Architecturale Verandering van Back-end applicatie naar Vue SPA

Afstudeerverslag

	Saxion HBO-ICT - 2020/2021
Opdrachtgever	Mobina IT
Student	A. V. Timan
Begeleider	A. van den Berg
Start	08/02/2021
eind	02/07/2021
Versie	1

1 Inhoud

2 Inleiding

Dit verslag beschrijft een project uitgevoerd bij Mobina IT. Mobina IT ontwikkelt een webapplicatie die gebruikt wordt om bedrijven te ondersteunen bij het implementeren van wijzigingen van hun bedrijfsprocessen.

De huidige applicatie is gebaseerd op een server-side rendering (SSR) van de front-end door Django templates met Vue interface componenten. De applicatie laat te wensen over en Mobina is geïnteresseerd in een nieuwe applicatie-architectuur.
Het bedrijf loopt tegen problemen aan binnen hun huidige applicatie-architectuur. De voornaamste reden voor deze problemen is de verandering van focus van de de applicatieontwikkeling. Het bedrijf is de afgelopen jaren meer gaan ontwikkelen in de front-end van de applicatie om de gebruikers van de applicatie een betere ervaring te geven. Dit houdt in dat er veel nieuwe of bestaande logica en componenten naar de front-end zijn verplaatst. Omdat er weinig prioriteit werd gelegd op de integratie tussen de front-end en de back-end is de applicatie langzamerhand groter en ingewikkelder geworden. De huidige set-up is volgens het bedrijf niet meer geschikt voor de grote hoeveelheid componenten die ontwikkeld zijn.

Mobina IT is geïnteresseerd in de haalbaarheid en het rendement van het ombouwen van de applicatie naar een zogenaamde single-page application (SPA) gebaseerd op het Vue ecosystem. Er wordt verwacht dat het toepassen van een dergelijke architectuur vooral voor verbeteringen zorgt in prestatie, onderhoudbaarheid en gebruikerservaring.
Mobina maakt op dit moment gebruik van Django en Vue voor het ontwikkelen van applicaties die naar een eigen architectuur ontworpen worden. De nieuwe gewenste architectuur dient nog steeds gebruik te maken van Vue en Django. Vue heeft recent een nieuwe versie beschikbaar gemaakt met moderne features en Mobina ziet dit als het uitgesproken moment voor een herstructurering.

Omdat een complete revisie van de applicatie-architectuur een significante impact heeft, moet eerst worden onderzocht wat de gewenste architectuur van de applicatie zou zijn, hoe groot de eventuele voordelen zijn en hoeveel werk de verandering met zich meebrengt. De vragen waar antwoord op wordt gezocht in dit verslag zijn:

“Hoe ziet een front-end/back-end gesplitste SPA met Django back-end eruit voor Mobina?”

1. Wat zijn de frameworks/technologieën die hierbij van toepassing zijn?
2. Hoe groot zijn de voordelen van een dergelijke applicatie versus de huidige architectuur?
3. Wat zou de impact zijn van de uit te voeren veranderingen?

Om antwoord te vinden op deze vragen is dit project in vier fases uitgevoerd: Orientatie, onderzoek, uitvoering en samenvatting.

In de oriëntatiefase is onderzocht wat de huidige stand van zaken is op het gebied van de relevante onderwerpen voor dit project. Er is onderzocht welke van de beschikbare technologieën Mobina zouden kunnen baten in de nieuwe applicatie-architectuur. Er zijn een aantal opties besproken met Mobina over deze technologieën. Met de gemaakte selectie is er later in het project een prototype opgesteld.

In de onderzoeksfase is er een diagnose gesteld van de actuele staat van de Mobina applicatie. Er is gekeken naar wat de applicatie-architectuur op dit moment gebruikt en hoe de flow van de applicatie er uit ziet. Er is met Mobina gesproken over de wensen van het prototype van de nieuwe applicatie. De non-functionele eigenschappen waar de nieuwe architectuur verbetering in moet brengen zijn bepaald aan de hand van ISO-25010 en kritieke prestatie-indicatoren.

In de uitvoering is er een prototype gerealiseerd van een kennisportaal applicatie die volgens de single page application architectuur functioneert. De gewenste eisen van het bedrijf zijn hier in verwerkt. Dit prototype kan gebruikt worden door Mobina om als voorbeeld aan te houden in de ontwikkeling van de huidige applicatie.

In de samenvattende fase is er gereflecteerd op de opzet van het prototype en wat de aanbevelingen zijn voor Mobina wanneer deze nieuwe architectuur wordt toegepast. Er zijn een business case en een projectproductomschrijving opgezet om Mobina te helpen met het project om een nieuwe architectuur te implementeren.

2.1 Leeswijzer

In hoofdstuk 3 wordt een inhoudelijke samenvatting gegeven op de resultaten van de bevindingen van dit onderzoek. In hoofdstuk 4 bespreek ik de methode zoals deze is bedacht aan het begin van dit project. Hoofdstuk 5 presenteert uitleg en achterliggende informatie van de relevante verschillen tussen het de huidige situatie en de wensen van Mobina. Hoofdstuk 6 bevat de analyse van de applicatie en de bevindingen die daaruit herleid zijn. In hoofdstuk 7 wordt het prototype gepresenteerd. In hoofdstuk 8 worden onderzoeksrichtingen besproken die niet relevant zijn gebleken voor dit project samen met advies gebaseerd op bevindingen tijdens ontwikkeling.

3 Samenvatting

Het onderzoek dat in dit verslag beschreven staat, is uitgevoerd bij Mobina IT. Tijdens dit onderzoek is er gekeken naar de haalbaarheid en het rendement van een revisie van de huidige applicatie-architectuur. Tijdens deze revisie wordt de huidige applicatie omgebouwd naar een single-page application (SPA), gebaseerd op het Vue ecosystem. Voor dit onderzoek is een proof of concept gecreëerd, waarbij er gekeken is naar een front-end/back-end gesplitste SPA gemaakt met een Django back-end. Om het proof of concept te ontwikkelen, is er onderzoek gedaan naar: de frameworks en technologieën die hierbij van toepassing zijn, hoe groot de voordelen zijn van een dergelijke applicatie versus de huidige architectuur van Mobina en wat de impact is van het uitvoeren van de revisie.
Dit onderzoek is gedaan in vier fases: Orientatie, onderzoek, uitvoering en samenvatting.

Tijdens de oriëntatie is er gekeken naar de verschillende relevante onderwerpen voor dit onderzoek. Er is gekeken naar Vue, de daarvan afgeleide frameworks, optionele packages, onafhankelijke tooling en de werking van Vue als SPA. Ook is er gekeken naar Django, het Django REST framework en de werking van Django als MPA.

De onderzoeksfase is gespecificeerd op de analyse over de huidige staat van de Mobina applicatie. Ter referentiekader is de ISO 25010 gebruikt. Dit is een standaard die de eigenschappen van kwaliteit van een softwareproduct of -systeem beschrijft. Ook zijn er SMART kritieke prestatie-indicatoren opgesteld, zodat bepaald kan worden of verbeteringen behaald zijn na het uitvoeren van de revisie. Hierna is er gekeken naar Mobina's huidige applicatie structuur, componenten en dependencies en zijn de wensen van Mobina besproken. Deze zijn meegenomen in het ontwikkelen van het prototype.

Het prototype is gerealiseerd tijdens de uitvoeringsfase. Hierin is een kennisportaal applicatie hervormt, die als proof of concept dient en als voorbeeld genomen kan worden tijdens ontwikkeling van de nieuwe Mobina app. De kennisportaal applicatie is een digitale locatie waar gebruikers toegang hebben tot informatie die ontwikkeld is door de content creators van Mobina. Het prototype gebruikt de huidige modellen en content modeler die door de applicaties van Mobina al geimplementeerd is. Hierdoor kan de huidige applicatie gebruikt worden als fundering voor de nieuwe applicatie. Het prototype heeft een duidelijk gescheiden front-end en back-end. De front-end is een Vue SPA, waarbij de Views en Componenten, de Router en de Store belangrijke aspecten zijn. Voor de back-end is er gekeken naar verschillende onderdelen zoals API, Serializers, Viewsets en Modellen en is er gekeken naar verschillende API Django apps.

De samenvatting fase beschrijft de aanbevelingen voor Mobina over de revisie van de applicatie naar de nieuw gekozen technologieën. Hierbij is een technische discussie die ingaat op alternatieve frameworks, alternatieve API tech, verschillende Vue versies en andere mogelijke databases. Uiteindelijk is er gekozen voor Vue, REST API van Django en een PostgreSQL database. Verder is er gekeken naar de impact van de implementatie. Er is gereflecteerd of een dergelijke revisie echt nodig is. Ook is er gekeken naar de voor- en nadelen van een dergelijke implementatie. Hierbij is specifiek gekeken naar: prestatie-efficientie, uitwisselbaarheid, betrouwbaarheid, beveiligbaarheid, onderhoudbaarheid, risico's en kosten. Verder zijn er business opties gepresenteerd, waarbij gekeken is naar de mogelijkheden van mate van implementatie. Het advies is om de gehele revisie te implementeren en daarbij ook te focussen op herstructurering van documentstructuur van de sourcecode, herstructurering van applicatie componenten naar SFC structuur en implementatie van bundler tools en andere nuttige bijkomstigheden.

4 De geplande methode

Om dit project gestructureerd te laten verlopen is er een plan opgesteld om in vier fases te werken.

De eerste fase bestaat uit het oriënteren naar de huidige stand van zaken binnen Vue, de daarvan afgeleide frameworks, optionele packages, onafhankelijke tooling, Django en het Django REST framework. De resultaten van de oriëntatie zijn verwerkt in het theoretisch kader.

De tweede fase bestaat uit het onderzoeken hoe de Mobina applicatie er uit ziet en wat er verbeterd zou kunnen worden. Er is met Mobina gesproken over de wensen van het prototype van de nieuwe applicatie. De non-functionele eigenschappen waar de nieuwe architectuur verbetering in moet brengen zijn bepaald aan de hand van ISO-25010 en kritieke prestatie-indicatoren.

Er bestaan naast de huidige software die gebruikt wordt door Mobina veel andere technologieën die gelijke services bieden op basis van een andere focus. De oriëntatie- en onderzoeksfases worden bewust gelimiteerd op technologieën die niet alleen relevant zijn voor webapplicatie ontwikkeling maar ook overeenkomsten hebben met de software die op dit moment gebruikt wordt.

De derde fase betreft het ontwikkelen van een prototype met de nieuw-ontworpen architectuur om te kijken of het resultaat wenselijk, haalbaar, rendabel en efficiënt is. Het proof of concept zal zich uiten in een klantenportaal. De gewenste eisen van het bedrijf zijn hier in verwerkt. Dit prototype kan gebruikt worden door Mobina om als voorbeeld aan te houden in de ontwikkeling van de huidige applicatie.

De laatste fase is een rapport opstellen waarin ik Mobina IT adviseer over de revisie van de applicatie naar de nieuwe gekozen technologieën. Hierin zullen de overwogen kosten en baten toegelicht worden en gespeculeerd worden over individuele veranderingen binnen de applicatie.

5 Theoretisch kader

5.1 Webapplicaties

Webapplicaties zijn programma's die gebouwd zijn op basis van de client/server-architectuur van een webbrowser. Dit wordt gedaan omdat een applicatie via de bestaande wegen van een webbrowser op een breed spectrum van apparaten beschikbaar kan zijn, zonder dat er een grote stijging is van ontwikkel overhead. Computers, smartphones en vele andere devices met internettoegang kunnen via een browser dezelfde applicatie serveren aan de gebruiker. In de praktijk gebruiken webapplicaties HTML om structuur aan een user interface te geven, CSS om styling toe te passen, en Javascript om logica en functionaliteit toe te voegen.

5.2 MPA versus SPA

Een webapplicatie bestaat meestal uit meerdere pagina's die individueel of gezamenlijk use-cases voldoen voor een gebruiker.
In een multi-page application (MPA) bezoekt een gebruiker vaak meerdere paginas van de applicatie, waarbij hij voor elke pagina een nieuw verzoek naar de server stuurt. De server ontvangt het verzoek, past de nodige logica toe, rendert de pagina en stuurt het als antwoord naar de gebruiker. Tegenwoordig hebben webapplicaties complexe functionaliteiten waarbij veel data overeenkomt tussen pagina's. Hierdoor laat de gebruiker, bij het navigeren door een MPA, vaak onnodig data opnieuw versturen en renderen door de server. Elke keer dat de server een verzoek ontvangt van een client, zijn er resources en tijd nodig om dat verzoek af te handelen. Dat zorgt voor een lange laadtijd voor de gebruiker terwijl een groot deel van de pagina in veel gevallen al beschikbaar was uit een vorig verzoek.
Een single-page application (SPA) lost dit op door alle content van de applicatie te sturen als één pakket in het eerste verzoek van de client naar de server. Wanneer de client de bundel binnen heeft wordt er een Javascript programma gestart bij de client. De SPA handelt vervolgens alle navigatie van de gebruiker af, in plaats van een verzoek te sturen en een gerenderd antwoord van de server te ontvangen. Dit staat bekend als local routing. De gebruiker blijft vanuit het perspectief van de browser op dezelfde pagina. Wanneer een gebruiker andere content wil zien, bijvoorbeeld door naar een ander deel van de applicatie te navigeren, worden er nieuwe interface onderdelen gerenderd door de applicatie in de browser van de client. De gebruiker merkt geen verschil in hoe er met de navigatie omgegaan wordt.
De bundel van de SPA die de gebruiker laadt, is niet geschikt voor dynamische data. Onder dynamische data wordt bedoeld: data die veranderlijk is per gebruiker, tijdstip, etc.
Voor het laden van dynamische data wordt er een aparte server opgezet, ook wel de back-end server. Deze server kan transacties met een database afhandelen en beschikbaar stellen door middel van een application programming interface (API). De SPA 'consumeert' de API van die server.

Django als MPA
Django is een voorbeeld van een server-side rendering (SSR) MPA web framework. Django apps renderen de interface van de gebruiker aan de kant van de server. Dit gebeurt door middel van de templates die door de ontwikkelaar gespecificeerd zijn. Data die daarvoor nodig is wordt in de URL parameters meegegeven. Een terugkoppeling van een Django app hoeft niet altijd een pagina te zijn, het kan bijvoorbeeld ook een confirmatie zijn van de logica die is uitgevoerd betreffende dynamische data.
Django apps routen met behulp van URLs, die worden afgehandeld door de URL configuratie van Django. De ontwikkelaar geeft met eigen code de onderverdeling van paden aan in een bestand wat door de configuratie van Django gebruikt wordt.

Django gebruikt het model-template-view (MTV) patroon voor interactie tussen interface en model.
Het model representeert de data, maar de precieze methode om het te manipuleren wordt geregeld door het framework. In het geval van Django is het model een Python abstractie van een losstaande applicatie database.
De templates in dit patroon zijn bestanden die gebruikt worden om de interface, die aan de gebruiker getoond wordt, te genereren.
De view is verantwoordelijk voor:

1. Het ontvangen van een request van de client.
2. Het toepassen van logica, interactie met data via het framework, of het gebruiken van een template om een response te bouwen.
3. Het versturen van een response naar de client.

In een voorbeeld van een webshop applicatie zou een Django implementatie de volgende routes kunnen hebben.

```
1)https://www.fictieveWinkel.com/register
2)https://www.fictieveWinkel.com/products
3)https://www.fictieveWinkel.com/account
```

Tabel 1. Voorbeelden applicaties routes.

Route 1 is een registratiepagina voor nieuwe gebruikers. De webbrowser stuurt een verzoek naar de server voor een formulier wat de gebruiker in kan vullen. Omdat dit formulier hetzelfde zal zijn voor iedereen die zich wil aanmelden voor de webshop, hoeft de applicatie alleen een template te gebruiken om een interface te renderen en terug te sturen naar de gebruiker. Op het 'inleveren' van het formulier verstuurt de client de data terug en wordt het afgehandeld door de server. De data wordt gecontroleerd en opgeslagen in de database.
Als de gebruiker de beschikbare producten wil inzien gaat deze naar route 2, voor een overzicht van het account gaat deze naar route 3. Als deze pagina's overeenkomstige informatie hebben, bijvoorbeeld de naam van de gebruiker, moet per verzoek aan de server deze informatie opnieuw uit de database gehaald worden en gerenderd worden.

In tegenstelling tot de MPA architectuur van Django is Vue een voorbeeld van een technologie die (onder andere) SPA architectuur mogelijk maakt.

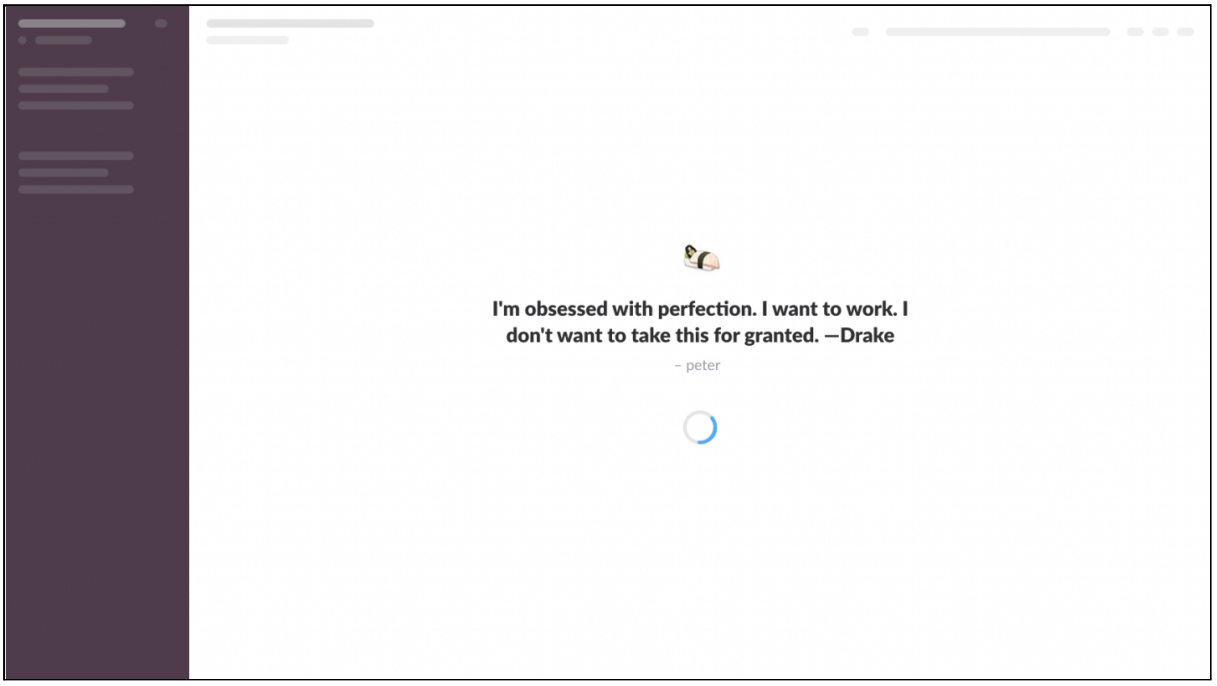
Vue als SPA
Vue is een voorbeeld van een front-end applicatie framework. Het rendert de interface van de gebruikers lokaal. Het staat bekend als een 'progressive framework', wat wil zeggen dat features van het framework iteratief geïmplementeerd kunnen worden in een applicatie.

Vue SPA's renderen de applicatie in de browser van de client. De templates worden tijdens deployment omgezet naar Javascript die door Vue weer vertaald kan worden naar HTML. Als er meer data nodig is wordt dit door asynchrone verzoeken opgevraagd bij de server.
Discrete onderdelen van de interface, die zich bezighouden met een op zichzelf staande functionaliteit, worden in Vue componenten genoemd. Een voorbeeld van een component is een navigatiebalk of een lijst van taken. Componenten hebben eigen templates en logica die door Vue gerenderd wordt.

Vue router beheert welke componenten gerenderd worden voor de gebruiker. Voor het renderen van die componenten kan de ontwikkelende partij de paden specificeren in een bestand. Het verschil tussen een 'normale' manier van router en de Vue router, is dat de Vue router de pagina dynamisch bij de client aanpast, in plaats van het doorsturen van de browser naar een nieuwe pagina. Omdat een bundel van alle Vue componenten voor een grote applicatie lange laadtijden kan opleveren, biedt de router een mogelijkheid de bundel te splitsen in chunks. De chunks zijn gegroepeerd op de route van de pagina van de componenten.

Vue gebruikt een model-view-viewmodel (MVVM) patroon. MVVM splitst een UI op in een view en model, net als de MVC, maar in plaats van een controller gebruikt deze een 'viewmodel' als buffer tussen de presentatie en de data. Er is geen interactie tussen de view en het model, alle logica wordt door de viewmodel georchestreerd. De viewmodel koppelt logica direct aan de UI elementen, in tegenstelling tot de separation of concerns die een controller creëert.
Interactie gebeurt als volgt:
1. De gebruiker communiceert met de view.
2. De veranderingen van de view worden gecontroleerd en toegepast door de viewmodel.
3. Het model signaleert een update naar de viewmodel met de vernieuwde data en/of een status.
4. Op basis van het bericht van het model wordt de view geupdate door de viewmodel.

In het eerder gegeven voorbeeld van de webshop applicatie zou een Vue implementatie als volgt geïmplementeerd worden.
Een gebruiker navigeert naar het domein van de applicatie (fictieveWinkel) en laadt alle componenten die de applicatie biedt. Wanneer de gebruiker zich wil registreren wordt de URL van de browser nog steeds aangepast naar 'https://www.fictieveWinkel.com/register' maar wordt er geen verzoek gedaan naar de server om deze pagina te renderen en serveren. De router van Vue neemt het verzoek af en rendert de 'pagina' (in Vue view genoemd) met de componenten die mee geladen werden in de bundel van de applicatie.
Als de gebruiker deze informatie wil inzien wordt er een verzoek gestuurd om de informatie van de back-end te ontvangen. Het verzoek wordt onafhankelijk van het laden van de pagina naar de server verstuurd. Wanneer de informatie is ontvangen kan dit lokaal worden opgeslagen of verwerkt.
Het komt vaak voor dat een component gerenderd is, voor de data ontvangen is. Voor deze gevallen zijn er animaties om de gebruiker er op te wijzen dat de interface nog niet klaar is.



Figuur 1. Een voorbeeld van een interface, waarbij de componenten al wel gerenderd worden, maar de data nog niet ontvangen is

6 Analyse van huidige applicatie

6.1 Non-functionele eigenschappen

Om een referentiekader te hebben voor de mankementen en verbeteringen van dit project, worden de applicatie-eigenschappen van ISO-25010 gebruikt. ISO 25010 is een standaard voor het beschrijven van de kwaliteit van een softwareproduct of -systeem.[1] Het model splitst 31 eigenschappen op in 8 categorieën die betrekking hebben tot technische kwaliteit en 11 eigenschappen over 5 categorieën betreffende gebruikskwaliteit. Als deze eigenschappen worden gecontextualiseerd naar de Mobina applicatie, kunnen er non-functionele kwalitatieve uitspraken over de applicatie gedaan worden.
Om dit te verduidelijken is er een lijst van vragen opgesteld die leidend zijn voor het bepalen van de individuele eigenschappen. Deze lijst is te vinden in de bijlagen.[2]

Technische eigenschappen		Gebruikseigenschappen
Functionele geschiktheid	**Betrouwbaarheid**	**Effectiviteit**
Functionele compleetheid	Volwassenheid	**Efficiëntie**
Functionele correctheid	Beschikbaarheid	**Voldoening**
Functionele toepasselijkheid	Foutbestendigheid	Bruikbaarheid
Prestatie-efficiëntie	Herstelbaarheid	Vertrouwen
Snelheid	**Beveiligbaarheid**	Plezier
Middelenbeslag	Vertrouwelijkheid	Welzijn
Capaciteit	Integriteit	**Vrijwaring tegen risico**
Uitwisselbaarheid	Onweerlegbaarheid	Economisch-risico-beperking
Beïnvloedbaarheid	Verantwoording	Gezondheids- en veiligheidsrisicobeperking
Koppelbaarheid	Authenticiteit	Omgevingsrisicobeperking
Bruikbaarheid	**Onderhoudbaarheid**	**Contextdekking**
Herkenbaarheid van geschiktheid	Modulariteit	Contextcompleetheid
Leerbaarheid	Herbruikbaarheid	Flexibiliteit
Bedienbaarheid	Analyseerbaarheid	
Voorkomen gebruikersfouten	Wijzigbaarheid	
Volmaaktheid	Testbaarheid	
gebruikersinteractie	**Overdraagbaarheid**	
Toegankelijkheid	Aanpasbaarheid	
	Installeerbaarheid	
	Vervangbaarheid	

Tabel 2. Verzameling van ISO-25010 non-functionele eigenschappen.

Als er gekeken wordt naar de twee hoofdcategorieën van de non-functionele eigenschappen kunnen er van tevoren al wat uitspraken gedaan worden. Zo is de lijst van gebruikseigenschappen niet bijzonder relevant voor dit project. Na overleg met Mobina is gebleken dat de belangrijkste categorieën prestatie-efficiëntie, uitwisselbaarheid, betrouwbaarheid, beveiligbaarheid, onderhoudbaarheid en overdraagbaarheid zijn.

6.1.1 Kritieke prestatie-indicatoren

Om te bepalen of er verbeteringen zijn behaald binnen niet-meetbare applicatie-eigenschappen tijdens de transformatie naar de nieuwe architectuur, zijn er kritieke prestatie-indicatoren (KPI) gedefinieerd. Deze KPI's moeten Mobina helpen om te achterhalen of ontwikkeling van de applicatie nog volgens de van tevoren bepaalde strategie loopt. KPI's dienen SMART te zijn. (Specifiek, Meetbaar, Acceptabel, Realistisch, Tijdgebonden)
Niet-meetbare eigenschappen zijn SMART te maken door middel van het stellen van een meetbaar doel. De volgende doelen zijn gesteld voor de nieuwe Mobina applicatie (in relatie tot de huidige applicatie):

[1] Applying the ISO/IEC 25010 Quality Models to Software Product: 25th European Conference, EuroSPI 2018, Bilbao, Spain, September 5-7, 2018, Proceedings
[2] Zie bijlage "Lijst ter verduidelijking ISO-25010"

Prestatie-efficiëntie:
- Een gebruiker moet sneller de gewenste content kunnen bereiken.
- Het apparaat van een gebruiker moet minder bandbreedte nodig hebben om dezelfde acties te kunnen uitvoeren.

Uitwisselbaarheid:
- De applicatie moet op hetzelfde platform aangeboden kunnen worden.
- De applicatie moet op dezelfde wijze content kunnen uitwisselen met andere Mobina software.

Betrouwbaarheid:
- De applicatie moet op dezelfde of minder krachtige apparatuur te gebruiken zijn.
- De applicatie moet ten alle tijden beschikbaar zijn om te downloaden.

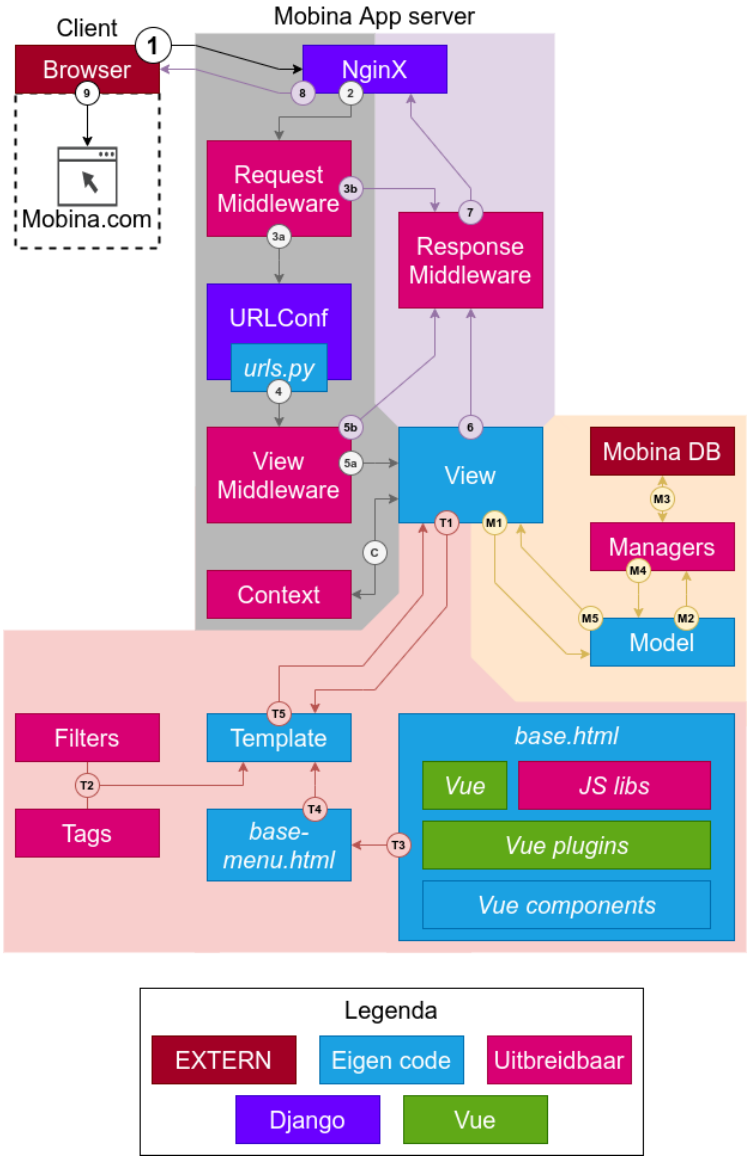
Beveiligbaarheid:
- De applicatie moet gebruikers op front-end en back-end kunnen laten verantwoorden op gebied van autorisatie en authenticatie.

Onderhoudbaarheid:
- De componenten van de applicatie moeten uit te wisselen zijn zonder dat de werking van de applicatie daaronder lijdt.
- De hoeveelheid componenten die te hergebruiken zijn moet groter zijn.
- De hoeveelheid componenten van de applicatie moet kleiner zijn
- De hoeveelheid documenten waaruit de applicatie bestaat moet kleiner zijn.

Overdraagbaarheid:
- De duur van het inwerken van een nieuwe ontwikkelaar en het toelichten tussen ontwikkelaars moet omlaag.

6.2 Applicatiestructuur

Om uitspraken te kunnen maken over de toegevoegde waarde van een andere applicatie-architectuur is er een uitgebreide analyse uitgevoerd op de bestaande applicatie. Zoals eerder vermeld is de bestaande webapplicatie een Django applicatie met Vue componenten. De applicatie verwerkt de verzoeken van de gebruiker op de applicatieserver en stuurt een gerenderd antwoord terug naar de client. Om de front-end functionaliteiten van de applicatie mogelijk te maken worden alle Vue componenten met elk gerenderd antwoord van de server verstuurd naar de client.



Figuur 2. De huidige applicatie-architectuur van Mobina.

De Vue componenten, Vue plugins en andere Javascript libraries die nodig zijn voor het correct functioneren van de applicatie worden gebundeld in één Django template bestand (base.html, T5). Dit bestand wordt geïnjecteerd in elk gerenderd antwoord van de server. Om een concreet voorbeeld te geven:
Als een gebruiker naar een onderdeel van de applicatie navigeert waar alleen component 1 tot en met 3 nodig is, wordt in het antwoord van de server het volgende geïnjecteerd:

- Een kopie van Vue applicatiecode + plugins.
- Een kopie van Vuetify applicatiecode + plugins.
- Kopieën van Javascript libraries die in de Mobina applicatie gebruikt worden.
- Componenten 1 tot en met 146.

Het injecteren van dit bestand maakt de huidige applicatie onnodig inefficiënt.

6.3 Vue gebruik van de huidige applicatie

6.3.1 Componenten analyse

De Mobina applicatie maakt op dit moment gebruik van 146 zelf-ontwikkelde Vue componenten. Deze componenten hebben voornamelijk hun functie in het renderen van front-end functionaliteiten die niet mogelijk waren met de templates van Django. Er zijn ook componenten die te maken hebben met het toevoegen van functionaliteiten die niet gerenderd worden.
Er zijn twee zelf-ontwikkelde Vuetify componenten in gebruik. Wat wil zeggen, componenten die erven van standaard Vuetify componenten met aangepast gedrag. Deze componenten zijn nodig om functionaliteit te bieden die niet standaard wordt aangeboden door de Vuetify componenten waar ze van erven.
De componenten worden gescheiden op de functionele categorie binnen de applicatie waar ze relevant voor zijn. Dit is gelijk aan de architectuur van de back-end, waar de functionaliteiten ook op de business cases worden gescheiden.
Een aantal componenten worden geïnitialiseerd als variabelen in plaats van de standaard definitie die gerenderd wordt door de Vue applicatie. Dit wordt gedaan om templating van deze componenten

./mobina/main/static/js/	146
├── vue-components.js	23
├── vuetify-extensions.js	2
├── **administration/**	1
├── **authentication/**	2
├── **blueprint/**	2
├── **collaboration/**	17
├── **critical_aspects/**	0*
├── **forms/**	2
├── **generic/**	38
├── **report/**	6
├── **requirements/**	9
└── **transition_agenda/**	32

Tabel 3. Verdeling van custom Vue componenten (rechts) over functionaliteiten van de applicatie.
** Componenten met betrekking tot critical aspects zijn op dit moment niet in gebruik.*

6.3.2 Dependency analyse

De applicatie is op dit moment afhankelijk van een aantal libraries en plugins die extra functionaliteiten bieden. Van deze uitbreidingen zijn vooral de front-end libraries van belang, omdat deze direct gedownload en geladen moeten worden door de client. Mobina heeft de volgende front-end dependencies:

- *Axios,* een HTTP client.
- *Chart,* een library die het mogelijk maakt om grafieken vorm te geven.
- *FindAndReplaceDOMText*, een library die de DOM van een html document kan aanpassen met reguliere expressies.
- *Jquery*, een populaire Javascript library die DOM manipulatie, events en HTTP calls kan afhandelen.
- *Moment*, een library voor het gebruik van date/time in Javascript.
- *Polyfill*, een library voor het toepassen van moderne HTML en CSS documenten in oude browsers.
- *Quill,* een library die het mogelijk maakt voor gebruikers om een “what you see is what you get” text editor te presenteren.
- *Sortable,* een library die drag ‘n drop functionaliteit levert voor Javascript lijsten.

De front-end dependencies worden door Mobina gehost. Dit betekent dat de sourcecode van de libraries door Mobina zelf gedownload is en meegestuurd wordt met de renders van de server.
Mobina maakt gebruik van Babel en Polyfill, twee libraries die als doel hebben om moderne code in oude browsers te ondersteunen. Dit geeft aan dat er een belang is om dit in de nieuwe applicatie te behouden.

De back-end van Mobina maakt gebruik van pip voor het beheren van Python dependencies. De meeste dependencies die gebruikt worden zijn uitbreidingen voor Django. Dit vormt voor de nieuwe architectuur geen probleem, gezien deze ook gebruik maakt van Django.

7 Prototype

Om Mobina te helpen met een keuze maken over de toekomst van de applicatie is er gevraagd om een prototype te maken die een nieuwe architectuur presenteert. Het bedrijf heeft een aantal eisen gesteld aan deze applicatie.

7.1 Functionele eisen

Om duidelijk te maken wat dit prototype zou moeten kunnen bereiken, is er een set aan functionele eisen opgesteld.

Mobina heeft laten weten dat de te onderzoeken applicatie frameworks gebaseerd moeten zijn op Vue voor de front-end en op Django voor de back-end van de applicatie. Deze eis is gesteld omdat de kennis binnen het ontwikkelteam gebaseerd is op de huidige applicatie die deze frameworks implementeert.

8.1.1 Kennisportaal
De kennisportaal applicatie van Mobina is een digitale locatie waar gebruikers toegang hebben tot informatie die ontwikkeld is door de content creators van Mobina. Het ondersteunt twee soorten gebruikers: Users en administrators.
Users moeten toegang hebben tot de informatie waar ze voor geautoriseerd zijn. Als een user een onderdeel van de informatiebank voor later wil bewaren, kan deze een bookmark toevoegen. Deze bookmarks zijn op een centrale locatie voor de user in te zien.
Administrators hebben dezelfde functies, maar moeten de mogelijkheid hebben om nieuwe gebruikers aan te maken en autoriseren. Verder kunnen administrators voor alle gebruikers bookmarks beheren, wat wil zeggen: toevoegen en verwijderen.
De informatie die geserveerd moet worden aan de Mobina gebruikers, wordt gegenereerd op een externe server in beheer van Mobina. Het kennisportaal moet de informatie van die server kunnen ontvangen en opslaan in eigen database. Het gaat hier om bestaande modellen die Mobina graag wil hergebruiken in de kennisportaal applicatie om eventueel later dit te integreren in die nieuwe iteratie van de Mobina applicatie.

De kennisportaal applicatie die ontwikkeld is als proof of concept, heeft een structuur die als voorbeeld genomen kan worden voor een revisie van de Mobina applicatie. Het project bevat de front-end en back-end van het project die duidelijk gescheiden zijn.

7.1.1 Business modellen

Het prototype implementeert de modellen die al gebruikt worden door de applicaties van Mobina. Dit is gedaan om de huidige applicatie te gebruiken als een fundering voor de nieuwe applicatie. De modellen vanuit Mobina die relevant zijn voor de opdracht zijn als volgt:

Enterprise
Binnen de modelstructuur van Mobina worden de bedrijven van de gebruikers gebruikt voor het beheren van de versie van de content die Mobina levert. In de huidige Mobina applicatie stellen Enterprises deze bedrijven voor.

User / UserEnterprise
Het User model is een representatie van een gebruiker binnen het Mobina systeem. De UserEnterprise is een koppeling tussen een gebruiker en een enterprise. In deze koppeling wordt de variabele data van de gebruiker bijgehouden.

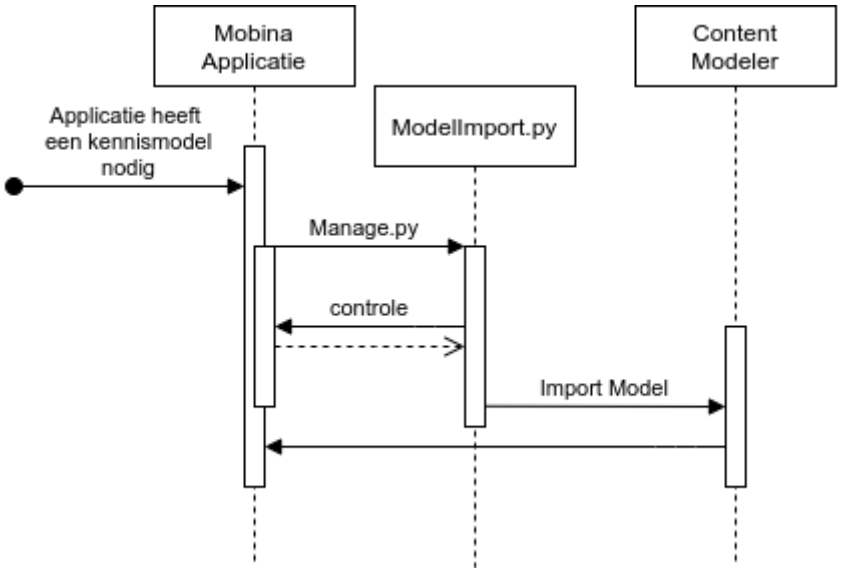
ModelElementEnterprise / CriticalAspectEnterprise / InnovationEnterprise
De content die geleverd wordt door Mobina is specifiek per bedrijf. De content die relevant is voor het kennisportaal kan worden opgesplitst in de volgende modellen: ModelElement, CriticalAspect en

Innovation. De inhoud van de modellen en wat ze leveren valt buiten de scope van de opdracht. De content moet geleverd worden aan een gebruiker zoals Mobina gespecificeerd heeft.

Bookmark
De gebruikers van het kennisportaal gebruiker een bookmark om elementen bij te houden die van belang zijn. Deze Bookmarks zijn simpele koppelingen tussen UserEnterprises en ModelElementEnterprises.

7.1.2 Content Modeler

De specificaties van de modellen zijn duidelijk gemaakt door Mobina. De invulling van deze modellen voor het ontwikkelen brengt een probleem met zich mee in de vorm van de content modeler. De content modeler is een door Mobina ontwikkelde applicatie om de kennis die Mobina over wil brengen met haar andere applicaties te genereren.



Figuur 3. De flow van de content model aanvraag binnen een Mobina applicatie.

Om de benodigde data te kunnen ontvangen van de externe content modeler moet het volgende worden meegegeven aan het ModelImport commando:

- Een actie om uit te voeren (info, install, update). ‘Install’ om een nieuw kennismodel te downloaden en te importeren.
- De ID van het Enterprise model om dit model voor te genereren. De enterprise dient te bestaan bij de client.
- Een geldig content versienummer van het te genereren kennismodel.
- Een geldig categorienummer. De categorie bepaalt voor welk vakgebied dit kennismodel gegenereerd wordt.
- Een “ModelAPI”. Deze API bepaalt naar welk endpoint van de content modeler het verzoek gestuurd wordt. De details van de API staan in de SETTINGS.py file van de applicatie.

Door de manier waarop de kennis met de modellen van Mobina (kennismodel) gegenereerd wordt in de content modeler, is er een probleem met de “enterprise structuur” die Mobina hiervoor aanhield. Om een kennismodel te genereren is er vanuit de applicatieserver, die het kennismodel aanvraagt, een geldige Enterprise nodig. Het is gewild voor het prototype om toegang te verschaffen voor alle gebruikers, ongeacht werkgever.
Er is overlegd met Mobina over wat de beste oplossing is voor de rol van de Enterprise binnen het prototype, waarna besloten is de Enterprise modellen aan te passen naar de categorie van die enterprises. Wat wil zeggen: De categorie van het kennismodel is een dummy enterprise om de bestaande structuur van database modellen te waarborgen.
Het hiervoor gebruikte script om kennismodellen te downloaden is aangepast om te werken met de nieuwe applicatie-architectuur.

7.2 Applicatiestructuur

Wat volgt is een beknopte samenvatting, voor meer details wordt gerefereerd naar de bijlagen.[3]

7.2.1 Front-end

De belangrijke aspecten van een Vue applicatie zijn de Views en Componenten, de Router en de Store.

Views en componenten

De verschillende views van de applicatie zijn zover mogelijk verdeeld volgens de back-end modellen. De views van de applicatie bestaan uit:

- **Home**, een landing page voor de gebruiker. Ongeautoriseerde of foute requests van de gebruiker worden naar de home view verwezen.
- **AdminView**, een view voor alle componenten die te maken hebben met de functionaliteiten van een administrator gebruiker. Deze view bevat de volgende componenten:
    - **UserList**, een lijst van gebruikers die toegang hebben op het kennisportaal.
    - **UserDetail**, een overzicht van een gebruiker met een lijst van diens bookmarks.
    - **UserCreate**, een formulier om een nieuwe gebruiker aan te maken in het systeem.
- **BookmarkView**, een view voor een overzicht van een gebruikers bookmarks. Het bevat de volgende component: **BookmarkList**, een lijst van bookmarks voor een specifieke gebruiker.
- **CriticalAspectView**, een view voor de componenten die te maken hebben met de CriticalAspects van de applicatie.
    - **CriticalAspectList,** een lijst van gewenste CriticalAspects.
    - **CriticalAspectDetail,** een gedetailleerd overzicht van een opgevraagde CriticalAspect.
- **InnovationView**, een view voor de componenten die te maken hebben met de Innovations van de applicatie.
    - **InnovationList,** een lijst van gewenste Innovations.
    - **InnovationDetail,** een overzicht van een individuele Innovatie.
- **LoginView,** een view voor de login component van de applicatie.
- **ModelElementView,** een view voor de componenten die te maken hebben met de ModelElements van de applicatie.
    - **ModelElementList,** een lijst van de gewenste ModelElements.
    - **ModelElementDetail,** een overzicht van een gewenste ModelElement.

De inhoud van de view is afhankelijk van de routing van de browser. Een view kan één of meer componenten renderen die te maken hebben met de functionaliteit van de view. Deze componenten kunnen deel uitmaken van dezelfde view of in elkaar genest zijn.

HTTP client

De front-end van de applicatie heeft een HTTP client nodig die asynchrone verzoeken kan verwerken voor de communicatie met de back-end. Mobina maakte al gebruik van Axios in de huidige applicatie, dus is er voor gekozen om dit te behouden in de nieuwe architectuur.
Axios kan en wordt in de nieuwe applicatie gebruikt als standaard HTTP client die door de Vue instantie aangeroepen kan worden.

Vuex store

De Vuex store wordt gebruikt om de gedeelde data binnen de applicatie te beheren. De data wordt bijgehouden in de state van de store. Hieronder valt:

- **Status**, een generieke status die de huidige stand van zaken binnen de applicatie beschikbaar maakt voor de componenten.
- **Token**, de login token gegenereerd door de server. Deze token wordt meegestuurd met alle verzoeken na de initiële login.
- **Authrole**, de autorisatie rol van de gebruiker. Kan ‘user’ of ‘admin’ zijn. Deze informatie wordt gebruikt om te bepalen wat de gebruiker wel of niet gerenderd mag krijgen in de applicatie.

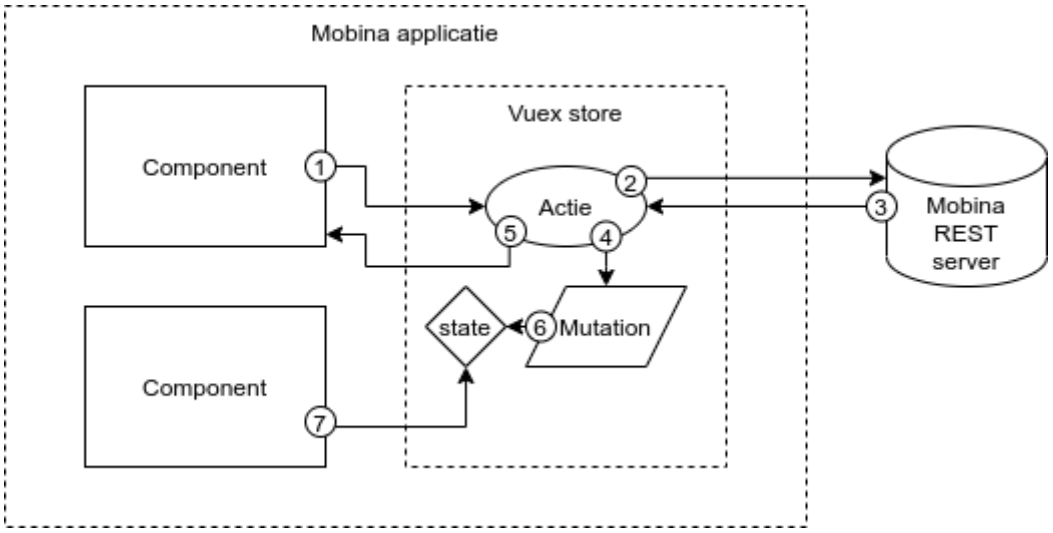
[3] Zie bijlage “back-end documentatie”

- **currentUserEnterprise**, de huidige UserEnterprise die de gebruiker op de applicatie gebruikt. Dit object is onder andere nodig om te bepalen wat de bookmarks van de gebruiker zijn.
- **UserEnterprises**, een verzameling van UserEnterprise ID's die horen bij deze gebruiker. Dit wordt gebruikt om de gebruiker snel van UserEnterprise te kunnen laten wisselen in de applicatie.

Om de verandering van state bij te houden in de back-end kan de store communiceren door middel van de volgende acties:

- **Login, Logout:** De acties die de inloggegevens van de gebruikers controleren.
- **CreateUser:** De actie voor de admins die het mogelijk maakt nieuwe gebruikers te maken.
- **Addbookmark, Removebookmark:** De acties die het de gebruikers mogelijk maakt om bookmarks toe te voegen of te verwijderen.

Er is niet één juiste methode voor het gebruik van een Vuex store. In de oriëntatie zijn er veel verschillende manieren gevonden om hetzelfde probleem op te lossen. In overleg met Mobina is besloten dat de conventionele methode voor de nieuwe applicatie als volgt werkt:



Figuur 4. Het gebruik van de Vuex store binnen de Mobina applicatie.

Een component maakt een verzoek om een actie van de Vuex store te gebruiken (1). De actie wordt gevalideerd en maakt een verzoek naar de REST server van de applicatie (2).
Waar veel methoden van elkaar afwijken is in de data die uitgewisseld wordt met de REST server. Initieel was het volgende bedacht:

1. De REST server valideert het verzoek en voert de actie in het verzoek uit.
2. Waar mogelijk wordt met het antwoord van de server een object meegestuurd wat gecreëerd of bewerkt is door de server. In de gevallen waar dit niet nodig is, wordt alleen een status teruggestuurd.

Deze methode werd toegepast om maximale zekerheid te bieden in de gelijkheid van data die in de front-end van de applicatie werd weergegeven en de data die op de server was bewerkt. Dit bleek niet een onderhoudbare structuur met de komst van de functionaliteiten van de administrators.
Daarom is er voor gekozen om met elk verzoek enkel data te versturen als deze is opgevraagd en in andere gevallen een statuscode te sturen (3).
Als de state van de applicatie aangepast moet worden wordt dit door de desbetreffende mutatie uitgevoerd (4, 6). Omdat de acties binnen de applicatie gebaseerd zijn op Javascript Promises is het mogelijk voor de actie om direct terug te koppelen naar de component die de actie heeft uitgevoerd (5). Dit is een bewuste keuze voor data die snel gerenderd moet worden en niet van belang is voor andere applicaties. In andere gevallen is de state beschikbaar (7).

Vue Router

De router van de applicatie heeft voornamelijk twee functionaliteiten.

- Het zorgt voor correcte routing tussen views.
- Door de navigation guards zorgt het voor de autorisatie en authenticatie van elke navigatie die plaatsvindt vanuit de gebruiker. Via de meta fields van de routes binnen de applicatie wordt gecontroleerd of de gebruiker ingelogd moet zijn of een bepaalde rol moet hebben om de view te mogen renderen.

7.2.2 Back-end

De back-end van de applicatie bestaat uit een
Deze apps bestaan uit de volgende onderdelen:

- API
- Serializers
- Viewsets
- Modellen

Content API

De content API Django app is verantwoordelijk voor alle interactie die een gebruiker kan hebben met de inhoudelijke informatie van de kennisportaal applicatie.

Auth API

De auth API Django app is verantwoordelijk voor alle interactie die een gebruiker heeft met betrekking tot autorisatie en authenticatie.

8 Discussie en aanbevelingen

8.1 Technische discussie

8.1.1 Alternatieve frameworks

Tijdens dit project is er onderzoek gedaan naar alternatieven voor de door het bedrijf gesuggereerde frameworks. Er waren meerdere alternatieven voor Vue, waarvan de meest belovende frameworks direct gebaseerd waren op Vue, namelijk Quasar en Nuxt. Deze frameworks implementeren een andere manier van het renderen van een applicatie. Andere frameworks die zijn onderzocht waren te jong of niet van toepassing voor dit project.
Na de gepresenteerde bevindingen werd er door Mobina besloten dat de focus van deze alternatieve frameworks geen prioriteit heeft en er niet voor wordt gekozen om deze frameworks te gebruiken voor de nieuwe applicatie.

8.1.2 Alternatieve API tech

Tijdens dit project is er onderzoek gedaan naar alternatieven voor de door het bedrijf gesuggereerde REST API functionaliteit van Django, namelijk OpenAPI, gRPC en GraphQL. Mobina heeft laten weten dat op het moment er geen prioriteit lag bij het veranderen van de specifieke API technologie.
Hoewel er andere invulling mogelijk is voor een Django back-end server is het advies inderdaad om een REST server te implementeren. De REST optie is het meest gebruikt door andere projecten, heeft de grootste userbase en is er is ervaring vanuit het team met REST oplossingen in Python.

8.1.3 Vue 3

In het begin van het onderzoek voor Mobina werd er interesse getoond in de nieuwste versie van Vue (Vue 3). In de bijlagen is te lezen wat Vue 3 een aantrekkelijke optie maakt.[4]
Vue 3 wordt niet aangeraden om te gebruiken op het moment van schrijven om de volgende redenen:

1. Er is nog geen of weinig ondersteuning voor Vue 3 in de Vue libraries die benodigd zijn om dezelfde functionaliteiten te bieden als de applicatie nu biedt. Vooral Vuetify is van belang voor de interface en is dus cruciaal voor het implementeren van een SPA versie van Mobina. De ontwikkelaars van Vuetify verwachten een 1.0 release aan het einde van het derde kwartaal van 2021[5].
   Het werd tijdens ontwikkeling van het prototype duidelijk dat veel projecten nog in de beginfase zaten van het implementeren van Vue 3 ondersteuning.
2. Volgens de implementatie van een RFC[6] (2 april 2021) van de officiële Vue 3 repository wordt ondersteuning voor Internet Explorer (IE11) niet voortgezet. Op dit moment ondersteunt Mobina IE11 en deze ondersteuning wordt voortgezet in de nieuwe applicatie-architectuur in het kader van complete coverage van de bestaande functionaliteiten.

Wanneer de Vue 3-relevante versies van technologieën stabiel zijn zou het voor Mobina interessant kunnen zijn om de overstap naar Vue 3 te maken. Desalniettemin is het advies om eerst een complete SPA te bouwen voor dat die overweging gemaakt wordt.

8.1.4 Database

Het is belangrijk voor Mobina om een PostgreSQL database aan te houden in verband met de koppeling met de content modeler. Bepaalde Django database-manager functionaliteiten zijn alleen geschikt voor PostgreSQL. Dit heeft als gevolg dat het inladen van de gedownloade modellen niet mogelijk is voor applicaties die niet beschikken over een Postgres database.

[4] Zie bijlage “Oriëntatie samenvatting”
[5] https://vuetifyjs.com/en/introduction/roadmap/#v3-0-titan
[6] https://github.com/vuejs/rfcs/discussions/296

Tijdens dit project is er onderzocht of SQLite een acceptabele vervanging van PostgreSQL zou kunnen zijn. Door de afwezigheid van de ondersteuning voor deze database-manager functionaliteiten is ondervonden dat dit niet een optie is.

8.2 Impact van implementatie

8.2.1 Business case

Om het project van Mobina voor de nieuwe applicatie goed te starten is er via de PRINCE2 methode een opzet gemaakt in de vorm van een Business case. Het doel van de business case is de rechtvaardiging van het project te presenteren door het volgende te presenteren:

- Waarom het project geopperd is.
- Wat de potentiële voordelen zijn.
- Wat de potentiële risico's zijn.
- De kosten van geld en tijd.
- De mogelijke business opties.

Waarom?

Mobina heeft baat bij een nieuwe architectuur. Uit vooronderzoek is gebleken dat de huidige gang van zaken binnen de applicatie niet efficiënt verloopt vergeleken met de mogelijke situatie van een nieuwe applicatie architectuur. Uit onderzoek is gebleken dat de beste oplossing die bij het bedrijf past een Vue SPA front-end applicatie is met een Django REST back-end.

Wat zijn de baten?

De volgende conclusies zijn getrokken op basis van de ISO-25010 normen die in de onderzoeksfase gesteld zijn. Deze bevindingen zijn met Mobina besproken.

Prestatie-efficiëntie
Hoewel render/transfer-tijd niet 1:1 te vergelijken valt met de applicatie zoals deze nu beschikbaar is, kan er nog steeds de conclusie worden getrokken dat de architectuur van het prototype efficiënter werkt. Deze conclusie kan worden gemaakt omdat het niet nodig is alle libraries/componenten die benodigd zijn voor de front-end te mee te sturen met elk server antwoord.
De rendertijd van de applicatie kan in sommige gevallen nog steeds traag zijn, dit heeft te maken met de data die wordt opgevraagd bij de REST back-end. Als de dynamische data de bulk van het verzoek is, zal het voor de gebruiker net zo traag zijn als de oude architectuur.
Omdat er minder werk nodig is om dynamische data te verwerken in de templates van Django, zal de nieuwe applicatie-architectuur beter scoren op het gebied van middelenbeslag.
De capaciteit eigenschap zal meer afhangen van de computer van de gebruiker. Omdat er meer render-werk gedaan moet worden en er meer logica bij de gebruiker uitgevoerd wordt. De lazy-loading functionaliteit van de router kan hier gedeeltelijk uitkomst in bieden, omdat deze functionaliteit de applicatie opgesplitst in chunks die just-in-time gerenderd worden bij de gebruiker. Dit zorgt voor minimale rendering van componenten die niet meteen nodig zijn.

Uitwisselbaarheid
Uitwisselbaarheid blijft van belang voor de Mobina applicatie, voornamelijk door de koppeling die nodig is met de content modeler. Omdat er gekozen is voor technologie die gelijk is, of erg verwant is aan de technologie die op dit moment gebruikt wordt, zal er een minimale impact zijn op de uitwisselbaarheid van de applicatie. Dit wil niet zeggen dat er geen hordes komen. Zoals tijdens ontwikkeling van het prototype is bevonden, kunnen subtiele verschillen grote impact hebben op de uiteindelijke invulling van de applicatie.

Betrouwbaarheid
De nieuwe ontwikkelomgeving (Vue CLI) geeft de mogelijkheid voor een fail-fast methode van ontwikkelen. Hierdoor zullen slordigheidsfoutjes sneller opvallen, wat zorgt voor een meer volwassen systeem.

Beveiligbaarheid
De autorisatie en authenticatie van gebruikers kan gewaarborgd worden met een duidelijk en robuust systeem. Zoals de applicatie nu werkt was er geen methode of behoefte om de autorisatie en authenticatie van gebruikers te controleren aan de front-end van de applicatie. Met de nieuwe methode worden gebruikers gecontroleerd aan beide kanten van de applicatie.
Het groeperen van gebruikers in administratieve groepen is een krachtige functionaliteit waar eerder minder of geen gebruik van werd gemaakt. Het is dan ook toegepast in de geplande methode van beveiliging.

Onderhoudbaarheid
De applicatie componenten zijn erg overzichtelijk in de SFC vorm. Gezien het plan om een hele applicatie te reviseren, is dit een uitgesproken moment om opnieuw te denken over de inrichting van de sourcecode van de applicatie. De nieuwe component vorm maakt het gemakkelijk in te zien:

1. Welke componenten gegeneraliseerd kunnen worden om met minder code een meer geconcentreerde applicatie te maken.
2. Welke componenten hergebruikt kunnen worden.

Ook is dit project een uitgelezen moment om een documentatie richtlijn op te zetten en aan te houden.

Wat zijn de risico's?

Zoals eerder vermeld is de keuze om overeenkomstige technologie te gebruiken gunstig voor het soepel verlopen van de overgang. Het is echter tijdens het ontwikkelen van het prototype een aantal keer voorgekomen dat er tegen een onverwacht obstakel werd gestruikeld. Deze obstakels zijn tot nu toe niet onoverkoombaar geweest, maar het is mede hierom extra van belang om de planningsfase voor het nieuwe project goed door te nemen.

Kosten en projectduur

Het is in dit stadium niet mogelijk een uitspraak te maken over de kosten van dit project. Het zal waarschijnlijk beter te bepalen zijn voor de opdrachtgever zelf met de bevindingen en adviezen uit het afgeronde onderzoek.
Hetzelfde geldt voor de precieze tijdsduur. In het implementatieplan[7] wordt een algemene planning gepresenteerd zonder expliciete duur van taken. Pas tijdens ontwikkeling kan bepaald worden wat realistisch is voor het gehele project.

Business opties

Er zijn drie mogelijke situaties:

1. Mobina voert het project niet uit. De situatie die nu frustreert zet door en de inspanning die gedaan moet worden om de applicatie te reviseren wordt naar voren geschoven.
2. Mobina voert alleen het hoognodige van dit project uit. Hieronder wordt verstaan:
    a. Geen herstructurering van de documentstructuur van de sourcecode.
    b. Geen herstructurering van applicatie componenten naar SFC structuur.
    c. Geen implementatie van bundler tools of andere nuttige bijkomstigheden.
    etc.
3. Mobina doet een normale investering in dit project. Buiten de minimale taken die technisch nodig zijn om de applicatie om te zetten naar de nieuwe architectuur, wordt ook geïnvesteerd in de bijkomende optionele features.

Gezien optie één niet een oplossing biedt, valt deze af. Optie twee is een minder dan wensbare uitkomst. Een groot deel van de frustratie die nu heerst bij Mobina is de onoverzichtelijkheid van de codebase. Deze minimale investering zou dit probleem niet oplossen. Er is daarbij geen onderzoek gedaan naar het omzetten van de Django templates naar iets anders dan single-file-components.Het advies voor Mobina is

[7] Zie hoofdstuk 8.2.3

dan ook dat er niet geschimpt wordt op de planningsfase van de implementatie van de nieuwe architectuur.

8.2.2 Projectproductbeschrijving

Om te concretiseren wat het doel en het product is van het SPA project voor Mobina is er een projectproductbeschrijving (Prince2) opgezet. Deze beschrijving legt vast wat de eisen zijn om als succes ervaren te worden. De projectproductomschrijving kan later bij de officiële opstart van het project verfijnd worden door de Mobina zelf. Tegen het einde van het project kan aan de hand van de beschrijving worden bepaald of de oplevering/lancering van de nieuwe applicatie voldoet aan de gestelde criteria.

Titel:

Mobina Vue SPA Transformatie Projectproductbescrijving

Doel:

Het verbeteren van performance en maintainability door het omzetten van de Mobina applicatie naar een Vue SPA architectuur met een Django REST Framework API back-end.

Producten:

- Een Vue SPA front-end applicatie
- Django REST back-end applicatie
- Documentatie

Benodigde competenties:

- Javascript ervaring, specifiek Vue applicatie structuur
- Python ervaring, specifiek Django/Django REST framework structuur

Kwaliteitsverwachting vanuit Mobina:

Het ontwikkelteam van Mobina is bekend met de competenties die nodig zijn voor dit project. Het is dus bekend bij Mobina wat er redelijkerwijs terecht gesteld kan worden. Mobina verwacht een applicatie die:

- Een overzichtelijke codebase heeft.
- Betere performance levert.
- Zich houdt aan standaard conventies.
- Meer front-end mogelijkheden biedt.

De applicatie dient duidelijke verbeteringen te tonen in de volgende aspecten: prestatie-efficiëntie, uitwisselbaarheid, betrouwbaarheid, beveiligbaarheid, onderhoudbaarheid en overdraagbaarheid.

Acceptatiecriteria:

De vervaardigde applicatie moet dezelfde (of meer) functionaliteiten bieden dan de applicatie die op dit moment ontwikkeld is. Het gaat hier om:

- Front-end single file components die de functionaliteiten van de huidige applicatie ondersteunen.
- Back-end Django applicatie die de functionaliteit heeft om de front-end applicatie te kunnen voorzien van de data die deze nodig heeft.

De functionaliteiten kunnen op dezelfde manier gecategoriseerd worden als in de front-end gebeurt:

- Administratief
- Authenticatie
- Blueprint modellen
- Collaboration
- Critical aspects
- Formulieren
- Verslaglegging
- Transition agenda
- Algemeen

Projectkwaliteitstoleranties:

Omdat dit een in-house project is van Mobina zal er niet anders gehandeld worden naar de marge van tolerantie wat betreft code, functionaliteit, of design.

Acceptatiecontrole:

Om te controleren of er voldaan is aan de eisen van een omgeschreven functionaliteit zal er een visuele controle plaatsvinden. Dit is hoe Mobina op het moment van schrijven test op

functionaliteit. Om de code van een onderdeel van het project te controleren wordt er een code review geörganiseerd. De commentaren van die code reviews worden verwerkt, waarna een merge-request goedgekeurd wordt.

Verantwoordelijke acceptatie:

De verantwoordelijke voor de acceptatie van de onderdelen van de applicatie zal een andere developer binnen het team zijn die ervaring heeft met de applicatie.

8.2.3 Stappenplan implementeren

Om Mobina te helpen met een roadmap voor dit project is er een stappenplan gemaakt op basis van bevindingen die tijdens de ontwikkeling van dit project gemaakt zijn. In de bijlagen wordt dit stappenplan gedetailleerder besproken.[8] In een beknopte omschrijving is het advies als volgt:

1. Het ontwerpen van de API's van de back-end.
2. Opsplitsen op use-case van modellen in de back-end.
3. Nieuwe URL tree bepalen. Er kan veel tijd en custom implementaties bespaard worden met de viewsets van het Django REST framework.
4. Views ontwerpen voor de front-end/router indelen
5. Componenten omzetten naar sfc's
6. Besluiten wat er bij de client lokaal moet blijven en wat er in een Vuex store opgeslagen moet blijven.

De verandering van architectuur kan side-by-side of in hetzelfde project als de oude applicatie worden uitgevoerd. Door de verandering van een Django app naar een Vue app moeten hoe dan ook twee servers gerund worden om de front-end en back-end werkend te krijgen.
Ik raad Mobina dan ook aan om verder te bouwen op het prototype wat nu klaar staat.
Ook raad ik Mobina aan om een onderzoek uit te voeren betreffende het gebruik van IE11 bij hun klanten. Ik denk dat de baten van Vue 3 groter zijn dan de userbase die gebruik maakt van IE11.

8.3 Overdracht

De overdracht van het project heeft later in het project een grotere rol gekregen. Omdat de ontwikkelaars onafhankelijk dit project verder met het bestaande prototype om moeten gaan, is er voor gekozen om een grotere focus te leggen op gemak van overdracht. In plaats van een los adviesdocument is er gekozen om een implementatieplan toe te voegen aan de documentatie van de code.
De documentatie tooling die wordt gebruikt voor dit project bestaat uit een Vue versie van Styleguideist voor de front-end en Sphinx voor de back-end.

[8] Zie bijlage "Implementatieplan Mobina SPA architectuur"

9 Reflectie

9.1 Project Management

Het plan was om dit project in tandem uit te voeren met de andere ontwikkelaars binnen Mobina. Helaas bleek dit door een aantal factoren niet erg haalbaar. Door de Corona-lockdown werd er voor het grootste gedeelte van het project in isolatie gewerkt, wat als gevolg had dat communicatie met directe begeleiders en de rest van het ontwikkelteam stroef verliep.
Een ander struikelblok was het niet kunnen werken op een agile-scrum methode. Gezien de oriëntatie en onderzoekende fases het langst duurde en deze werkzaamheden lastig in een agile werkmethode passen, kwam er helaas niet veel van de originele planning terecht. Ik heb dan ook geen eindresultaten die normaliter na een soortgelijk project besproken kunnen worden.

9.2 Leermomenten

Tijdens dit project heb ik meerdere malen gemerkt dat ik mijn eigen geplande koers niet kon aanhouden, omdat ik me te veel liet beïnvloeden door goedbedoelde begeleiding. De situatie was onhandig omdat de belangen van Saxion en Mobina vaak haaks op elkaar stonden. Omdat ik beide partijen tevreden probeerde te houden, heb ik helaas minder prioriteit in mijn eigen werk kunnen herkennen. Dit zorgde er voor dat er, achteraf gezien, veel werk is uitgevoerd waarvan de resultaten minder relevant waren voor het eindproduct en producten die wel belangrijk waren niet afgerond konden worden.

9.3 HBO-I competenties

Tijdens dit project heb ik mijn best gedaan om mij te verdiepen in de achtergrond van alle relevante onderwerpen om er zeker van te zijn dat ik mijn bevindingen kon onderbouwen.
Ik heb mijn best gedaan om Mobina te betrekken bij mijn bevindingen en keuzes. Het advies wat ik geef heb ik zo duidelijk mogelijk geprobeerd te maken.
Ik hoop dat Mobina nut haalt uit mijn afstudeerproject.

10 Bijlagen

10.1 Figuurlijst

10.2 Tabellijst

10.3 Lijst ter verduidelijking ISO-25010

10.3.1 Technische eigenschappen

Functionele geschiktheid: Voldoet de applicatie aan de behoeften van de gebruiker?
- ***Func. compleetheid***: Zijn alle gespecificeerde features ondersteund?
- ***Func. correctheid***: Zijn alle feature-resultaten juist en nauwkeurig?
- ***Func. toepasselijkheid***: Behalen de features de gestelde doelen?

Prestatie-efficiëntie: Werkt de applicatie met gewenst verbruik?
- ***Snelheid***: Zijn de verwerkingstijden en antwoordtijden van de applicatie volgens wensen?
- ***Middelenbeslag***: Verbruikt de applicatie een gewenste hoeveelheid middelen (kloktijd, RAM)?
- ***Capaciteit***: Voldoen de functionaliteiten aan hun grenzen?

Uitwisselbaarheid: Kan de applicatie informatie uitwisselen met andere software zonder problemen?
- ***Beïnvloedbaarheid***: Kan de applicatie functioneren zonder nadelig te zijn voor een ander product?
- ***Koppelbaarheid***: kan de applicatie informatie uitwisselen en deze gebruiken?

Bruikbaarheid: Kan de applicatie naar wensen gebruikt worden door klanten?
- ***Herkenbaarheid van geschiktheid***: Kan een gebruiker herkennen of de applicatie geschikt is voor hun behoeften?
- ***Leerbaarheid***: Kan een gebruiker redelijkerwijs leren hoe de applicatie gebruikt moet worden?
- ***Bedienbaarheid***: Kan een gebruiker met zo min mogelijk moeite de applicatie gebruiken?
- ***Voorkomen gebruikersfouten***: Is de gebruiker beschermd tegen het maken van fouten?
- ***Volmaaktheid gebruikersinteractie***: Is het voldoenend om de applicatie te gebruiken?
- ***Toegankelijkheid***: Is de applicatie door zo veel mogelijk mensen te gebruiken?

Betrouwbaarheid: Kan de applicatie functioneren onder verschillende omstandigheden?
- ***Volwassenheid***: Voldoet de applicatie aan betrouwbaarheid in normale omstandigheden?
- ***Beschikbaarheid***: Is de applicatie te gebruiken wanneer een klant hem wil gebruiken?
- ***Foutbestendigheid***: Werkt de applicatie ondanks aanwezige fouten?
- ***Herstelbaarheid***: Kan het de applicatie herstellen van een fout of onderbreking?

Beveiligbaarheid: Is informatie binnen de applicatie en in communicatie goed beschermd?
Vertrouwelijkheid: Kunnen alleen geautoriseerde personen bij gegevens?
Integriteit: Worden ongeautoriseerde acties verhinderd?
Onweerlegbaarheid: Is er bewijs van gebeurtenissen waar dat voor nodig is?
Verantwoording: Kunnen gebeurtenissen teruggeleid worden naar gebruikers?
Authenticiteit: Is de identiteit van gebruiker of onderwerp zoals beweerd?

Onderhoudbaarheid: Kan de applicatie gewijzigd worden door de beheerders?
Modulariteit: Zijn de onderdelen van de applicatie losstaand?
Herbruikbaarheid: Zijn applicatie onderdelen herbruikbaar?
Analyseerbaarheid: Kunnen er gemakkelijk inschattingen gemaakt worden over de applicatie?
Wijzigbaarheid: Kan de applicatie makkelijk gewijzigd worden?
Testbaarheid: Kunnen applicatiefuncties goed getest worden?

Overdraagbaarheid: Kan de applicatie worden overgezet naar andere platformen?
Aanpasbaarheid: Kan de applicatie aangepast worden voor andere platformen?
Installeerbaarheid: Hoe moeilijk is het een applicatie werkend te krijgen op of te verwijderen van een platform?
Vervangbaarheid: Hoe goed kan de applicatie een andere applicatie vervangen?

10.3.2 Gebruikseigenschappen

Effectiviteit: Hoe nauwkeurig en volledigheid kunnen gebruikers zijn in het bereiken van doelen?
Efficiëntie: Hoeveel hulp heeft een gebruiker nodig in het bereiken van doelen?

Voldoening: Hoe goed worden de behoeften van een gebruiker vervuld?
Bruikbaarheid: Is de gebruiker tevreden met de mogelijkheden van de applicatie?
Vertrouwen: Heeft de gebruiker er vertrouwen in dat de applicatie zich gedraagt als verwacht?
Plezier: Heeft de gebruiker plezier/geen ergernis van het gebruik van de applicatie?
Welzijn: Is de gebruiker tevreden over zijn welzijn?

Vrijwaring tegen risico: Hoe goed wordt er rekening gehouden met risico's?
Economisch-risico-beperking: Beperkt de applicatie economische risico's?
Gezondheids- en veiligheidsrisicobeperking: Beperkt de applicatie risico's met betrekking tot de gebruiker?
Omgevingsrisicobeperking: Beperkt de applicatie risico's met betrekking tot de context van de gebruiker?

Contextdekking: Hoe goed werkt de applicatie in verschillende omstandigheden?
Contextcompleetheid: Hoe goed werkt de applicatie in alle omstandigheden die zijn gespecificeerd?
Flexibiliteit: Hoe goed kan de applicatie werken in omstandigheden die niet gespecificeerd waren tijdens ontwikkeling?

10.4 Back-end documentatie + implementatieplan

Deze bijlage is bijgevoegd als apart bestand. In de documentatie staat ook het implementatieplan beschreven.

10.5 Samenvatting oriëntatiefase

Deze bijlage is bijgevoegd als apart bestand. In de documentatie staat ook het implementatieplan beschreven.