt er een unieke identificatie nummer gegenereerd (UUID) met de toegewezen gebruikersnaam. Ik heb voor UUID’s gekozen in plaats van een normale ID omdat je met Python Django gemakkelijk UUID’s kan generen.



Figuur 18: De gele gemarkeerde string is een UUID

Tijdens het versturen worden alle ingevulde vraag keuzes omgezet naar Antwoord objecten die vervolgens samen worden verstuurd met het ingevulde VirDocument. Ten slotte worden zowel het ingevulde VirDocument als de opgeslagen Antwoord objecten verstuurd en opgeslagen in een aparte Collectie model object. Binnen deze View wordt er ook gekeken of alle Collectie objecten wel of niet uniek zijn op basis van hun gegenereerde UUID’s.

```python
@staticmethod
def post(request, slug):
    virdocument = VirDocument.objects.get(slug=slug)
    rap_uuid = request.POST["rap_uuid"]

    collectie = Collectie.objects.get(collectie_uuid=rap_uuid)  # Een opgeslagen vragenlijst
    flag = request.POST["inputFlag"]

    vragen = {vraag.vraag_code: vraag for vraag in virdocument.vragen.all()}
    post_keys = list(request.POST.keys())

    step = request.POST["step"]
    max_step = request.POST["max_step"]

    print(post_keys)
    for post_key in post_keys:
        if post_key in vragen.keys():  # voor het opslaan van een ingevulde antwoord
            print(post_key)
            vraag_code_button = post_key
            antwoord_op_vraag = request.POST[vraag_code_button]
            vraag_code_toelichting = post_key + "_toelichting"
            antwoord_toelichting = request.POST[vraag_code_toelichting]

            antwoord = Antwoord(
                antwoord_vraag=vragen[vraag_code_button],
                antwoord_collectie=collectie,
                antwoord_body=antwoord_op_vraag,
                antwoord_toelichting=antwoord_toelichting
            )
            antwoord.save()

    Collectie.objects.update(collectie_flag=flag)  # updates the collectie met een flag title
```

Figuur 19: Het codefragment voor het opslaan en versturen van een Antwoord model naar een Collectie

Het idee was eerst om alle vragen op een enkele webpagina te tonen. Echter merkte ik snel dat de vragenlijst snel onoverzichtelijk en traag wordt wanneer er veel vragen worden ingeladen. Dat is vooral onhandig aangezien een Self Assessment meer dan 200 vragen bevat. Hierdoor wordt het requirement voor de gebruiksvriendelijkheid niet gehaald.

Mijn eerste oplossing was om bepaalde vragen in tabs binnen de webpagina te tonen. Hiermee kan je op de volgende knop klikken als je klaar bent met het invullen van een tab. Elke tab bevat ongeveer 10 vragen. Echter zou dat ook niet gebruiksvriendelijk zijn omdat er alsnog veel moet worden geklikt. Ik had dat idee gedeeld met mijn opdrachtgevers en ze hadden dezelfde mening als ik. Daarom bedacht ik om een webpagina altijd alle vragen van een enkele categorie te laten weergeven en met een knop de pagina te laten verversen met de volgende categorie met de bijbehorende nieuwe set van vragen.

Figuur 20: Een pagina van een vragenlijst die alleen een enkel categorie weergeeft

Als er eenmaal voor een vragenlijst wordt gekozen laadt de pagina dan de eerste categorie met al zijn vragen. Als een vraag met “Nee” wordt beantwoord, dan kan je niet naar de volgende pagina wanneer het toelichtingstekst veld van de overeenkomstige vraag niet ingevuld is.

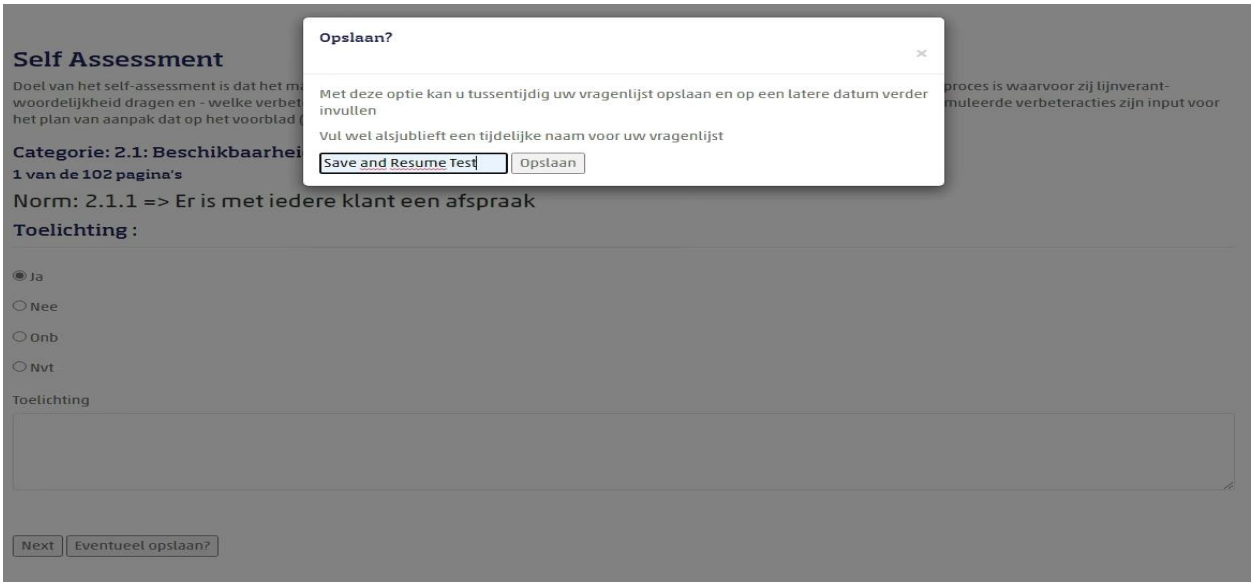
Wanneer een vraag met “Onb” werd beantwoord, dan komt er een pop up venster om te vragen of alle vragen binnen de categorie ook automatisch naar “Onb” worden beantwoord met een standaard toelichting. Aangezien het een van de eisen is. Dit werd gedaan met een combinatie van Javascript en Bootstrap Modal omdat er HTML variabelen dynamisch moesten worden gemanipuleerd (DOM Manipulation), een functionaliteit dat niet mogelijk is met Python.

Wanneer alle vragen binnen een pagina worden beantwoord, druk je op de knop ‘Next’ en dan wordt de pagina ververst en wordt de volgende categorie met zijn vragen ingeladen. De vorige ingevulde vragen worden vervolgens verzonden met een POST request naar de backend. Wanneer alle categorieën worden beantwoord en als de Rapportage User pagina bereikt is, dan worden alle ingevulde vragen met een POST request opgeslagen in een Collectie model.

7.3.2.1 Opslaan en Hervatten

Tijdens het presenteren van een demo aan mijn opdrachtgevers kwamen ze op een idee om nog een belangrijke software requirement toe te voegen als een Must Have. In dit geval gaat het om de mogelijkheid om tijdens het invullen van een vragenlijst halverwege op te kunnen slaan en op een later tijd verder te kunnen gaan met het invullen. Het was een requirement waarvan wij met zijn drieën niet eerder hadden gerealiseerd dat dit een Must Have moest zijn. Totdat wij merkten hoe lang het invullen van een volledige self-assessment op de webpagina kan duren. In eerste instantie dacht ik dat dit een ingewikkelde taak zou zijn, omdat de applicatie ervoor moet zorgen dat alleen een gedeelte van een vragenlijst kan worden opgeslagen met behulp van een POST request. Vervolgens moet alle informatie worden opgehaald en opstarten bij de laatste onbeantwoorde vraag door middel van een GET request.

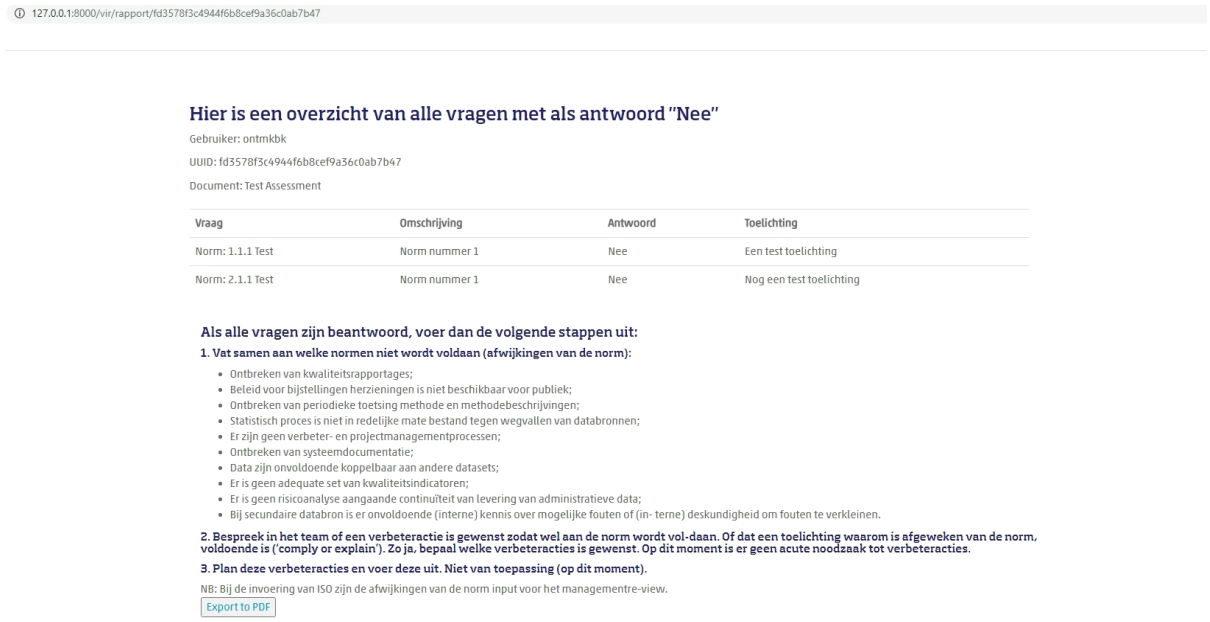
Als oplossing bedacht ik als volgt; telkens wanneer een invuller halverwege zijn/haar vragenlijst opslaat en een naam ingeeft, dat dan alle ingevulde antwoorden tijdelijk opgeslagen worden in de database. Wanneer de gebruiker de vragenlijst hervat en alles invult, dan worden de oude en nieuwe ingevulde antwoorden opgeslagen in een nieuw Collectie object. Dit Collectie object wordt dan opgeslagen in de database.



Figuur 21: Het scherm tijdens het tussentijds opslaan van een vragenlijst

7.3.3 Rapportage User

Wanneer een VirDocument wordt opgeslagen en verstuurd naar de database komt de gebruiker terecht op een overzichtspagina waarop alle vragen die met “Nee” zijn beantwoord te zien zijn. De bijbehorende vervolgstappen worden ook op deze pagina weergegeven. Deze pagina is te zien op het onderstaande plaatje. Vanuit mijn opdrachtgever is het een eis dat de invuller kan zien aan welke normen niet zijn voldaan, en dat het duidelijk wordt wat in dat geval de vervolgstappen zijn voor verbetering. Voor het ophalen van alle beantwoorde vragen met “Nee” wordt het huidige Collectie object opgehaald op basis van hetzelfde UUID. Ten slotte worden alleen de ingevulde vragen met antwoord “Nee” eruit gefilterd.



Figuur 22: Het Rapportage User pagina

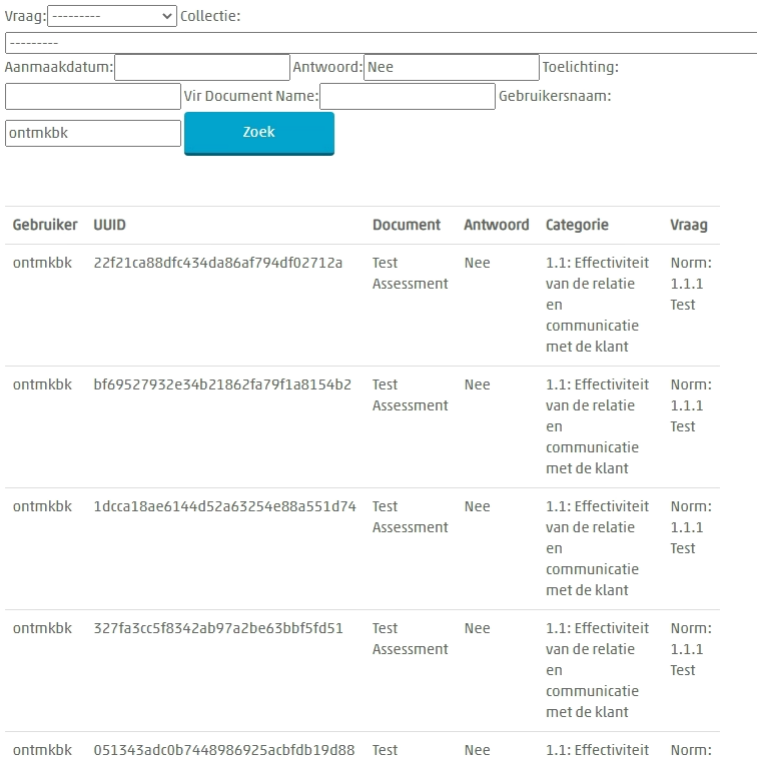
Verder heeft deze webpagina ook een speciale knop waardoor het mogelijk is om de hele overzichtspagina op te slaan en te exporteren naar een pdf-bestand. Hierdoor kan een invuller het pdf-bestand gebruiken voor aan zijn eigen werk gerelateerde onderzoeksdoeleinden. Om deze functionaliteit mogelijk te maken heb ik gebruik gemaakt van de libraries ReportLab en XHTML2PDF.

Ik heb gekozen voor Reportlab omdat het een populaire python library is die wordt gebruikt door vele organisaties onder anderen HP, NASA en Wikipedia voor het genereren van pdf-bestanden. XHTML2PDF is een andere open source library dat Reportlab gebruikt voor het genereren van PDF bestanden op basis van html templates. Normaliter kan je een Python library automatisch laten downloaden, installeren en configureren met ‘’Pip’’. Met Pip kan je een python library met al zijn dependencies automatisch laten downloaden en configureren. Echter is dat voor mij niet mogelijk, omdat ik geen gebruik kan maken van Pip’s download functie, met als reden dat het Ontwikkel VM een zeer beperkt internet toegang bevat. Hiervoor moest ik dan handmatig een library met al zijn dependencies downloaden vanuit CBS’s library lijst als ze beschikbaar zijn. Indien ze niet beschikbaar zijn dan moet ik toestemming vragen aan de CBS Service Desk of ik direct de library mag downloaden vanuit zijn oorspronkelijke website. Na het downloaden moet ik ze dan één voor één installeren en configureren zodat ik uiteindelijk de library kan gebruiken. Naar mijn mening was dat het grootste nadeel om te werken met het VM omdat het handmatig installeren en configureren van libraries erg tijdrovend is.

Voor beide libaries is er veel documentatie beschikbaar op het internet, waardoor ik oplossingen kon vinden voor de problemen waar ik tegen ben gelopen.

7.3.4 Rapportage Admin

Nadat een gebruiker is ingelogd als een administrator (zoals een kwaliteitsbeheerder) krijgt deze persoon een extra optie op de Indexpagina, een doorverwijslink naar de rapportage admin pagina. Via deze pagina kan een gebruiker alle informatie over opgeslagen VirDocumenten opzoeken met filter opties. Dit houdt in dat er direct een verbinding wordt gemaakt met het template en de gehele database. Een gebruiker kan bijvoorbeeld opzoeken wat het meest voorkomende antwoord is op een specifieke vraag of categorie. Een administrator kan zelfs dieper onderzoek verrichten door bijvoorbeeld te kijken welke gebruiker welk VirDocument heeft ingevuld, op welke datum dit is gedaan en vervolgens zijn of haar ingevulde vragen te bestuderen.



Figuur 23: De Rapportage Admin pagina

Hiervoor heb ik gebruik gemaakt van een speciale library genaamd Django-filters. Met deze library kan ik vanuit de View dynamisch SQL queries uitvoeren gebaseerd op de meegegeven zoekopties. Dit bespaarde mij veel codeerwerk vergeleken met Python's standaard filtering functie.

7.3.5 Uitdagingen

Dit bleek een van de uitdagendste onderdelen om uit te werken. Dat kwam voornamelijk door de volgende twee punten waar ik op vast liep.

7.3.5.1 Verzenden van een ingevulde vragenlijst

De grootste uitdaging was het verzenden van een ingevulde vragenlijst naar de database. Om vragen weer te geven per vir-document, moest ik een manier vinden om ze weer te geven met HTML terwijl er een database koppeling is en validatie checks gedaan moeten worden. Als eerste oplossing probeerde om gebruik te maken van Django's FormModel, een standaard library van Django. De library zorgt ervoor dat het template als een normaal formulier wordt gegenereerd. Dat alleen regelt de bovengenoemde problemen met zo min mogelijk ontwikkelwerk. Hierdoor zou ik dan meer tijd hebben om aan andere onderdelen en problemen te werken.

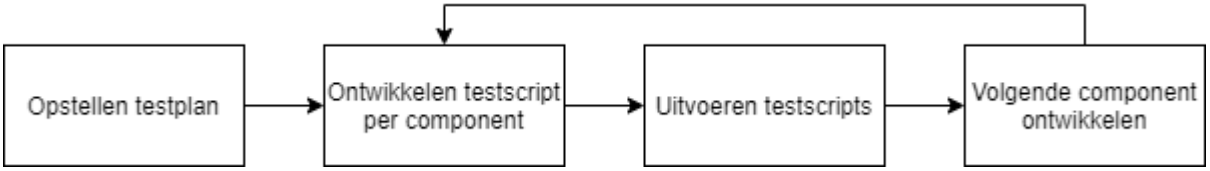
Echter had ik problemen met het weergeven en opslaan van antwoorden op een template via FormModel. Uiteindelijk had ik toch geen gebruik gemaakt van Django's FormModel. In plaats daar van heb ik de logica in de Details View geïmplementeerd voor het opslaan en antwoorden, terwijl het Template zorgt voor het weergeven van vragen en antwoorden met een mix van HTML en Python code. Nadeel is wel dat er geen gebruik kan worden gemaakt van typische FormModel functies, wat handig zou zijn voor deze opdracht, met name de automatische foutafhandeling en de sorteeropties.

7.3.5.2 DOM Manipulation

Een van de eisen (Requirement #21) was dat een invuller een toelichting moet toevoegen wanneer een vraag niet met "Ja" werd beantwoord. Om dit probleem op te lossen zal de webpagina aangeven dat het niet mogelijk is om een ingevulde vir-document op te slaan, totdat het tekstveld waarin de toelichting moet worden gegeven niet leeg is. Het dynamisch veranderen van een html-element (in dit geval het tekstveld) heet Domain Object Model (DOM) Manipulation. Echter kan Python Django niet dynamisch een DOM manipulatie uitvoeren zonder de pagina te verversen, waardoor de invuller helemaal opnieuw moet beginnen met het invullen[12].

Om dit probleem op te lossen heb ik gebruik gemaakt van Javascript, een programmeertaal dat voornamelijk wordt gebruikt voor web development. Met Javascript is DOM Manipulation wel mogelijk. Omdat Javascript werkt met HTML, is het mogelijk om Javascript toe te passen in een Python Django project, omdat Javascript wordt opgeroepen vanuit een HTML Template.

8. Deelproduct 3: Testen



8.1 Opstellen testplan

Dit hoofdstuk beschrijft de testmethodes die zijn gebruikt voor de onderdelen die werden ontwikkeld tijdens de realisatiefase. Bij het uitvoeren van de testen heb ik gebruik gemaakt van testscripts. De scripts zijn ontwikkeld met behulp van de testtools en frameworks die ik heb gebruikt tijdens mijn afstudeertraject.

Tijdens het opstellen van mijn aanpak heb ik onderzoek gedaan naar de verschillende manieren om een Django applicatie te testen. Gedurende mijn opleiding was ik alleen bekend met Unit Testing, een veelgebruikte methode om software te testen. Met unit testing worden afzonderlijke componenten en modules getest om vast te stellen of er problemen zijn. Het gaat om de functionele juistheid van de componenten. Het hoofddoel is om elke eenheid van het systeem te isoleren om de defecten te identificeren, te analyseren en te verhelpen.

Mijn doel is om te kijken of de MVT componenten correct samenwerken. Dit was vooral van belang omdat ik al die onderdelen zelf heb ontwikkeld. Ik heb er daarom voor gekozen om gebruik te maken van zowel Unit als Integration Testing. Met Unit kan ik nalopen of de MVT componenten goed samenhangen met elkaar. Met Integration kan ik nalopen of de functionaliteit, prestaties en betrouwbaarheid tussen de MVT componenten goed worden geïntegreerd.

Omdat de Models en Views onderwater worden uitgevoerd, beschouw ik het als de backend van de applicatie. De Templates zijn zichtbaar voor de gebruiker waardoor ik het testen van de Templates als front-end beschouw.

Normaliter wordt er pas getest wanneer de realisatiefase bijna compleet is. Alles wordt dat in een keer getest. Deze methode heet Big-Bang Testing. Een nadeel is dat wanneer er iets mis is gegaan tijdens het testen het te laat is om deze fout te herstellen in de applicatie. Daarom besloot ik om mijn onderdelen incrementeel te testen, ofwel Incremental Testing. Hierbij heb ik genoeg tijd om de constateringen uit de gefaalde tests te verbeteren.

Met Incremental Testing heb je drie verschillende manieren om de methode aan te pakken. Ten eerste kan je tijdens het testen componenten tijdelijk vervangen met mock componenten, genaamd ‘stubs’. Hierbij begin je het testen van stubs die veel afhankelijkheden hebben, dit

heet Top-Down Testing. Het kan ook omgekeerd worden gedaan. Hierbij begint het met het testen van onderdelen die juist weinig afhankelijkheden hebben. Deze methode heet dan Bottom-Up Testing. Tot slot kunnen de twee bovengenoemde methodes ook worden gecombineerd waardoor alle componenten en functies worden getest en gesimuleerd in plaats van gebruik te maken van stubs. Deze methode heet Hybrid Testing, ofwel Sandwich Testing [13]. In dit geval heb ik besloten om gebruik te maken van Hybrid Testing omdat ik dan zowel unit als integratie tests tegelijkertijd kon uitvoeren voor mijn software.

Na het integration- en unit testing is het ook belangrijk om alles na te lopen of het volledige en geïntegreerde softwareproduct te valideren. Dit soort testen heet System Testing. Dit ga ik doen door de user interface van de software te testen. Op deze manier kan de test zich vooral richten op het gebruiksgemak van de software, de flexibiliteit in de bediening en het vermogen van het systeem om zijn doelstellingen te bereiken. Hiervoor zal ik dan gebruik maken van Usability Testing voor het systematisch testen van de applicatie.

Dit testplan met bijbehorende resultaten is te vinden in de bijlage: Testrapport.

8.2 Ontwikkelen en uitvoeren van testscripts

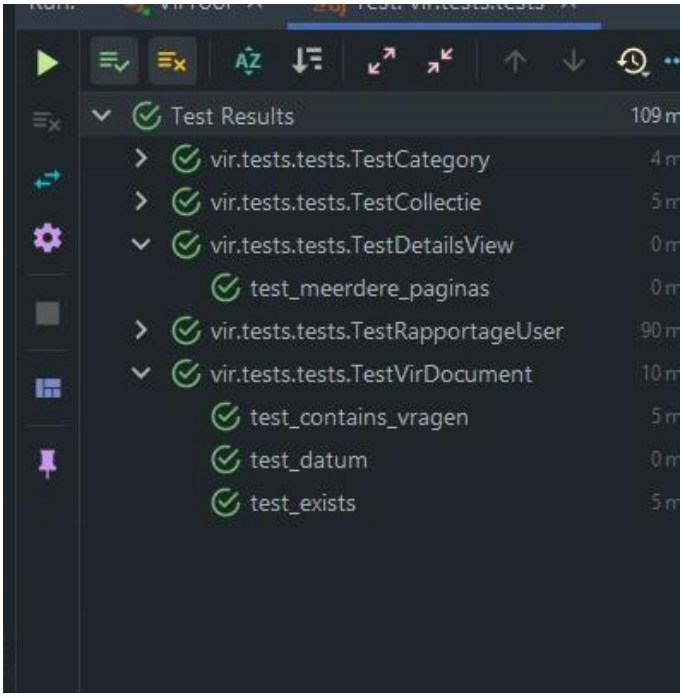
Om de back-end componenten te testen moet er eerst een applicatie worden gebouwd deze kan de testcases automatiseren. Omdat ik met Django werk was het mogelijk om testscripts te schrijven waarmee ik voor elk onderdeel kan worden gecontroleerd of een component of functie naar behoren functioneert. Per scenario wordt er een testscript ontwikkeld. Mocht in de toekomst deze webapplicatie door een andere ontwikkelgroep beheerd worden, dan kunnen zij makkelijk nieuwe testscripts toevoegen of aanpassen met mijn bestaande scripts als basis.

```
class TestRapportageUser(BaseModelTest):
    def test_collectie_opgeslagen(self):
        sa = VirDocument.objects.create(vir_name="SA", vir_description="vir_description",
                                        vir_published_date=timezone.now(),
                                        vir_display_option=VirDocument.BY_QUESTION,
                                        slug=5464861523785)

        Collectie.objects.create(collectie_vir_document=sa, collectie_uuid=sa.slug)
        collecties = Collectie.objects.filter(collectie_vir_document=sa)
        collectie = collecties.all()[0]
        url = reverse("vir-rapportage", args=(collectie.collectie_uuid,))
        response = self.client.get(url)
        self.assertEqual(response.status_code, 200)
        return response
```

Figuur 24: Voorbeeld van een testscript voor het testen van het rapportage user functie

Door het script te runnen met parameter “./manage.py test” worden alle testscripts automatisch uitgevoerd, waarna de resultaten weergegeven worden. Als elke unit test een groen vinkje krijgt, dan betekent dat dat de tests zijn geslaagd. Indien er wordt gewerkt met PyCharm, kan elke test apart worden uitgevoerd dankzij PyCharm’s testing functionaliteit.



Figuur 25: Output van bepaalde succesvolle Unit test resultaten

De resultaten van de uitgevoerde tests zijn ook te vinden in het Testrapport.

8.3 User Interface testen

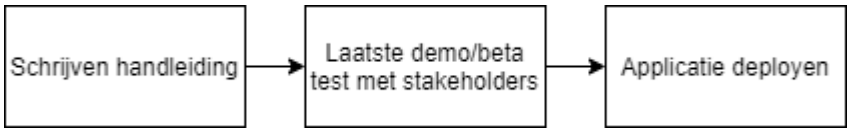
Naast de backend (Models en Views) moeten ook alle templates worden getest, die dienen als user interface en front-end van de applicatie. Hiervoor is Usability Testing van groot belang. Met behulp van deze test methodiek kan worden nagelopen of een gebruiker op de verwachtte manier door de webapplicatie kan navigeren. Usability Testing is tevens een onderdeel van System Testing. Met Python is het bijna niet mogelijk om webpagina's programmatisch te testen zonder externe frameworks of programma’s.

Daarom gebruiken ontwikkelaars binnen het CBS het test-framework Selenium voor het testen van webpagina’s. Selenium is een test framework dat met behulp van voorgeprogrammeerde instructies een webbrowser automatisch aanstuurt zoals een gebruiker dat zou doen, hetzij lokaal of op een externe machine met behulp van een Selenium server. Voor Python heb je ook Selenium Python dat een API aanbiedt om acceptatie en integratie testen te schrijven met Selenium. Omdat Selenium Python beschikbaar was op het CBS VM omgeving besloot ik om gebruik te maken van dat framework voor het testen van mijn templates.

De resultaten van de uitgevoerde UI tests zijn ook te vinden in het Testrapport.

9. Overdracht

Deze fase dient voor het afronden van mijn afstudeeropdracht.



9.1 Schrijven handleiding

Nadat er een complete versie van mijn applicatie ontwikkeld was heb ik een demo gegeven en een presentatie van mijn uitgevoerde Kanban bord aan mijn opdrachtgevers. Zo hebben ze kunnen zien hoe het prototype eruit ziet, functioneert en welke stappen ik heb ondernomen bij het ontwikkelen van het prototype. Daarnaast was het ook een moment om laatste feedback te krijgen voordat ik mijn applicatie door de stakeholders liet gebruiken. Tijdens dit contactmoment was mijn volgende taak om alvast een handleiding te maken voor zowel een invuller als kwaliteitsbeheerder voor het gebruik van dit prototype. Dit was voor mij geen verrassing omdat ik in mijn plan van aanpak had opgenomen om aan een handleiding te gaan werken aan het einde van het ontwikkelen van het prototype. Het doel van een handleiding is om de huidige en toekomstige invullers effectief en correct het prototype te laten gebruiken tijdens mijn afwezigheid.

Omdat het prototype verschillende functies heeft voor een invuller en administrator, werd mij aangeraden om twee aparte handleidingen te maken specifiek bedoeld voor elk soort gebruiker. In de invullers handleiding beschrijf ik hoe je stap voor stap gebruik kan maken van de web applicatie voor het invullen en opslaan van een vragenlijst met bijbehorende regels. De handleiding voor de administrator is een beetje uitgebreider dan van een invuller omdat een administrator ook toegang krijgt tot het beheer van data gebaseerd op (reeds) ingevulde vragenlijsten die alleen voor hem zichtbaar zijn.

Beide handleidingen zijn te vinden als Bijlagen ‘Handleiding Vir-Tool voor de invullers’ en ‘Handleiding Vir-Tool voor de admins”.

9.2 Demo/Beta test en deployment

Na het schrijven van de handleidingen voor een laatste beta test was mijn volgende taak om de applicatie opzetten in een server binnen het CBS’s VM netwerk. Omdat de applicatie een webapplicatie is kan het dan niet zomaar worden verzonden als een bestand naar mijn opdrachtgevers en stakeholders. Want een web applicatie moet continu blijven draaien vanuit een server. Binnen CBS gebruiken ze Cloud Foundry als een Linux server voor het hosten en

opzetten van web applicaties. Om mijn applicatie in Cloud Foundry te laten opzetten moet ik eerst een nieuwe database binnen Cloud Foundry opzetten. Hiervoor moet ik een PostgreSQL database opzetten aangezien dat een standaard en een eis is binnen het CBS. Daarnaast moet ik ook alle libraries die ik had gebruikt voor mijn applicatie configureren zodat ze zonder problemen automatisch worden overgezet naar Cloud Foundry. Echter was dat niet het geval omdat twee libraries die gebruikt worden voor de functies Rapportage Admin en het exporteren naar een PDF compatibiliteit conflicten hadden met een Linux server. Ik had het vervolgens opgelost door middel van brainstormen en het zoeken naar documentatie van de desbetreffende libraries. Nadat mijn applicatie met zijn libraries succesvol werd opgezet had ik vervolgens het unieke URL van mijn applicatie op Cloud Foundry gedeeld met mijn opdrachtgevers.

Nadat mijn opdrachtgevers mijn applicatie hebben getest kreeg ik nog opmerkingen omtrent de applicatie, voornamelijk opmaak en bugs. Vanwege gebrek aan tijd als gevolg van mijn afstudeerverslag had ik samen met mijn opdrachtgevers alle opmerkingen geprioriteerd. Hiermee werd er bepaald welke opmerkingen moeten worden toegepast voordat mijn afstudeertraject voorbij is. Na het toepassen van het belangrijkste opmerkingen had ik de applicatie opnieuw op Cloud Foundry gezet zodat zij vanaf nu verder kan werken met mijn applicatie.

Oorspronkelijk was het idee om mijn applicatie door de stakeholders te laten testen. Jammer genoeg ging dat niet door omdat een groot gedeelte van de stakeholders onbereikbaar waren of niet beschikbaar gedurende deze periode. Volgens mijn planning in mijn plan van aanpak zou dat tijdens mijn laatste 3 weken van mijn traject gebeuren. Echter, liep het uit omdat ik veel tijd had besteed om de twee libraries aan de praat te krijgen op Cloud Foundry. Na een overleg met mijn opdrachtgevers hebben wij besloten dat de stakeholders mijn applicatie kunnen testen nadat ik mijn afstudeerverslag heb ingeleverd. Aangezien ik een fulltime functie kreeg aangeboden als een software developer na afronding van mijn afstudeerperiode.

10. Evaluatie/Reflectie

10.1 Evaluatie werkzaamheden

Tijdens mijn afstudeeropdracht liet ik mijn werk regelmatig reviewen door ontwikkelaars binnen CBS en opdrachtgevers. Voor mijn afstudeerdossier zelf liet ik het reviewen door mensen in mijn vrienden en familiekring die kritisch naar verslagen kijken voor hun beroep. Ieder keer wanneer ik een component of een belangrijke functionaliteit had geïmplementeerd, liet ik het reviewen door een ontwikkelaar binnen CBS om te kijken of ik juiste technieken zoals design patterns correct had toegepast en of het invloed heb op het prestatie van de applicatie. Naar mijn mening vond ik dat belangrijk voor mijn ontwikkeling omdat ik op zo'n manier nieuwe technieken kan toepassen in de toekomst wanneer ik een soortgelijke applicatie moet gaan ontwikkelen.

Mijn plan van aanpak en requirements rapport heb ik regelmatig laten nakijken door de opdrachtgevers. Dit heb ik gedaan om te verifieren dat de structuur van mijn documentatie naar wens is. Dit was belangrijk omdat een goed plan van aanpak en een requirements rapport de basis zijn voor een succesvolle ontwikkelfase en product.

Tijdens mijn afstudeertraject waren er meerdere momenten waarin ik vastliep met mijn werk, voornamelijk tijdens mijn realisatiefase in hoofdstuk 7. Dat kwam vooral omdat ik voorafgaand aan deze opdracht weinig ervaring had met Python, laat staan met Django. Ik had ook moeite met het realiseren van de eis om een vragenlijst op te splitsen in meerdere pagina's en het verzenden van een zowel compleet als half ingevulde vragenlijst omdat ik geen gebruik kon maken van Django Forms. Door het doornemen van veel tutorials, documentatie en het bezoeken van hulpforums over Python Django en het eventueel hulp vragen aan mijn collega's heb ik een groot gedeelte van mijn problemen weten op te lossen. Hierdoor heb ik geleerd om eerder hulp te vragen zodat ik niet vastloop. Jammer genoeg omdat ik veel tijd besteedde aan twee bovengenoemde problemen, had ik uiteindelijk niet genoeg tijd om requirements R5 en R12 (zie bijlage Requirementsrapport) volledig te realiseren.

Omdat ik nog erg gewend was aan het ritme als student vond ik het lastig om tijdens de zomervakantie te focussen op mijn werk. De consequentie was dat ik voor een korte periode achter liep met mijn werk. Maar ik kon wel snel een gedeelte van die gemiste tijd inhalen door tijdens een weekend door te werken. Nog een verbeterpunt was dat ik in het begin van mijn opdracht aannam dat ik met C# zou werken, echter was dat niet het geval waardoor ik met een taal moest werken waar ik geen ervaring mee had. Dit kan een negatieve invloed hebben wanneer ik werkzaam ben bij een bedrijf of organisatie als ik veel niet-geverifieerde aannames doe. Dit kan leiden tot het plotseling wijzigen van plan van aanpak, wat ook meer stress kan veroorzaken. Ik heb geleerd door te vragen en te onderzoeken of mijn aannames juist zijn.

Ten slotte was thuiswerken voor mij een unieke ervaring tijdens mijn afstudeertraject vanwege Covid-19. Ondanks dat ik online lessen had gekregen tijdens mijn studie vanwege de Covid-19

pandemie, was het voor mij toch een andere en nieuwe ervaring om te werken op afstand. Gelukkig maakt CBS veel gebruik van Zoom, Slack en Outlook waardoor de communicatie tussen werknemers enigszins soepel verloopt. Wanneer ik vastliep met mijn werk, kon ik dus een bericht sturen naar een collega om te vragen of hij/zij op Zoom kan gaan om samen naar het probleem te kijken. Hierdoor kon ik het risico met het achterlopen van mijn opdracht grotendeels te beperken

10.2 Evaluatie Beroepstaken

A1 Analyseren probleemdomein & opstellen probleemstelling

Voor deze beroepstaak hield ik tijdens het begin van mijn opdracht gesprekken met mijn opdrachtgevers en heb ik interviews gehouden met de stakeholders over het probleemdomein. Deze gesprekken komen voor in hoofdstuk 3.4 en 4. Aan de hand van deze gesprekken heb ik een plan van aanpak opgesteld met daarin een beschrijving van het probleemdomein, de probleemstelling en de doelstelling. De uitgevoerde werkzaamheden zijn terug te vinden in hoofdstuk 5.2. De plan van aanpak dat ik twee keer heb gedeeld met mijn opdrachtgevers om te bespreken of de wensen voor dit project duidelijk waren en hoe deze konden worden geïmplementeerd. Daarnaast onderzocht ik zelf ook documentatie rondom self-assessments vanuit het CBS intranet met behulp van deskresearch, en vulde ik zelf een sjabloon in om te ervaren waarom het invullen zo inefficiënt was.

A2 Informatie vergaren, analyseren & verwerken

Deze beroepstaak werd uitgevoerd tijdens de requirementsfase in hoofdstuk 4. In deze fase had ik kwalitatieve ongestructureerde interviews gehouden met alle stakeholders over hun ervaring met het huidige self-assessment en hun wensen voor een nieuwe product. Hiervoor had ik eerst aparte vragenlijsten gemaakt voor invullers, statistische onderzoekers en kwaliteitsbeheerders. De gesprekken hebben mij geholpen bij het vergaren van informatie en mijn vragen beantwoord.

Op basis van de informatie uit de kwalitatieve ongestructureerde interviews heb ik software requirements opgesteld volgens het MoSCoW principe. Hiermee kon ik makkelijker aangeven welke eisen een hogere prioriteit krijgen. Daarnaast had ik gebruik gemaakt van CBS’s intranet en eigen wiki om meer informatie te zoeken en te bestuderen rondom het self-assessment en het principe rondom de vir-documenten.

C1 Ontwerpen Software

Een belangrijke onderdeel van mijn opdracht was natuurlijk het ontwerpen van nieuwe software. Hiervoor had ik voor de Model View Template architectuur gekozen voor het ontwerpen van de software, omdat Python Django geoptimaliseerd is voor deze architectuur. Als resultaat had ik uiteindelijk een klassendiagram gemaakt dat de Models, Views, Templates en helper methodes bevat. Daarnaast had ik ook een Process Flow Diagram geschetst om de wenselijke werking van de software in kaart te brengen. Ten slotte had ik ook mocks gemaakt om te laten zien hoe het user interface van de applicatie daadwerkelijk eruit komt te zien. Deze beroepstaak werd voornamelijk uitgevoerd in hoofdstuk 6.

C2 Ontwerpen datamodel & database

Om te zorgen dat de nieuwe software informatie kan halen vanuit de ingevulde self-assessments is het noodzakelijk om een database te ontwerpen om al deze gegevens op te slaan. Tijdens het begin van mijn opdracht kreeg ik te horen dat PostgreSQL een vereiste is voor het ontwikkelen van databases. Desondanks heb ik ontwikkeld met SQLite3, omdat het standaard was met Python Django, kon ik gemakkelijk het software migreren van SQLite3 naar PostgreSQL zonder het datamodel aan te passen. Voor het maken van een datamodel had ik gekozen om het te ontwerpen volgens het Entity Relationship Diagram. Ik had het datamodel reeds op een laag niveau aangepast (denk aan nieuwe attributen) tijdens de realisatiefase. Deze beroepstaak werd voornamelijk uitgevoerd in hoofdstuk 6.

D1 Realiseren van software

Deze beroepstaak werd toegepast tijdens de realisatiefase in hoofdstuk 7, kort na de ontwerpfase. Deze software had ik in een incrementele werkwijze ontwikkeld, dit houdt in dat ik terug kan gaan naar een deel van de implementatie mocht deze fouten bevatten blijkend tijdens het testen. Ik liet mijn code regelmatig controleren door een senior ontwikkelaar gespecialiseerd in Python Django binnen CBS voor feedback of wanneer ik vastliep. Uiteindelijk was het gelukt om de applicatie te realiseren met al hun Must Have requirements volgens het MosCoW principe.

GC Kritisch, onderzoekend en methodisch werken

Deze beroepstaak was vooral van belang bij de onderzoeken gedurende mijn voorbereiding, requirements, realisatie en testfasen. Ik had in mijn gehele afstudeertraject gebruik gemaakt van het agile werkmethode Incrementeel Kanban vanwege verschillende redenen dat ik had benoemd in hoofdstuk 3.5. In het begin had mijn opdrachtgever aangegeven dat ze heel graag met Kanban werken. Naar aanleiding daarvan had ik gekeken wat Kanban inhoud omdat dat methode niet werd behandeld in mijn studie. Als resultaat kan ik zien waarom Beleidstaf gebruik maakt van Kanban vanwege zijn agile bord dat werkflow overzichtelijk maakt vergeleken met Waterval, een vergelijkbare methode.

Tijdens mijn voorbereiding fase ging ik verschillende websites onderzoeken om te kijken of ik met Python of Blaise zal werken voor mijn afstudeeropdracht. Nadat ik had gekozen voor Python ging ik toen na welk Python web framework het beste zou zijn voor deze opdracht. Hiervoor had ik verschillende websites doorgenomen om te kijken wat de beste keuze is op basis van beschikbare documentatie en moeilijkheidsgraad.

Persoonlijk zou ik liever gebruik hebben gemaakt van C# ASP.NET voor het bouwen van de webapplicatie. Mijn redenering hierachter was dat C# ASP.NET een compileer taal is terwijl Python Django een script taal is. Een compileer taal is efficiënter dan een script taal want een compiler heeft meer tijd om van te voren optimalisatie stappen toe te passen. Met een script taal kan dit niet. Deze redenatie is tevens te vinden in hoofdstuk 3.4.

Ook had ik gekeken naar voorbeelden voor het opstellen van een plan van aanpak en methodisch onderzoeken via CBS intranet en gewoon via het internet zelf. Tijdens en wanneer ik klaar ben met een versie van mijn plan van aanpak en mijn bijlagen, deel ik ze met mijn opdrachtgevers voor eventuele feedback en of alles klopt tot hen behoren. Dat is voornamelijk terug te vinden in hoofdstuk 3.3 tot en met 3.5.

Tijdens mijn ontwikkelfase had ik verschillende cursussen en tutorials gevolgd voor het leren over de Python Django taal. Telkens wanneer ik een functie van een component had geïmplementeerd, vroeg ik daarna voor feedback aan verschillende ontwikkelaars binnen CBS om te kijken of het beter kan worden geïmplementeerd.

GF Leren leren

Voor deze beroepstaak had ik voornamelijk drie grote leermomenten gehad die belangrijk zullen zijn voor mijn werk carrière na mijn studie. Het eerste leermoment kwam voor in hoofdstuk 3.5, toen ik tijdens mijn voorbereidingsfase te horen kreeg dat het web app geschreven moet worden in Blaise of Python, twee talen waar ik weinig kennis over bevat.

Nadat ik uiteindelijk voor Python Django had gekozen, moest ik Python Django onder de knie krijgen zodat ik een eindproduct kan leveren met een hoge kwaliteit. Hiervoor had ik dan Udemy cursussen aangeschaft en in de tussentijd tutorials gevolgd op het internet. Vervolgens vroeg ik ook regelmatig om feedback en tips aan CBS ontwikkelaars die ervaren zijn met Python Django. Uiteindelijk was het mij toch gelukt om succesvol een web applicatie te bouwen in Python Django.

Het volgende leermoment kwam voor in heel hoofdstuk 7. Voor bepaalde functies was het gebruik van een specifieke Python module noodzakelijk om bepaalde requirements te implementeren, zoals de functionaliteit dat werd doorgenomen in hoofdstuk 7.3.3 en 7.3.4. Echter moest ik Python libraries handmatig gaan downloaden, installeren en configureren omdat het CBS Ontwikkel VM internettoegang zeer beperkt is vanwege veiligheidsredenen.

Normaliter heeft Python een ingebouwde installer genaamd “pip” waardoor je automatisch een library met al zijn benodigde modules kan downloaden en configureert.

Gelukkig had een collega binnen het CBS mij aangeboden om een kleine demonstratie te geven over hoe ik een library handmatig installeer en configureer voor zowel productie als development omgeving. Dat gebeurde tijdens een gesprek dat plaatsvond in hoofdstuk 4.1. Indien ik bepaalde stappen vergat, kon ik dan bepaalde informatie terugkoppelen in het CBS intranet. Sinds die demonstratie was ik in staat om Python libraries te beheren zonder gebruik te maken van een internet verbinding of met Pip.

Mijn laatste grootste leermoment was het continu bevorderen van de Nederlands taal, aangezien Nederlands niet mijn moedertaal is. Hierdoor had ik moeite met bepaalde grammaticale regels tijdens schrijven van bepaalde documentatie, voornamelijk dit afstudeerverslag. Als voorbeeld ik had toen nog moeite met lidwoorden en woorden die eindigen met ‘d’ of ‘t’. Om mijn documentatie te schrijven op hoog Nederlands niveau liet ik voornamelijk mijn afstudeerverslag kritisch nakijken door verschillende personen welke de taal op een hoog niveau beheersen. Hierdoor kon ik zien welke fouten ik had gemaakt, en hoe ik ze kon corrigeren en toepassen wanneer ik verder werk aan mijn verslag of aan andere documentatie. Als resultaat was ik in staat om deze afstudeerdossier in te leveren met een hoog niveau Nederlands.

Literatuurlijst

1	*CBS Organogram*. (z.d.). CBS Organogram. https://www.cbs.nl/-/media/_pdf/organogram-nl.pdf?la=nl-nl
2	Winterbottom, M. (2021, 16 maart). *Flask vs. Django: Which Web Framework is Best?* Udemy Blog. https://blog.udemy.com/flask-vs-django/
3	*Django documentation | Django documentation | Django*. (z.d.). Djangoproject. Geraadpleegd op 18 augustus 2021, van https://docs.djangoproject.com/en/3.2/
4	vanderwardt@agilescrumgroup.nl. (2020, 24 december). *Wat is Kanban? Een volledige uitleg van de Kanban Methode + template*. Agile Scrum Group. https://agilescrumgroup.nl/wat-is-kanban-methode/
5	*Atlassian. (z.d.). Kanban vs Scrum. Geraadpleegd op 18 augustus 2021, van https://www.atlassian.com/agile/kanban/kanban-vs-scrum*
6	M. (2020, 3 november). *Leer prioriteiten stellen door de MoSCoW-methode*. OMCBase. https://www.omcbase.nl/moscow/

7	*CBS Intranet. (z.d.). CBS Intranet (niet toegankelijk buiten het CBS omgeving). https://cbsintranet.net/default.aspx*
8	Kumar, N. (2021, 30 juli). *How the Model View Controller Architecture Works – MVC Explained*. FreeCodeCamp.Org. https://www.freecodecamp.org/news/model-view-architecture
9	K., & K. (2021, 20 maart). *Django Model View Template (MVT) Overview*. Onlinetutorialspoint. https://www.onlinetutorialspoint.com/django/django-model-view-template-mvt-overview.html
10	Percival, H., & Gregory, B. (2020). *Architecture Patterns with Python: Enabling Test-Driven Development, Domain-Driven Design, and Event-Driven Microservices* (1ste ed.) [E-book]. O'Reilly Media.
11	Sassoulas, P. (z.d.). *GitHub - Pierre-Sassoulas/django-survey: A django survey app, based on and compatible with "django-survey" (ie: you can migrate your old django-survey app and its data), but ported to python 3 and allowing export and report as CSV or PDF. Available on PyPi as 'django-survey-and-report'*. GitHub. Geraadpleegd op 18 augustus 2021, van https://github.com/Pierre-Sassoulas/django-survey
12	*Update DOM without reloading the page in Django*. (2017, 27 augustus). Stack Overflow.

	https://stackoverflow.com/questions/45906858/update-dom-without-reloading-the-page-in-django
13	*Nanda, V. (2021, 13 juli). What is Sandwich Testing (Definition, Types, Examples)? Sandwich Testing. https://www.tutorialspoint.com/what-is-sandwich-testing-definition-types-examples*