Academie Creatieve technologie

Studiejaar 2020-2021

Smart Mobility platform

Bachelor scriptie: Een onderzoek naar de beste manier hoe een systeem gemaakt kan worden inclusief (een deel van) de realisatie hiervan.

Auteur: Lloyd Waldt
Studentnummer: 438639
Opleiding: HBO-ICT – Software Engineering
Onderwijsinstelling: Hogeschool Saxion Enschede
Stagebegeleider: Ronald ten Berge
Stage coördinator: René van den Nieuwenhoff

Samenvatting

Elektrische auto's spelen steeds meer een rol in de automarkt. Vanwege deze toename is er vraag naar een systeem wat gebruikers op een mobiel platform kan adviseren of een elektrische auto bij hun past. Deze scriptie beschrijft het proces van het onderzoek en de realisatie van een prototype over dit probleem.

Het onderzoek is uitgevoerd door het toepassen van diverse literatuuronderzoeken voor het kiezen van technieken van bij het systeem horende applicaties. Het toepassen van veldonderzoeken voor specifieke kennis van het werkveld en een analyse van de stakeholders. Ook is er een werkplaatsonderzoek toegepast voor het iteratief ontwerpen van een bij het systeem horende databasestructuur. Om de uitkomst van de onderzoeken te valideren is er een Proof of Concept gerealiseerd waar de gekozen technieken getest worden op zowel individueel als samenwerkend vlak. De Proof of Concept heeft uiteindelijk gediend als basis voor het realiseren van een prototype voor het gevraagde systeem.

Uit het onderzoek is gebleken dat Flutter het meest geschikt is als front-end techniek, Java Spring Boot het beste als backend techniek en MySQL de best bijpassende databaseserver is. Bovendien is er uit het onderzoek gebleken welke data er nodig is voor het correct functioneren van het systeem, waarbij er ook een zo efficiënt mogelijke databasestructuur is ontworpen.

Als aanvulling op het onderzoek is er een prototype gerealiseerd waarbij gebruikers het brandstof verbruik van hun auto kunnen vergelijken met een andere auto. Deze vergelijking wordt gedaan op basis van een rit die geregistreerd kan worden doormiddel van het invullen van een formulier of het gebruiken van een gps-tracker. De autogegevens die worden gebruikt voor de vergelijkingen zijn afkomstig van openRDW, deze data wordt dagelijks gesynchroniseerd met de data van de eigen database. Alle geïmplementeerde functionaliteiten zijn getest met unit-, gui-, database- en/of exploratory testen.

Inhoudsopgave

Tabellen- en figurenlijst

1. Inleiding

Dit document bevat de scriptie over het onderzoek uitgevoerd door Lloyd Waldt, student en afstudeerder van Saxion. Het onderzoek is uitgevoerd onder leiding en toezicht van het bedrijf Sosialforce onder begeleiding van R. ten Berge. Sosialforce is gespecialiseerd in het ontwikkelen van applicaties met impact op business en de maatschappij. Het project is opgezet door adviesbureau Emodz. Emodz levert diensten en producten die bijdragen aan een versnelde transitie van conventionele mobiliteit naar duurzame mobiliteit. De aangewezen stage coördinator vanuit Saxion is R. van den Nieuwenhoff.

Het projectdoel is het vinden een manier om consumenten van auto's een advies te geven of een elektrische auto bij hun past. Hierdoor is er vraag gekomen naar een mobiel rit registratiesysteem met de mogelijkheid om ritten te simuleren met een bepaalde (elektrische) auto zodat er adviezen opgesteld worden voor consumenten.

Het systeem is complex en uniek door het gebruik van verschillende technieken om een zo accuraat mogelijk advies te geven. Een voorbeeld hiervan is het bijhouden van de afstand van een rit doormiddel van een gps-tracer. Een ander voorbeeld is het berekenen van de hoeveelheid tankbeurten op basis van een gekozen auto, waarvan de autogegevens van de auto bekend zijn door een externe koppeling met openRDW.

Er is onderzoek gedaan naar hoe dit systeem zo optimaal mogelijk ontwikkeld kan worden. Dit is gedaan door het uitvoeren van een onderzoek en het realiseren van een prototype. Er is hierbij onderzoek gedaan naar het antwoord op de hoofdvraag:

- Hoe kan er een systeem gerealiseerd worden wat gebruikers via een mobiele omgeving kan adviseren of een elektrische auto bij hen past?

Het onderzoek bestaat uit het vinden wat nodig is om het systeem te realiseren en welke aanvullende eisen er komen kijken voor de benodigde componenten.

In hoofdstuk 2 wordt er ingegaan op de probleemstelling omtrent het huidige systeem, tevens worden hier de hoofd- en deelvragen benoemd die beantwoord dienen te worden om het probleem op te lossen.

In hoofdstuk 3 is de requirementsanalyse te vinden. Hier staan de stakeholdersanalyse en de requirements die hieruit zijn voortgekomen.

In hoofdstuk 4 worden de gebruikte onderzoeksmethoden van de deelvragen besproken. Er wordt hier ingegaan op wat er gebruikt is, waarom en wat het opgeleverd heeft.

In hoofdstuk 5 zijn de resultaten met betrekking tot het onderzoek per deelvraag te vinden. Hier is alleen het eindresultaat te vinden, zie bijlage 2 voor de volledige uitwerking.

In hoofdstuk 6 wordt de conclusie op de hoofdvraag gevormd op basis van de resultaten van de verschillende onderzoeken van de deelvragen.

Hoofdstuk 7 bevat de discussie en aanbevelingen waarin wordt gereflecteerd op het proces en adviezen worden gegeven over vervolg van het project.

2. Probleemstelling

In het hoofdstuk *Probleemstelling* wordt het onderzochte probleem behandeld. Hierin zal een stuk achtergrondinformatie gegeven worden over het probleem wat opgelost dient te worden en het doel wat behaald is.

2.1 Achtergrondinformatie

Sosialforce is bezig met elektrische mobiliteit. Dit is een verzamelwoord voor 100% elektrische en plug-in hybride auto's. Er is momenteel een platform in ontwikkeling genaamd het *Smart Mobility Platform*, met dit platform kan kort door de bocht ondersteuning gegeven worden aan autodealers om een advies op te stellen of een elektrische auto geschikt is voor de consument. Door het invullen van diverse gegevens is het mogelijk om een advies op te stellen of elektrische mobiliteit bij de gebruiker past. Bij het huidige systeem komen een aantal problemen kijken, deze staan hieronder beschreven:

- Voor gebruikers zal het moeilijk in te schatten welke ritten ze per jaar rijden.
  → Momenteel dienen gebruikers een formulier in te vullen over het aantal afgelegde kilometers van een specifieke route en hoe vaak deze route per tijdseenheid wordt afgelegd. Het zal vaak voorkomen dat mensen niet weten hoe lang een bepaalde route is waardoor het formulier niet accuraat wordt ingevuld. Het gaat hier dus om de input van het formulier.
- De API is ingericht op de webapplicatie en heeft een vast model, dit kan problemen met zich meebrengen omdat data dus altijd in vaste vorm wordt opgevraagd.
- Autodata wordt momenteel in een Excel bestand geleverd en dient handmatig ingevoerd te worden.
- Het huidige systeem is oorspronkelijk opgezet voor één enkele klant maar moet geschikt worden gemaakt om als Software as a service ingezet te kunnen worden.

Om deze problemen op te lossen is er een project opgezet om een nieuw systeem te maken dat beter moet zijn ten opzichte van het huidige systeem. De problemen in het huidige systeem zullen in het nieuwe systeem opgelost worden. Ook zal het nieuwe systeem met Android en iOS samen werken.

Om deze opdracht te realiseren dienen in ieder geval de volgende componenten ontworpen en gemaakt te worden:

- Front-end
- Backend/ API
- Database (structuur)

2.2 Hoofd- en deelvragen

Om erachter te komen hoe en wat de beste manier(en) zijn om het systeem te realiseren, is er onderzoek gedaan waarbij er voor diverse vraagstukken een antwoord is gevonden. Met het antwoord op de deelvragen is een antwoord gevormd op de hoofdvraag. De hoofdvraag die wordt gesteld in de huidige context staat hieronder weergegeven.

Hoofdvraag

Hoe kan er een systeem gerealiseerd worden wat gebruikers via een mobiele omgeving kan adviseren of een elektrische auto bij hen past?

Om dit vraagstuk te beantwoorden, zijn er een aantal deelvragen gesteld die verschillende kanten van het probleem in beeld brengen om zo het hoofdprobleem op te lossen. De deelvragen met uitleg over het nut ervan om de hoofdvraag te beantwoorden staan in de hoofdstukken hieronder weergegeven.

Deelvraag 1

Wat is/zijn de beste techniek(en) om een iOS en Androidapplicatie te ontwikkelen die het mogelijk maakt om de gewenste functies te implementeren?

Met deze deelvraag wordt er vastgesteld welke techniek(en) in de huidige context geschikt is/zijn om de vraag naar een mobiele omgeving voor gebruikers waar te maken.

Deelvraag 2

Wat is/zijn de beste techniek(en) om een API te ontwikkelen die voldoet aan de eisen van de opdrachtgever.

Met deze deelvraag wordt er vastgesteld welke API-techniek(en) en structuur geschikt zijn voor het ontwikkelen van een API in de huidige context, te integreren is in het huidige systeem en flexibel kan communiceren met de front-end.

Deelvraag 3

Welke data gaat er opgeslagen worden in de database? En welk databasetype is geschikt?

Met deze deelvraag wordt er vastgesteld welke data er opgeslagen dient te worden zodat er bekend is welke data er nodig is om het systeem correct te laten functioneren. Ook wordt er op basis van deze data vastgesteld in welk databasetype de data het beste opgeslagen kan worden.

Deelvraag 4

Wat is de meeste efficiënte database structuur in dit scenario?

Met deze deelvraag wordt er vastgesteld wat de meest efficiënte databasestructuur in de huidige context is zodat de benodigde data voor het adviseren van gebruikers zo eenvoudig, snel en efficiënt mogelijk opgevraagd en opgeslagen kan worden.

Prototype
Als aanvulling op het theoretisch onderzoek zijn de onderzoeksresultaten gebruikt om een deel van het systeem te realiseren in de vorm van en prototype.

Met dit prototype is er gevalideerd dat de antwoorden op de deelvragen ook van toepassing zijn in de praktijk. Hiermee is er ook bewezen dat er een praktische oplossing voor de deelvraag is.

Hoofdvraag beantwoorden
Door onderzoek te doen naar de oplossing van elke deelvraag is er een antwoord gevormd op de hoofdvraag. Tevens is er met dit antwoord een oplossing gevormd over wat er nodig is om het systeem te realiseren. De aanpak van het onderzoek naar de oplossing van elk vraagstuk zal te vinden zijn in hoofdstuk 4. De resultaten van de uitvoering van het onderzoek zullen terug te vinden zijn in hoofdstuk 5.

2.3 Vormgeving van de front-end
Buiten de hoofd- en deelvragen is er ook tijd besteed aan het visuele deel van de front-end. Hierin is onderzocht en bepaald wat de prioriteit heeft voor het prototype op gebied van detaillevel van de lay-out en volgorde van schermen. Dit is terug te vinden in bijlage 9.

2.4 Implementatieonderzoek openRDW koppeling
Omdat er voor de openRDW koppeling erg veel data binnen wordt gehaald is er een beknopt onderzoek gehouden naar of dit goed afgehandeld kon worden met behulp van Big Data Solutions. Hier is uit gekomen dat Big Data Solutions niet geschikt is voor dit scenario omdat de huidige context niet past bij de scenario's dat Big Data Solutions ingezet dient te worden. De huidige context is niet geschikt omdat er gewerkt wordt met grote hoeveelheden gestructureerde data die opgehaald, opgeslagen en geüpdate wordt. Big Data Solutions is bedoeld voor het werken met grote hoeveelheden ongestructureerde data die continu geanalyseerd wordt om voorspellingen te kunnen maken. Zie bijlage 2, hoofdstuk 7 voor de volledige uitwerking van dit onderzoek.

3. Requirementsanalyse

Om requirements te kunnen bepalen is het eerst nodig om te bepalen wie de stakeholders zijn. In hoofdstuk 3.1 zal er een stakeholders analyse te vinden zijn. In hoofdstuk 3.2 zullen de hieruit naar boven gedreven requirements te vinden zijn.

3.1 Stakeholdersanalyse

De stakeholders van het product zijn de management van autodealers, verkopende autodealers, klanten van autodealers, Emodz, Sosialforce en de afstudeerder. Dit is bekend door een introducerende presentatie aan het begin van het project.

Alle stakeholders hebben invloed op het te realiseren product. De klanten van autodealers zullen in theorie de eindgebruiker van het product zijn. Omdat er nog weinig vraag is van autodealers naar het product, zullen zij weinig invloed hebben op het product. Wel hebben zij belang en kunnen dus gezien worden als geïnteresseerden. In dit geval zullen Emodz, Sosialforce en de afstudeerder de beïnvloeders zijn van het product.

3.1.1 Emodz

In de eerste week van het project is er kennis gemaakt met Sosialforce en samenwerkende partij Emodz. T Veldhuis (Medewerker van Emodz) heeft hier een presentatie gegeven over de werkzaamheden van Emodz en wat hun rol is binnen het project. Ook is er dieper ingegaan op het product over wat gerealiseerd dient te worden. Emodz is de partij die contact zoekt met de autodealers over het belang van het product.

3.1.2 Sosialforce

Sosialforce is de ontwikkelende partij, zij communiceren met Emodz over wat er dient te gebeuren en zorgen voor het ontwerpen en realiseren van een product op basis van de informatie die Emodz levert. Sosialforce ziet Emodz ook als opdrachtgever in de huidige use case.

3.1.3 Automarkt gerelateerde stakeholders

Om een vraag te creëren voor een product, is er eerst een aanbod gedaan aan autodealers. Hieruit bleek dat er belang is bij ondersteuning van verkoop voor elektrische auto’s. Autodealers willen verkopen, dit betekent dat zij tevreden zijn als het systeem correct functioneert. Zij zullen dus waarschijnlijk niet veel invloed hebben op welke specifieke functies het product moet bevatten, maar wel de waarde van het product. Als er namelijk geen belang is, is er ook geen waarde meer voor het product. Het is dan ook niet zo dat autodealers met wensen komen van het product die Emodz doorgeeft aan Sosialforce. Sosialforce en Emodz bedenken juist oplossingen en bieden dit aan bij autodealers om te valideren of zij hier de waarde van inzien.

3.1.4 Afstudeerder

De afstudeerder wordt ook gezien als stakeholder omdat deze persoon uiteindelijk het onderzoek zal uitvoeren. Dit betekent dat de afstudeerder geen invloed heeft op de eisen, maar wel op het uiteindelijke resultaat.

3.1.5 Prioriteitsvolgorde stakeholders

Op basis van de informatie die is verkregen uit de introductiepresentatie en de zojuist benoemde informatie van de stakeholders. Is er een prioriteitsvolgorde van stakeholders gemaakt die speelt rondom het project. Zie tabel 1 voor een visualisatie van de prioriteitsvolgorde van elke stakeholder. De prioriteit is op volgorde van hoog naar laag gezet, prioriteit 1 is hierin het hoogst.

Tabel 1: Prioriteitsvolgorde stakeholders

Prioriteit	Stakeholder	Beschrijving
1.	Hoofdkantoren autodealers	De hoofdkantoren van autodealers zijn uiteindelijk de hoofd stakeholders. Zij bepalen namelijk of de autodealers die 'onder' hun werken het systeem gaan gebruiken of niet. Hierdoor is deze partij ook van grote invloed op het eindresultaat van het product omdat het product voor hen van waarde moet zijn.
2.	Emodz	Emodz gaat in overleg met de hoofdkantoren van autodealers, zij zullen dus een zekere invloed hebben op wat de inhoud van het systeem gaat worden. Emodz wordt gezien als opdrachtgever vanuit Sosialforce
3.	Autodealers	Autodealers hebben een grotere invloed op het product, als zij het systeem uiteindelijk niet gebruiken zal het project alsnog 'gefaald' zijn.
4.	Sosialforce	Sosialforce is de ontwikkelende partij, zij hebben dus een zekere invloed op de inhoud van het systeem omdat daar uiteindelijk het systeem gerealiseerd wordt.
5.	Afstudeerder	De afstudeerder zal het project uitvoeren. Dit betekent dat de afstudeerder degelijke invloed zal hebben op het resultaat van het project. De afstudeerder heeft priority vier omdat het resultaat van het project gebaseerd zal zijn op de input van de stakeholders met hogere prioriteit.
6.	Klanten van autodealers	De klanten van autodealers hebben minimale invloed op het systeem. Er is namelijk nog geen belang van de klanten zelf. Wel zullen ze in de toekomst een grotere invloed krijgen omdat zij uiteraard de gebruiksvriendelijkheid van de mobiele app zullen bepalen.

3.1.6 Dataverzamelingsgesprekken

Er zijn een aantal gesprekken gevoerd met Emodz waarbij er gesproken werd over het nut van het systeem, de roadmap over hoe alles gerealiseerd wordt en wat alles dient in te houden. Uit deze gesprekken is de volgende informatie voortgekomen:

- Emodz is een bedrijf wat businessmodellen ontwerpt en implementeert voor importeurs en dealerholdings. Ze focussen zich vooral op de veranderingen binnen de automotive wereld.
- Momenteel heeft Emodz gesprekken met de hoofdkantoren van Hyundai, Volvo en Kia om informatie op te doen over een potentieel product. Hierbij is een product besproken waarbij op basis van een aantal parameters kan bepaald worden of een elektrische auto bij iemand hoort in de vorm van een vergelijking. In deze vergelijking moet een gebruiker zelf een auto kunnen kiezen, het liefst elke soort die er is op de markt.
- De informatie van elke auto op de markt ophalen zou mogelijk zijn door het ophalen van data van open RDW en AutoDisk, aangezien zij grote databases hebben met diverse informatie van elke geregistreerde auto in Nederland.
- In de toekomst is de gewenste situatie dat de gebruiker een auto uit de vergelijking kan kiezen, waarbij een meterkast bij de auto gekozen wordt. Op basis van de meterkast zouden er dan verschillende producten die compatibel zijn met de auto gekozen kunnen worden. Deze producten moeten geselecteerd en opgeslagen kunnen worden, om er vervolgens een overzicht ervan te kunnen downloaden. Ook dient er een offerte van de gekozen producten inclusief auto verkregen te kunnen worden.

Uit deze informatie kunnen verschillende functies vertaald worden. In een vervolggesprek met CEO O. de Rijter en CTO R. ten Berge van Sosialforce is de volgende informatie hieruit voortgekomen:

- Er dient een app gemaakt te worden die de klant kan meenemen naar huis om accuraat te registreren welke afstanden hij aflegt. Met deze afstand moet de gebruiker het verbruik van verschillende auto’s kunnen vergelijken.
- Deze app dient bereikbaar te zijn voor een zowel Android als iOS.
- De geregistreerde ritten moeten gebruikt kunnen worden in het vergelijkingsproces.

3.1.7 Interview

Om extra informatie over de stakeholders op te doen is er een veldonderzoek in de vorm van een interview uitgevoerd. In het interview zijn vragen gesteld waarmee kennis is opgedaan over extra eisen en voorkeuren voor het kiezen van technieken voor de realisatie van het prototype. Het interview is gehouden met CTO R. ten Berge. Onder elke vraag zal een onderbouwing van waarom de vraag is gesteld staan, gevolgd door het antwoord op de vraag.

Vraag 1

Welke technieken zijn er binnen Sosialforce al gebruikt voor het realiseren van native- en hybride mobiele applicaties?

Deze vraag wordt gesteld omdat het kennis oplevert over welke technieken er al wel en niet zijn gebruikt voor de realisatie van mobiele applicaties voor Android en iOS. Hiermee kan er ook rekening gehouden worden met de onderhoudbaarheid in de toekomst. Tevens dient het antwoord als basis voor de volgende vraag.

Antwoord

Binnen Sosialforce is gebruik gemaakt van React-native, Java, Kotlin, Swift en Objective-C voor het ontwikkelen van mobiele applicaties.

Vraag 2

Wordt er waarde gehecht aan het gebruiken van technieken die al eerder gebruikt zijn of wordt er meer voorkeur gelegd bij het gebruik van nieuwe technieken?

Met deze vraag kan er worden achterhaald of de voorkeur ligt bij al eerder gebruikte technieken of dat de voorkeur ligt bij het gebruiken van nieuwe technieken voor mobiele applicaties.

Antwoord

Dit ligt aan het scenario, stel dat Java beter geschikt zou zijn voor dit project, dan kies je hier uiteraard voor. Maar als volgens jou een voor Sosialforce onbekende techniek beter geschikt is, kies je hiervoor.

Conclusie

Met de antwoorden van het interview kan er geconcludeerd worden dat er voor de front-end geen eisen zijn voor het kiezen van een techniek voor de front-end.

3.2 Requirements

Op basis van de stakeholders analyse en de dataverzameling zijn er een aantal requirements opgesteld die aansluiten op de wensen van de stakeholders. Deze lijst is besproken met de stakeholders op basis van inhoudelijke betekenis van elke requirement. Bovendien is er gespard over de prioriteit, gebaseerd op wat belangrijk zou zijn voor een eerste versie van het prototype. Zie tabel 2 en 3 voor de opgestelde requirements.

Tabel 2: Requirements

Nr.	Requirement	Functionality	Priority
1.	De gebruiker moet kunnen inloggen.	Functional	M
2.	De gebruiker moet een overzicht van auto's kunnen krijgen op basis van externe gegevens (open RDW en AutoDisk).	Functional	M
3.	De gebruiker moet een auto uit een overzicht kunnen selecteren waarbij ook een foto van de auto getoond wordt.	Functional	M
4.	De gebruiker moet handmatig ritten kunnen registreren.	Functional	M
5.	De gebruiker moet via gps-gegevens ritten kunnen registreren.	Functional	M
6.	Het moet mogelijk zijn om een realtime overzicht te krijgen van gesimuleerde gegevens op basis van gps-gegevens.	Functional	M
6.1	Het moet mogelijk zijn om op basis van gps-gegevens de gesimuleerde actieradius te zien.	Functional	M
6.2	Het moet mogelijk zijn om de huidige snelheid te zien gebaseerd op gps-gegevens	Functional	M
6.3	Het moet mogelijk zijn om op basis van gps-gegevens het rijgedrag te bepalen.	Functional	M
6.4	Het moet mogelijk zijn om het rijgedrag te bepalen in het realtime overzicht.	Functional	M
7.	De gebruiker moet in een formulier de gegevens van minimaal 1 rit kunnen invullen. Welke auto, interval van de rit, afstand, wat voor soort rit (vakantie, vrije tijd, etc.)	Functional	S
8.	De gebruiker moet in het formulier de laadopties kunnen selecteren. (Ochtend, middag, dag, nacht)	Functional	S
9.	De gebruiker moet in het formulier de rijstijl kunnen selecteren. (Eco, normaal, sport)	Functional	S

Opmerking: Priority 'M' staat voor Must, 'S' staat voor Should.

Tabel 3: Requirements(vervolg)

Nr.	Requirement	Functionality	Priority
10.	De gebruiker moet een overzicht kunnen krijgen van het benodigde aantal tank- en oplaadbeurten op basis van de ingevoerde parameters in het formulier.	Functional	S
11.	De gebruiker moet de hoofdaansluiting (meterkast) kunnen selecteren. (Om de maximale laadsnelheid te kunnen bepalen)	Functional	S
12.	De gebruiker moet een mobiliteitsprofiel kunnen selecteren en aanmaken waar de rij auto, laadopties en rijgedrag in vastgesteld zijn.	Functional	M
13.	De gebruiker moet een overzicht kunnen ophalen van de optimale, maximale en op de toekomst voorbereide laadstations. (Afhankelijk van de hoofdaansluiting in de meterkast en de gekozen auto)	Functional	S
14.	De gebruiker moet visueel een laadstation kunnen selecteren op basis van resultaten van requirement 12.	Functional	S
15.	Het moet mogelijk zijn om minimaal meerdere product te selecteren. (Onder producten valt: schouwing van de hoofdaansluiting, laadpalen, laadpassen)	Functional	S
16.	Het moet mogelijk zijn om een overzicht te krijgen van geselecteerde producten op basis van requirement 14.	Functional	S
17.	Het moet mogelijk zijn om het resultaat te downloaden van de ingevoerde gegevens en berekeningen van de rit en geselecteerde producten.	Functional	S
18.	Het moet mogelijk zijn om een offerte op te vragen van de installatie van de geselecteerde laadpaal.	Functional	S
19.	De app moet minimaal iOS 11 ondersteunen	Non-functional	M
20.	De app moet minimaal Android 6 (Marshmallow) ondersteunen	Non-functional	M
21	De apps moeten voldoen aan de richtlijnen van Apple en Google (inclusief de laatste privacy eisen)	Non-functional	M

Opmerking: Priority ‘M’ staat voor Must, ‘S’ staat voor Should.

4. Methoden

In het hoofdstuk *Methoden* worden de verschillende aanpakmethoden die gebruikt zijn binnen het project besproken. In hoofdstuk 4.1 zal de gehanteerde projectmanagementmethode worden benoemd en besproken. In hoofdstuk 4.2 zullen de 3 fases van het project worden toegelicht. In hoofdstuk 4.3 zullen de gehanteerde aanpakmethodes per deelvraag worden behandeld.

4.1 Projectmanagementmethode

Voor het afstudeerproject gaat Kanban gebruikt worden als projectmanagement methode. Er is gekozen voor Kanban omdat hiermee het overzicht van de werkzaamheden centraal staat. Er wordt niet in sprints gewerkt maar in één flow per taak. Dit is handig omdat een onderdeel van het projectonderzoeken is. Om dit in bijvoorbeeld sprints of punten te verdelen zal erg lastig zijn omdat het niet bekend is hoelang een onderzoekstaak zal duren of wat de complexiteit hiervan is. Overigens bevat Kanban geen vaste meetings waarin je met je team reflecteert op het proces, dit is beter omdat dit project solo wordt uitgevoerd. Omdat het project solo wordt uitgevoerd kan er direct bijgestuurd en gereflecteerd worden wanneer nodig, de reflectie op het proces zal continu gebeuren. Overigens wordt er 4x in de week gesproken met collega's in de vorm van een stand-up om alsnog de nodige feedback en adviezen te krijgen tussen de werkzaamheden door.

4.2 Projectfases

Het project is onder te verdelen in 4 fases. De voorbereidende fase, de onderzoeksfase, de realisatiefase en de afrondingsfase. In de onderstaande hoofdstukken wordt de inhoud en het doel van elke fase toegelicht.

4.2.1 Voorbereidende fase

De voorbereidende fase was de eerste fase van het project. Tijdens deze fase is er een plan van aanpak gemaakt. Hierin staat beschreven hoe het project aangepakt gaat worden. Er is in deze fase gedacht aan de stappen die nodig waren om het project uit te voeren. Er is nagedacht over de aanleiding en het doel van de opdracht, de onderzoeksvragen die beantwoord dienden te worden, de methodische aanpak voor het beantwoorden van elke deelvraag, de onderzoeksopzet en de op te leveren beroepsproducten. Op basis van de uit te voeren stappen zoals deze in het plan van aanpak beschreven staan, is er een globale planning gemaakt voor de verschillende taken en fasen tijdens het project.

4.2.2 Onderzoeksfase

De onderzoeksfase of tweede fase waren gericht op beantwoorden van de hoofd- en deelvragen. Om dit te doen is er eerst een stakeholdersanalyse uitgevoerd om achter de requirements te komen. Na elk onderzoek dat hoorde bij een deelvraag is er een conclusie getrokken op deze deelvraag. Nadat alle deelvragen beantwoord waren is er een conclusie getrokken op de hoofdvraag.

4.2.3 Realisatiefase
De realisatiefase of derde fase was gericht op het realiseren van het prototype. De conclusies die getrokken zijn vormen het plaatje van ‘wat’ er nodig is om het product te realiseren. In de realisatiefase is het doel dat er bezig wordt gegaan met ‘hoe’ het product gemaakt wordt.

4.2.4 Afrondingsfase
De afrondingsfase is de laatste fase van het project. In deze fase zal de voortgang van het product worden afgesloten tot een fatsoenlijk op te leveren product. Ook is er hier gewerkt aan het afronden van de scriptie met bijbehorende bijlagen.

4.3 Onderzoeksmethoden
In dit hoofdstuk zal per deelvraag algemeen besproken worden welke onderzoeksmethoden zijn toegepast om informatie te verzamelen. Ook zal hierbij benoemd worden waarom er gekozen is voor de methode per specifieke deelvraag.

4.3.1 Deelvraag 1
Deelvraag 1 luidt: ‘Wat is/zijn de beste techniek(en) om een iOS en Androidapplicatie te ontwikkelen die het mogelijk maakt om de gewenste functies te implementeren?’. Om deze vraag te beantwoorden zijn de onderstaande onderzoeksmethoden gehanteerd.

Literatuuronderzoek
Met het literatuuronderzoek is er informatie verkregen over welke technieken geschikt zijn voor het ontwikkelen van een Android- en iOS applicatie. Voor het literatuuronderzoek is er op Google gezocht met de volgende zoekopdrachten:
- Apple app development
- Android app techniques
- How to make an Android and Apple app?

Van elke techniek die hieruit naar voren is gekomen, zijn de eigenschappen en voor- en nadelen met elkaar gedocumenteerd en vergeleken. Er is bij dit onderzoek veel informatie verkregen van de officiële developer websites van Apple, Android en het forum Stackoverflow.

Er is gekozen voor een literatuuronderzoek omdat er hierdoor veel informatie over veel verschillende technieken verkregen kon worden. Overigens was de keuze van de techniek gebaseerd op best passend bij het product, in plaats van een techniek waarvan het bedrijf al kennis heeft. Dit betekent dat er geen restricties aan het kiezen van de techniek zaten. Met dit in beeld zal een literatuuronderzoek de meeste resultaten opleveren.

De conclusie is getrokken op basis van de volgende criteria:
- Performance
- Ondersteuning (iOS en Android)
- Duidelijkheid (Is de techniek bekend/ is er een (grote) community en is er duidelijke documentatie)

4.3.2 Deelvraag 2

Deelvraag 2 luidt: 'Wat is/zijn de beste techniek(en) om een API te ontwikkelen die voldoet aan de eisen van opdrachtgever?'. Om deze vraag te beantwoorden zijn de onderstaande onderzoeksmethoden gehanteerd.

Veldonderzoek - observatie

Met het veldonderzoek is er informatie verkregen over de huidige situatie van het project. Er is hier informatie verkregen over wat de technieken en tools zijn die in het huidige systeem zijn, dit is gebruikt om te voldoen aan de eis dat het systeem te integreren moet zijn in met het huidige systeem.

Voor het veldonderzoek is er een observatie uitgevoerd op het huidige systeem onder leiding van CTO R. ten Berge. Uit het veldonderzoek is informatie verkregen over de gebruikte technieken van het huidige systeem en gebruikte API-structuur.

Er is gekozen voor een veldonderzoek vanwege de eis dat het systeem makkelijk te integreren moet zijn in het huidige systeem. De beste manier hiervoor is observeren hoe het huidige systeem werkt en in elkaar zit.

Literatuuronderzoek

De techniekkeuze dient aangevuld te worden met een structuur keuze. Met het literatuuronderzoek is er informatie verkregen over verschillende API-structuren die het huidige systeem flexibeler kunnen maken in gebruik en ontwikkelingen in de toekomst.

Het onderzoek naar een de verschillende API-architecturen is uitgevoerd door op Google de volgende zoekopdrachten uit te voeren:

1. API architectures
2. REST API alternatives

Op basis van de zoekresultaten zijn er een aantal API-architecturen onderzocht op basis van werking, ondersteuning, opbouw en scenario's waar ze geschikt voor zijn. Op basis hiervan zijn de voor- en nadelen te opzichte van de requirements in kaart gebracht. Op basis van die resultaten is er besloten welke API-architectuur geschikt is voor het huidige project.

Er is gekozen voor een literatuuronderzoek omdat er via deze methode veel vrijheid is in het vinden van verschillende mogelijkheden. Via een simpele Google zoekopdracht kom je al erg veel te weten over wat een geschikte API-structuur zou kunnen zijn.

De conclusie van deelvraag 2 is getrokken op basis van de volgende criteria:

- Managebility (integreerbaarheid)
- Flexibiliteit
- Ondersteuning van techniek op gebied van programmeertaal
- Overzichtelijkheid

4.3.3 Deelvraag 3

Deelvraag 3 luidt: 'Welke data gaat er opgeslagen worden in de database? En welk databasetype is geschikt?'. Om deze vraag te beantwoorden zijn de onderstaande onderzoeksmethoden gehanteerd.

Veldonderzoek - Interview (incl. observatie)

Met het interview is er informatie verkregen over welke data er daadwerkelijk opgeslagen dient te worden in de database. In combinatie met de observatie is er ook een beeld gevormd van welke data bij welk dataobject (kan) horen en waarom dit zo is. Ook biedt dit ondersteuning aan verschillende ideeën hoe de huidige omgeving verbeterd zou kunnen worden. Overigens heeft het interview kennis geleverd over de database server die gebruikt wordt in de huidige omgeving.

Er is gekozen voor een interview omdat er op deze manier specifieke vragen gesteld kunnen worden aan een kennis dragend persoon met betrekking tot het project. Voordat het interview gestart was, was er al een eerste versie van de data die opgeslagen werd gemaakt, met dit interview kon deze eerste versie ook gevalideerd worden tijdens het interview.

Literatuuronderzoek

Nadat de data die opgeslagen moet worden bekend werd, diende er nog een bijpassende database server gekozen te worden. Met het literatuuronderzoek is er bekend geworden welke relationele database bij het huidige scenario past.

Er is gekozen voor een literatuuronderzoek omdat er hiermee veel mogelijkheden naar voren komen. Hiermee kunnen veel mogelijkheden vergeleken worden met elkaar waardoor er afgewogen is wat de meest geschikte techniek is voor het huidige scenario.

4.3.4 Deelvraag 4

Deelvraag 4 luidt: 'Wat is de meest efficiënte databasestructuur in dit scenario?'. Om deze onderzoeksvraag te beantwoorden zijn de onderstaande onderzoeksmethoden gehanteerd.

Literatuuronderzoek

Met het literatuuronderzoek is er kennis opgedaan over wat er bedoeld wordt met een efficiënte database structuur. Ook is er kennis opgedaan over hoe een efficiënte databasestructuur ontwerpt kan worden.

Voor dit onderzoek is er een Googleonderzoek uitgevoerd waarbij de volgende zoekopdrachten zijn gebruikt:

- Efficiënte databasestructuur maken
- Wanneer is een databasestructuur efficiënt?
- Hoe maak je een database efficiënt

Omdat er specifieke regels en principes zijn voor een efficiënte databasestructuur, is het met een literatuuronderzoek het meest eenvoudig en snel om erachter te komen wat deze zijn. Om deze reden is ervoor gekozen om een literatuuronderzoek uit te voeren.

Veldonderzoek - observatie

Met het veldonderzoek is er informatie verkregen over de databasestructuur van het huidige systeem. Belangrijke informatie die hier gevonden is gaat over het gebruik van koppeltabellen en de relaties tussen tabellen in het huidige systeem.

Er is gekozen voor een veldonderzoek omdat dingen uit het huidige systeem te hergebruiken zijn in het nieuwe systeem, maar ook om dingen juist niet mee te nemen in het nieuwe systeem zodat het beter wordt dan het huidige systeem.

Werkplaatsonderzoek - prototyping

Met het werkplaatsonderzoek zijn er meerdere versies van een databasestructuur ontworpen, waarbij doormiddel van feedback telkens een betere versie van de databasestructuur gemaakt kon worden.

Er is gekozen voor een werkplaatsonderzoek omdat er hiermee doormiddel van trial en error steeds een betere versie van de databasestructuur naar voren komt, tot het punt dat er 'geen' betere versie meer is. (Uiteraard zal er in alle gevallen plek zijn voor verbeteringen)

4.3.5 Proof of concept

Als aanvulling op deelvragen 1, 2 en 3 is er een Proof of concept uitgevoerd om de resultaten van deze drie deelvragen te valideren in de vorm van een praktijkonderzoek.

Er is gekozen om een Proof of concept uit te voeren omdat de theorie vaak anders blijkt te werken dan de praktijk. Met behulp van een Proof of concept is er gevalideerd of de verschillende theorie onderzoeken naar de front-end, backend en database ook valide zijn in de praktijk.

4.3.6 Realisatie

Als aanvulling op het theoretisch onderzoek is een deel van het systeem gerealiseerd. Voor de realisatie is een technisch- en functioneel ontwerp gemaakt, een prototype van het systeem en er zijn diverse testen geschreven en uitgevoerd om de functionaliteit van het prototype te valideren. In de onderstaande hoofdstukken staan de opgeleverde producten beschreven, als aanvulling hierop zal de aanpak van de realisatie worden beschreven.

Technisch ontwerp

Het technisch ontwerp bevat het de documentatie voor de technische kant van het systeem. Hierin wordt ingegaan op de opbouw en indeling van de front-end, backend en database. Het ontwerp is gebaseerd op de onderzoeksresultaten van de deelvragen. Er is een technisch ontwerp gerealiseerd omdat dit een essentieel onderdeel is van het systeem. Het biedt overzicht op de technische aspecten van het systeem op gebied van gebruikte technieken, structuur en belangrijke functies van individuele componenten. Bovendien wordt duidelijk hoe verschillende componenten met elkaar samenwerken.

Functioneel ontwerp
Het functioneel ontwerp bevat de documentatie voor de functionele kant van het systeem. Hierin wordt ingegaan op de functies van de front- en backend zodat er een beeld geschept wordt van hoe de functies te gebruiken zijn. Er is een functioneel ontwerp gemaakt omdat het dient als een handleiding voor het gebruik van de front- en backend. Hierdoor zal het duidelijk worden hoe de verschillende functies in het systeem gebruikt dienen te worden.

Prototype
Voor de realisatie is een deel van het systeem gerealiseerd in de vorm van een prototype. Het prototype is gemaakt omdat er hiermee gevalideerd kan worden of het theoretische onderzoek valide, het gaat hierbij zowel om de technische als functionele werking. Bovendien dient het prototype als resultaat voor het onderzoek naar hoe het systeem gerealiseerd kan worden.

Testen
Voor de realisatie zijn unit-, widget(gui)-, database-, exploratory- en acceptatietesten geschreven en/of uitgevoerd. De unit-, widget- en database testen zijn geschreven om te valideren of de verschillende functies en componenten correct functioneren. De acceptatietesten zijn uitgevoerd om te valideren of het prototype (op verschillende momenten tijdens de realisatie) voldoet aan de vraag en wensen van de opdrachtgever. Tot slot is er ook automatisch getest door middel van handmatig testen, hierbij wordt er tijdens het programmeren een script of functie uitgevoerd om te valideren of het resultaat ook is wat er wordt verwacht.

Als resultaat en aanvulling op de uitgevoerde testen is er respectievelijk een code-coverage percentage gegenereerd op basis van de geteste regels code van de front- en backend.

Aanpak
De gebruikte technieken zijn gekozen op basis van de conclusies van de deelvragen. Voor de front-end is er een schematisch design gerealiseerd als houvast voor het realiseren van het prototype, zie bijlage 9 voor meer informatie over de realisatie hiervan. Verder is de realisatie uitgevoerd op basis van een combinatie tussen Acceptance Test Driven Development en Behaviour Driven Development. Hier is voor gekozen omdat er van tevoren nog niet bekend was wat de precieze functionele uitkomst van elke requirement was, bovendien was er ook een grote leercurve voor de afstudeerder wat meer risico bracht bij de realisatie. Uiteraard zijn van alle taken die uitgevoerd dienden te worden toegevoegd aan het Kanban bord.

Definition of done
Een implementatie taak van het Kanban bord wordt als afgerond gezien als de implementatie correct functioneert in de app, getest is en de nodige documentatie is toegevoegd aan de bijbehorende documenten. Bovendien dient de totale code coverage van de aangepaste applicaties minimaal 80% te zijn zodat code kwaliteit gewaarborgd wordt.

5. Onderzoek

In het hoofdstuk *Onderzoek* zal het uitgevoerde onderzoek naar de antwoorden op de deelvragen beschreven worden. Bovendien zullen de conclusies horend bij elke deelvraag behandeld worden.

5.1 Deelvraag 1: Wat is/zijn de beste techniek(en) om een iOS en Androidapplicatie te ontwikkelen die het mogelijk maakt om de gewenste functies te implementeren?

In deelvraag 1 is er onderzocht wat de beste technieken zijn voor het ontwerpen en realiseren van een iOS en Androidapplicatie die het mogelijk maken om de gewenste functionaliteiten te implementeren.

In het literatuuronderzoek is er dieper onderzoek gedaan naar potentiële technieken voor het realiseren van de applicaties.

5.1.1 Resultaten

De onderzoeksresultaten per techniek worden beschreven in de onderstaande hoofdstukken.

Literatuuronderzoek

De resultaten van het literatuuronderzoek staan hieronder beschreven. Eerst wordt weergegeven welke resultaten uit de zoekopdrachten kwamen. Daarna worden de resultaten en conclusies of technieken wel of niet geschikt zijn als front-end techniek beschreven. De volledige uitwerking van het onderzoek is te vinden in bijlage 2, hoofdstuk 3.

Uit de zoekopdrachten zijn de volgende technieken naar voren gekomen:

- Swift
- Objective-C
- Java
- Kotlin
- Flutter (Dart)
- React-native (Javascript)
- Xamarin (C#)

De technieken worden op basis van het onderzoek beoordeeld of ze geschikt zijn voor het project in de onderstaande hoofdstukken.

Swift (native iOS)

Uit het onderzoek kan geconcludeerd worden dat Swift geschikt is voor het realiseren van de front-end applicatie. Het bevat de benodigde support en zorgt voor de snelste iOS applicaties in zijn soort. Omdat de taal voor iOS is geschreven is het dus ook geoptimaliseerd voor Apple apparaten.

Objective-C (native iOS)

Uit het onderzoek kan geconcludeerd worden dat Objective-C niet geschikt is als programmeertaal voor dit project. Ten eerste is Objective-C de minder snelle versie van Swift, ten tweede is de kans op datalekken en stuk groter met Objective-C vanwege het gebruik van pointers.

Java (native Android)

Java zou een geschikte taal zijn voor het ontwikkelen van de Androidapplicatie van het platform. Het voldoet aan de eisen en is native, wat betekent dat de performance voldoende zal zijn (afhankelijk van het device waar het op draait).

Kotlin (Cross platform)

Kotlin zou een geschikte taal zijn voor het ontwikkelen van een Android en iOS applicatie Volgens de developer website van Kotlin zou het op performance- en securitygebied goed moeten functioneren. Maar omdat Google heeft aangegeven dat Kotlin de voorkeur heeft voor het ontwikkelen van Androidapplicaties zou het wijzer zijn om een alternatief te gebruiken.

Flutter (Dart - Cross Platform)

Op basis van het onderzoek kan er geconcludeerd worden dat Flutter een geschikte techniek is voor de realisatie van het product. Het is een cross platform ontwikkel framework wat betekent dat het een Android- en iOS applicatie kan maken met één codebase. Een klein nadeel is dat het een hybride app is waardoor de performance niet optimaal is, maar nog steeds goed.

React Native (JavaScript – Cross platform)

Op basis van het onderzoek kan geconcludeerd worden dat React Native een geschikte techniek is voor de realisatie van het project. Wel zal er goed gekeken moeten worden naar de performance, het zou mogelijk kunnen zijn dat andere oplossingen bij dit punt geen issue hebben.

Xamarin (C# - Cross platform)

Op basis van het onderzoek kan geconcludeerd worden dat Xamarin geschikt zou kunnen zijn als ontwikkel framework voor dit project. Toch zouden andere oplossingen een hogere plek kunnen hebben binnen de ‘geschikte technieken lijst’ vanwege de relatief kleine community en performance.

5.1.2 Conclusie

Op basis van de onderzoeksresultaten is ervoor gekozen om Flutter te gebruiken de redenen hiervoor zijn hieronder opgesomd:

- Flutter biedt cross platform ontwikkeling. Met andere woorden worden er 2 applicaties ontwikkeld door maar 1 codebase te bewerken. Dit zorgt voor ±50% minder ontwikkelkosten in het geval dat Sosialforce de applicatie in de toekomst besluit uit te breiden.
- De performance zou zo goed als gelijk moeten zijn aan native applicaties van zowel Android als iOS. Dit betekent dat het voldoet aan de performance criteria.
- De ondersteunende OS-versies van Flutter voldoen aan de eisen van de opdrachtgever. Dit betekent dat het voldoet aan de ondersteuningscriteria.
- Er is duidelijke documentatie en een grote community aanwezig. Dit betekent dat het voldoet aan de duidelijkheids criteria.
- De kennis binnen Sosialforce kan worden uitgebreid als gevolg van innovatie.

Zoals eerder benoemd, is de conclusie getrokken op basis van de volgende criteria:

- Performance
- Ondersteuning (iOS en Android)
- Duidelijkheid (Is de techniek bekend/ is er een (grote) community en is er duidelijke documentatie)

5.2 Deelvraag 2: Wat is/zijn de beste techniek(en) om een API te ontwikkelen die voldoen aan de eisen van de opdrachtgever?

Voor deze deelvraag is er een literatuur- en veldonderzoek uitgevoerd. Door eerst het veldonderzoek uit te voeren is er achterhaald wat de huidige situatie is en waarmee er rekening gehouden dient te worden voor een integratie op het huidige systeem. In het literatuuronderzoek zijn de resultaten te vinden van het onderzoek naar verschillende API-structuren en querytalen.

5.2.1 Resultaten

De resultaten van deelvraag 2 'Wat is/zijn de beste technieken om een API te realiseren die voldoet aan de eisen van de opdrachtgever?' staan hieronder weergegeven.

Veldonderzoek - Observatie

Voor het veldonderzoek is er een observatie uitgevoerd waarbij er gekeken is naar het huidige systeem. Hierbij is gekeken naar welke API-techniek en structuur worden gebruikt. Bovendien is er ook gevraagd naar wat volgens R. ten Berge de problemen zijn die opgelost dienen te worden.

Met de verkregen informatie van het veldonderzoek kan er geconcludeerd worden dat er gebruik gemaakt gaat worden van Java Spring Boot als API framework en MySQL als database. De reden hiervoor ligt bij de managebility, dit framework wordt namelijk in alle projecten van Sosialforce gebruikt voor de API. Overigens zijn de volgende punten bekend geworden, deze kunnen meegenomen worden in het vervolg van het huidige vraagstuk:

- Is er een overzichtelijkere en flexibelere API-architectuur dan REST die past bij het huidige scenario?
- Is er een flexibelere querytaal dan SQL die toegepast kan worden in het huidige scenario en die compatible is met zowel MySQL als Java Spring Boot?

Literatuuronderzoek

In het literatuuronderzoek is er onderzoek gedaan naar manieren om de API flexibeler te maken met het gebruik van Java Spring Boot en MySQL als database. Dit wordt gedaan door te kijken naar mogelijke structuren die REST kunnen vervangen en/of een alternatieve query taal voor SQL om database calls flexibeler te maken. Voordat dit weergegeven wordt zullen eerst de resultaten van de zoekopdrachten weergegeven worden.

Uit deze zoekopdrachten kwamen de volgende resultaten die onderzocht zijn:

- REST
- GraphQL
- gRPC
- SOAP
- Falcor

Elke techniek zal op basis van het onderzoek worden beschreven en beoordeeld of ze geschikt zijn voor het project op de volgende pagina's. Zie bijlage 2 hoofdstuk 3.2 voor de diepere kijk op de werking van de onderzochte API-architecturen.

REST-architectuur

REST staat voor Representational State Transfer Application Program Interface. REST wordt vaak gebruikt om webservices te bouwen. Het gebruikt HTTP-methoden om data tussen cliënt en server te POST-en en GET-ten. Met REST kan response gegeven worden in JSON, XML of andere tekstindelingen. Zie tabel 4 voor een overzicht van de voor- en nadelen van een REST API gebaseerd op het uitgevoerde onderzoek.

Tabel 4: Voor- en nadelen van REST

Voordelen	**Nadelen**
Goed te organiseren met de standaard patronen die aangehouden worden.	Een Stateless API kan zorgen voor meer berekeningen en behouden van de state door de cliënt wat een complexere cliënt applicatie kan betekenen.
Schaalbaar in de zin dat de ene call de andere niet beïnvloed en dat er makkelijk meerdere cliënts bij kunnen komen.	Het gebruik van URL's voor calls is niet geschikt voor vertrouwelijke datatransmissie tussen cliënt en server (indien niet encrypted)
Flexibiliteit door verschillende mogelijkheden van response formaten zoals JSON en XML.	
Lagensysteem voor duidelijke scheiding tussen functies in de API.	
Data is cachable.	

GraphQL-architectuur

GraphQL is een door facebook ontwikkelde Querytaal en architectuur. In tegenstelling tot REST waarbij je een verzameling van URL's hebt waar je calls naartoe maakt, heb je bij GraphQL één URL waarbij je verschillende typen data kunt opvragen. Bovendien zijn GraphQL API's niet front-end afhankelijk, dit maakt ze goed backwards compatible. Zie tabel 5 voor een overzicht van de voor- en nadelen van het gebruik van GraphQL, opgesteld op basis van het uitgevoerde onderzoek.

Tabel 5: Voor- en nadelen van GraphQL

Voordelen	**Nadelen**
Cliënts hebben maar 1 API endpoint die ze aanroepen.	Response status codes zullen altijd 200 zijn.
Calls zijn erg flexibel omdat er specifiek bepaald kan worden wat er opgehaald dient te worden.	Geen standaard cache support omdat alle calls als POST gezien worden.
Geen over- of onder-fetching problemen door specifieke data opvragen.	

gRPC-architectuur

gRPC gebruikt protocol buffers als Interface Definition Language (IDL) en onderliggend boodschap formaat. Er wordt gebruik gemaakt van een server/cliënt structuur waarbij de cliënt een methode opvraagt van de server inclusief parameters en return type. Protocol buffers zijn gestructureerde dataprofielen, opgebouwd uit name-value paren. De voor- en nadelen van het gebruik van gRPC staan weergegeven in tabel 6.

Tabel 6: Voor- en nadelen van gRPC

Voordelen	**Nadelen**
Gebruik van proto3 voor compacte messages.	Gelimiteerde browser support.
Bi-directional streamen, met andere woorden zowel naar de service als naar de cliënt tegelijk.	Geen voor ingestelde statuscodes.

SOAP

SOAP staat voor Simple Object Access Protocol. Het protocol is afhankelijk van XML en is samen met gedefinieerde schema's een goed 'messaging framework'. Wel houdt het zich sterk aan voor-ingestelde standaarden zoals berichtstructuren, encodingregels en een standaard van het verstrekken van requests en responses. SOAP wordt gezien als een zwaar protocol ten opzichte van REST door de grote overhead van een message. Zie tabel 7 voor een overzicht van de voor- en nadelen van het gebruik van een SOAP API.

Tabel 7: Voor- en nadelen SOAP

Voordelen	**Nadelen**
Keuze tussen stateless en statefull	Veel pre-defined regels, niet flexibel
Hoge security	Alleen XML
Keuze tussen diverse transfer protocollen	Grote payload
	Calls kunnen niet gecached worden

Falcor

Falcor is een API-stijl gemaakt door Netflix. Falcor is een JavaScript library. Falcor maakt het mogelijk om alle backend data als één virtueel JSON-object terug te geven. Data die een cliënt nodig heeft kan worden vrijgegeven via één URL, dit zorgt voor verminderde vertraging (latency) als gevolg van minder opeenvolgende server calls. Zie tabel 8 voor een overzicht van de voor- en nadelen van het gebruik van Falcor.

Tabel 8: Voor- en nadelen Falcor

Voordelen	**Nadelen**
Efficiënt data opvragen	Alleen voor Javascript
Alle data zit in één JSON model	
Ingebouwde SET en GET operations door het gebruik van het Falcor model	

5.2.2 Conclusie

Op basis van de resultaten is ervoor gekozen om GraphQL als API-architectuur te gebruiken. De redenen hiervoor staan hieronder opgesomd, hierbij worden ook de criteria die ze ondersteunen bij benoemd:

- De API is erg eenvoudig omdat er maar naar 1 endpoint calls worden gemaakt, dit geeft overzicht.
- Het is erg flexibel omdat er specifiek opgevraagd kan worden wat voor data er nodig is, wat overigens ook over- en onder-fetching kan voorkomen, dit geeft flexibiliteit en overzichtelijkheid.
- Het ondersteunt verschillende response types (XML, JSON, etc.) en kan gebruikt worden met Java, dit biedt voldoende ondersteuning voor de criteria.
- GraphQL is erg schaalbaar in het toevoegen van nieuwe calls, dit zorgt voor flexibiliteit.

Waarom er niet voor andere API-architecturen is gekozen:

- Falcor ondersteunt alleen Javascript, de wens is dat er Java gebruikt wordt.
- SOAP is minder flexibel dan REST, de wens was dat het flexibeler was dan REST
- gRPC heeft gelimiteerde browser support wat nadelig zou kunnen werken in de toekomst.

5.3 Deelvraag 3: Welke data gaat er opgeslagen worden in de database? En welk databasetype is geschikt?

In deelvraag 3 staan de resultaten naar het onderzoek naar welke data er opgeslagen gaat worden, dit wordt aangevuld met de resultaten van het onderzoek naar welk databasetype geschikt is.

5.3.1 Resultaten

De resultaten van deelvraag 3 'Welke data gaat er opgeslagen worden in de database? En welk databasetype is geschikt?' staan hieronder weergegeven.

Interview + observatie

Het onderzoek heeft helder gemaakt welke data nodig is voor de functionaliteit van het product. Zie bijlage 1 voor het interview met vragen, antwoorden en onderbouwing per vraag. Zie tabel 9 voor een overzicht van de data die opgeslagen gaat worden.

Tabel 9: Overzicht data

Car
Merk
Uitvoering
Meterkast
Foto
Verdere gegevens van openRDW, zie bijlage 2, blz. 24 voor een volledig overzicht van de auto data.
Kenteken

Trip
ID
Account ID
Datum
Beschrijving
Afstand
Trip type
Trip interval
Interval type

Account
Naam
Wachtwoord
Access token
Email
Gender
Profielfoto
ID

Mobiliteitsprofiel
ID
Account ID
Auto ID
Rijstijl
Laadopties
Profielnaam
Mode

Literatuuronderzoek

Er is onderzoek gedaan naar welke relationele databases er gebruikt zouden kunnen worden. Er was hier een handjevol mogelijkheden voor, maar er is gekozen voor MySQL omdat deze databaseserver standaard binnen Sosialforce gebruikt wordt. Ondanks dat deze conclusie is gebaseerd op managebility, is er wel gekeken naar alternatieven zoals MariaDB en PostgreSQL.

5.3.2 Conclusie

Op basis van de onderzoeksresultaten is ervoor gekozen om een bepaalde collectie data op te slaan en om gebruik te maken van de relationele databaseserver MySQL. Zie figuur 2 voor de data die opgeslagen gaat worden. Er is hiervoor gekozen omdat:

- De data die opgeslagen gaat worden nodig is voor de functionaliteit van het systeem.
- Een relationele database essentieel is in dit scenario omdat erg veel data met elkaar verbonden is. Ook kan er met een relationele database integriteit afgedwongen worden. Denk hierbij aan foreign keys, zodat er niet zomaar een object verwijderd kan worden dat nog verbonden is aan een ander object.
- MySQL als enige databaseserver gebruikt wordt binnen Sosialforce, dit draagt bij aan de managebility.
- MySQL de nodige functies bevat om de database juist te laten functioneren.

5.4 Proof of concept

Om de praktische werking van de resultaten van deelvragen 1, 2 en 3 te valideren, is er een Proof of concept gerealiseerd. In de conclusie wordt de uitkomst van het Proof of concept besproken. Onder resultaten zal het resultaat van de PoC worden besproken. De volledige uitwerking van het Proof of concept is te vinden in bijlage 2, hoofdstuk 6. Er is overigens geen code oplevering van de PoC zelf, de PoC heeft gediend als basis van het uiteindelijke prototype. Zie bijlage 10 voor de code van het prototype waar de PoC code in verwerkt zit.

5.4.1 Resultaten

Met de PoC is een front-end, backend en database die de GraphQL structuur aanhouden gerealiseerd. De front-end kan een GET- en POST-request sturen naar de backend. Deze data zal worden gepost of opgeslagen in de database. De front- en backend zullen in de volgende hoofdstukken uitgelegd worden. De database heeft geen speciale werking in dit scenario, het slaat alleen een Car object op met een merk, type en ID, om deze reden wordt er geen uitleg over de database gegeven.

Front-end

De front-end is gemaakt in Flutter en bevat de functionaliteit om een ‘GetAllCars’ query en ‘deleteCar’ mutatie uit te voeren. De ‘GetAllCars’ query wordt uitgevoerd met behulp van de ‘Query’ widget, hieraan wordt de gedefinieerde query in String vorm meegegeven, deze wordt vervolgens door de widget uitgevoerd wanneer deze wordt aangeroepen. Hetzelfde gebeurt bij de ‘deleteCar’ mutatie, behalve dat dit wordt uitgevoerd door de Mutation widget. De ‘graphql_flutter’ library is gebruikt om GraphQL te implementeren in de front-end.

Backend

De backend is gemaakt in Java Spring Boot. Het bevat de functionaliteit om GraphQL requests te ontvangen en op de juiste manier te verwerken. Dit gebeurt bijvoorbeeld door het ontleden van de parameters uit de binnenkomende query of mutatie, om het vervolgens te binden aan de juiste methode. Zie figuur 1 voor het class diagram van de backend.

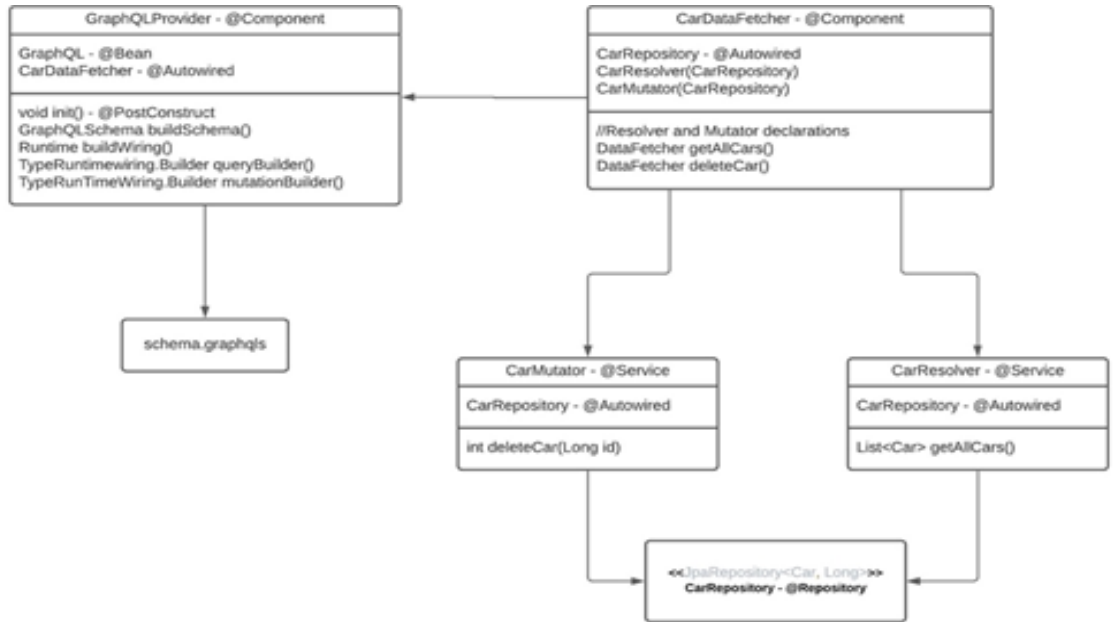


Figure 1: Proof of concept class diagram

De backend werkt in volgorde van de volgende punten:

- In het schema.graphqls bestand worden de types, query's en mutaties gedefinieerd.
- De GraphQLProvider class verbindt de definities uit het schema.graphqls bestand aan de bijbehorende fetcher(verwerk) classes, in dit geval de CarDataFetcher.
- De CarDataFetcher ontleedt de binnenkomende query's en mutaties van de parameters en bind de query of mutatie aan de bijbehorende functie. Bij een mutatie gebeurt dit bij een Mutator class, bij een query is dit een Resolver class.
- De Mutator of Resolver voert de functie uit, deze functies bestaan uit calls naar de repository classes.
- De repository classes zijn de brug tussen de backend en de database.

5.4.2 Conclusie

Op basis van de resultaten van het Proof of concept is er gevalideerd dat GraphQL correct kan samenwerken met Flutter, Java Spring Boot en dat deze backend ook kan communiceren met een MySQL database. Dit is gevalideerd door een werkende applicatiecyclus te maken waarbij een Flutter applicatie een query en mutatie heeft uitgevoerd naar een Java Spring Boot API die GraphQL gebruikt voor operaties naar de database.

Hierbij is ook gekeken hoe deze cyclus getest kan worden per individueel component. Er is daarbij ondervonden hoe antwoord van GraphQL calls op de front-end gesimuleerd kunnen worden. Dit is ook gelukt op de backend. Ook is er getest hoe database operaties getest kunnen worden door het uitvoeren van database functies op een lokale database.

5.5 Deelvraag 4: Wat is de meest efficiënte databasestructuur in dit scenario?

In deelvraag 4 staan de resultaten van het onderzoek naar wat de meest efficiënte databasestructuur is. De resultaten zijn onderverdeeld in de resultaten van het literatuuronderzoek, veldonderzoek en werkplaatsonderzoek.

5.5.1 Resultaten

De resultaten van deelvraag 4 'Wat is de meest efficiënte databasestructuur in dit scenario?' staan hieronder weergegeven.

Literatuuronderzoek

Uit het onderzoek op basis van het artikel van Lucid (Z.D.). is naar voren gekomen waar een database aan moet voldoen om het te valideren als efficiënt. Zie tabel 10 voor een overzicht van principes die aangehouden dient te worden bij het ontwerpen van een databasestructuur om het zo efficiënt mogelijk te maken.

Tabel 10: Principes van een efficiënte databasestructuur

Principes voor een efficiënte databasestructuur
Zorg dat een attribuut maar één waarde kan hebben.
Attributen van een dataobject moeten volledig afhankelijk zijn van één primary key, dit kan gewaarborgd worden door het gebruik van koppeltabellen.
Elke non-key attribuut moet onafhankelijk zijn van een ander dataobject.
Voeg indexen toe aan kolommen.
Stel regels in voor gegevens, zoals dat een primary key niet NULL mag zijn.

Veldonderzoek

In het veldonderzoek zijn er een aantal punten gevonden die in de oude databasestructuur beter kunnen. De punten staan hieronder weergegeven:

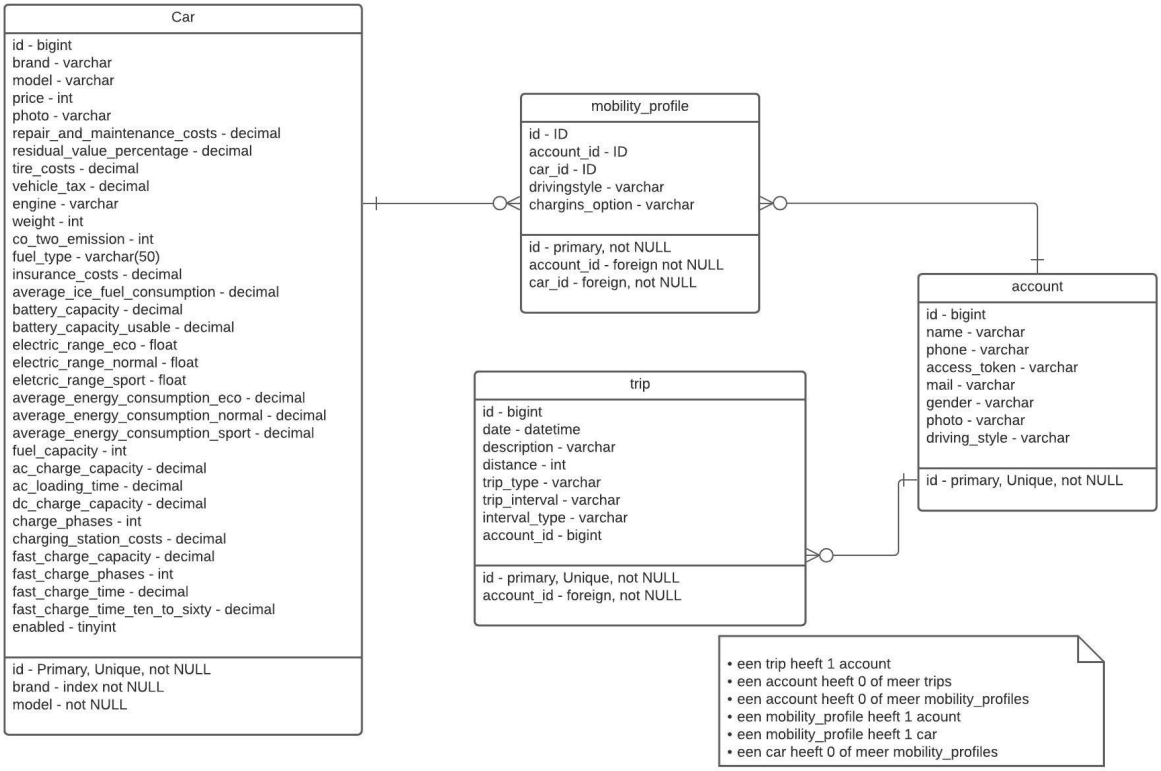
- Er worden te veel koppeltabellen gebruikt, op veel plekken is het niet nodig omdat er bijvoorbeeld one-to-one relaties zijn.
- Veel onnodige data wordt opgeslagen, zoals berekeningen, terwijl het alleen nuttig is om het resultaat van deze berekening op te slaan.

Deze punten zullen meegenomen worden in het ontwerp van de databasestructuur.

Werkplaatsonderzoek

Met de verkregen kennis van de vorige twee onderzoeken zijn er een aantal databasestructuren gemaakt. Zie figuur 2 voor de definitieve versie van de databasestructuur. Zie bijlage 2 hoofdstuk 5.3.1 voor alle versies van het prototype als gevolg van het werkplaatsonderzoek.

Opmerking: elke primary key bevat een index voor efficiëntie.



Figuur 2: Database structuur

5.5.2 Conclusie

Na meerdere iteraties als resultaat van het werkplaats- en veldonderzoek is ervoor gekozen om de databasestructuur die in figuur 2 te gebruiken voor het te realiseren system. Er is hiervoor gekozen omdat:

- Deze structuur alleen data laat opslaan voor het correct functioneren van het systeem. Hiermee wordt niet te veel data opgeslagen wat schijfruimte bespaard.
- De structuur ontworpen is op basis van de principes van een efficiënte database.

Kanttekening: dit is de databasestructuur voor de huidige versie van het prototype. Indien er besloten wordt om functies toe te voegen die andere data vereist, zou dit betekenen dat de databasestructuur in figuur 1 alsnog aangepast wordt.

5.6 Realisatie

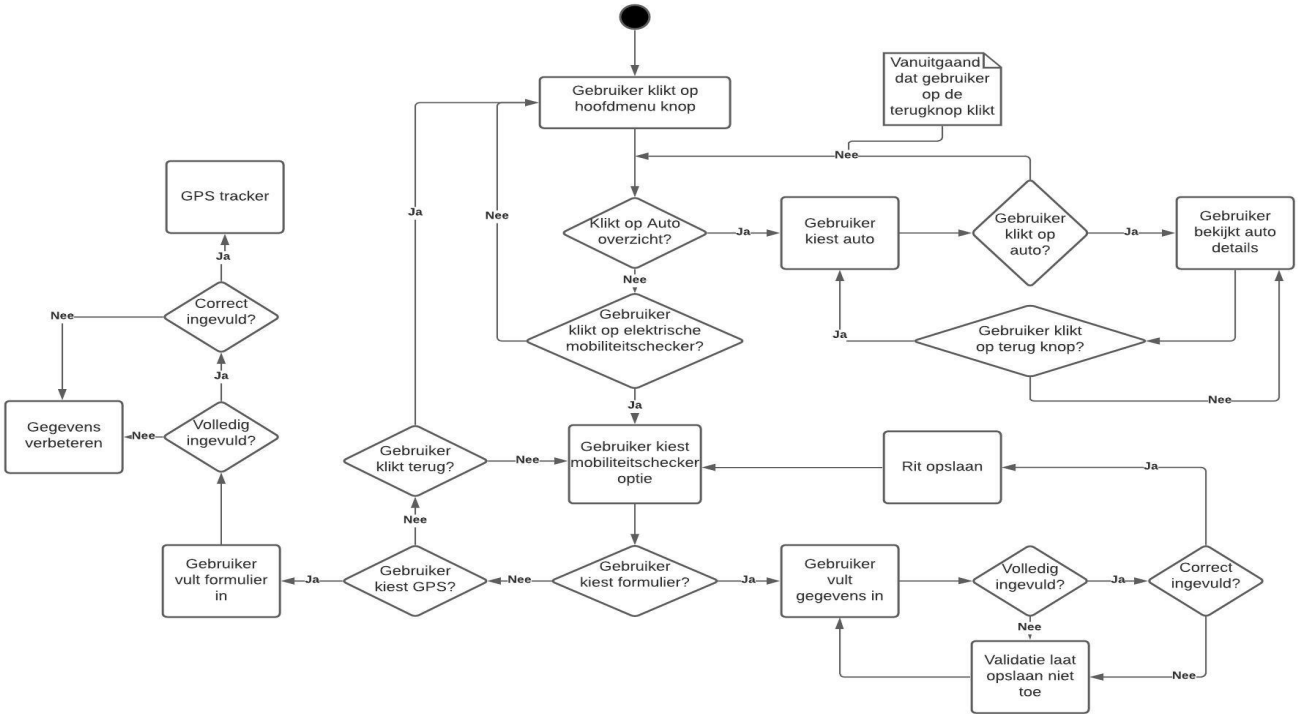
De schriftelijke resultaten van de realisatie zijn in de onderstaande hoofdstukken te vinden. Zie bijlage 10 voor de praktische realisatie (het prototype).

5.6.1 Functioneel ontwerp

Het functioneel ontwerp beschrijft de functies van de front- en backend. Er wordt hier ingegaan op de werking van de globale flow van de front-end en de werking van de functies van de front- en backend. Zie bijlage 7 voor de volledige uitwerking van de functionele documentatie

Front-end

De front-end bevat veel functies die op verschillende manieren bereikbaar zijn.



Figuur 3: Activiteiten diagram front-end
Opmerking: bij elke keuze kan de gebruiker ervoor kiezen de app af te sluiten, wat dus betekent dat bij elke keuze ook een eindpunt hoort.

Zie figuur 3 voor een activiteitendiagram waarbij de globale flow van de front-end in kaart wordt gebracht. Deze flow omschrijft hoe elke functie in de applicatie bereikt kan worden. Zie bijlage 7, hoofdstuk 1.2 voor een specifieke handleiding van de individuele functies en schermen. De belangrijkste functies van de applicatie zullen in de onderstaande hoofdstukken omschreven worden. Bovendien wordt er onderbouwing gegeven waarom de functie is gebouwd en welke requirement nummers het afdekt, zie tabel 2 en 3 voor de beschrijving van de requirements.

Auto overzicht

In het auto overzicht wordt een lijst van kaartjes met auto informatie weergegeven. Elk kaartje bevat een stock foto, automerk, model en eventuele uitvoering. Deze data wordt opgehaald vanuit de database die weer samengesteld is uit data van openRDW. Er kan hier een auto geselecteerd worden waardoor de gebruiker naar een nieuw scherm wordt geleid waarbij gedetailleerde informatie wordt getoond over de auto. Het auto overzicht is gebouwd om in de gebruiker te kunnen laten oriënteren welke auto's er op de markt aanwezig zijn, bovendien dekt het requirement 2 en 3 af.

Rit registratie via formulier

De gebruiker kan ritten registreren door het invullen van een formulier. Hierbij moet de gebruiker een datum, beschrijving, rit type, interval, intervaltype en afstand invullen. Het formulier bevat validatie en zal de niet of niet correct ingevulde velden rood laten oplichten en een melding tonen indien deze niet goed zijn ingevuld. Ritten worden doorgestuurd en opgeslagen in de database. Deze functie is gebouwd om gebruikers eenvoudig een rit te laten registreren die uiteindelijk gebruikt kan worden voor de berekening van het verbruik. Ook dekt het requirement 4, 7, 8 en 9 af.

Rit registratie via GPS

De gebruiker kan ritten registreren met behulp van een gps-tracker. Hiervoor dient de gebruiker eerst een mobiliteitsprofiel te selecteren. In dit profiel wordt de auto waarin gereden wordt, de rijstijl en de laadopties vastgelegd. Nadat er een profiel is gekozen kan de gps-tracker worden gestart, hierbij zullen de huidige snelheid en de afgelegde afstand realtime worden bijgehouden. Indien de gebruiker een tussenstop heeft kan de gps-tracker op pauze worden gezet. Als de rit voorbij is, kan er een overzicht worden opgevraagd van de afgelegde rit met het bijbehorende profiel. De gebruiker heeft bij het overzicht de keuze om de rit wel of niet op te slaan. Deze functie is gebouwd zodat gebruikers een alternatief met accurater resultaat hebben voor het registratieformulier, met de gps-gegevens wordt er namelijk van start- tot eindpunt bijgehouden wat de afgelegde afstand is. Deze functionaliteit dekt requirement 5, 6, 6.1 en 6.2 af.

Auto zoeker

De gebruiker kan een auto opzoeken op basis van een zoekopdracht met het auto merk en model of een kenteken. Bij de merk en model zoekopdracht wordt een lijst van auto's met dat merk en model getoond. Bij de kenteken zoekopdracht wordt de auto (indien aanwezig) in de lijst weergegeven. Indien er een auto is geselecteerd wordt dit weergegeven in een kaartje waarin het merk en model van de auto zijn te vinden, ook bevat het kaartje een vuilnisbak icoon waarmee de auto gedeselecteerd kan worden. Deze functie is gebouwd voor de vergelijking en de gps-tracker. De gebruiker kan nu zijn/ of haar eigen auto opzoeken en selecteren en een willekeurige auto in Nederland om me te vergelijken. Deze functie dekt requirement 7 en 10 gedeeltelijk af. (voor deze requirements dient een auto gekozen te worden)

Mobiliteitsprofiel maken

De gebruiker kan een mobiliteitsprofiel maken door het invullen van een formulier. Hiervoor dient een profielnaam, rijgedrag, laadopties en een auto geselecteerd te worden. Ook dit formulier bevat validatie. Deze functie is gemaakt omdat een gebruiker een mobiliteitsprofiel nodig heeft om een overzicht te kunnen krijgen van het benodigde aantal tank- en oplaadbeurten. Deze functie is nodig om een deel van requirement 10 af te dekken en dekt requirement 12 volledig af.

Ritten vergelijken

Een rit kan vergeleken worden op gebied van verbruik door het selecteren van een mobiliteitsprofiel en een auto om mee te vergelijken. Het rijgedrag en de laadopties van het mobiliteitsprofiel worden gebruikt in de vergelijking, evenals de auto van het profiel die wordt gezien als de auto waarin is gereden. De tweede auto die wordt gekozen is de auto waarmee wordt vergeleken. Op basis van de verbruiksgegevens die bekend zijn zal op basis van de rijstijl en laadopties (indien het om een elektrische auto gaat) een calculatie worden uitgevoerd waarbij de uitkomst zal laten zien hoeveel brandstof er is verbruikt en hoe vaak er getankt moet worden of hoe vaak en hoelang de auto opgeladen moet worden op basis van de laadopties. Deze functie is gemaakt voor het vergelijken van het verbruik tussen twee auto’s, met het resultaat van de vergelijking kunnen autodealers een advies voor consumenten opstellen. De functie dekt requirement 10 gedeeltelijk af. Samen met de functies ‘auto zoeker’ en ‘mobiliteit profiel maken’ is requirement 10 volledig afgedekt.

Backend

De backend heeft een aantal REST calls waarmee functies met betrekking tot de openRDW koppeling kunnen worden geactiveerd. De functies worden in de onderstaande hoofdstukken beschreven. Zie bijlage 7, hoofdstuk 2.1 voor een specifieke handleiding van hoe de functies aangeroepen kunnen worden.

Auto-importeer functie

De auto-importeer functie activeert het importeren van de volledige dataset ‘Gekentekende voertuigen’ en ‘Brandstof gekentekende voertuigen’ van openRDW. Hierbij zal er op basis van kenteken gekeken worden of de auto al bestaat in de database van het eigen platform. Als dit zo is dan zal de record aangepast worden, zo niet dan zal er een nieuwe record gecreëerd worden.

Het proces zal worden opgestart in een aparte thread, hierdoor zal de main thread (databaseconnectie) niet geblokkeerd worden waardoor calls vanaf de front-end nog steeds mogelijk zijn. Ook zal dit proces zelf meerdere threads maken zodat het verwerken en opslaan van data efficiënter gebeurt. De openRDW data wordt opgehaald in batches van 10.000 records. De data ophalen in batches zorgt voor een snellere fetch dan wanneer alles in één keer opgehaald wordt wat CPU en memory bespaard.

Auto-importeer interruptie functie

De auto-importeer functie duurt lang door de grote hoeveelheid data die verwerkt dient te worden. Indien de functie gestopt moet worden kan de auto-importeer interruptie functie worden aangeroepen. Door deze functie aan te roepen zal de op dat moment uitgevoerde batch worden afgemaakt waarna het importproces vervolgens wordt afgekapt.

Auto update

De eigen database blijft gesynchroniseerd aan de openRDW datasets met behulp van de auto update functie. Deze functie is gepland en wordt dagelijks om 2:00 uur uitgevoerd. Hierbij worden de updates van de laatste 24 uur van openRDW naar binnengehaald. Door dit elke 24 uur te doen blijft de eigen data gesynchroniseerd aan de data van openRDW.

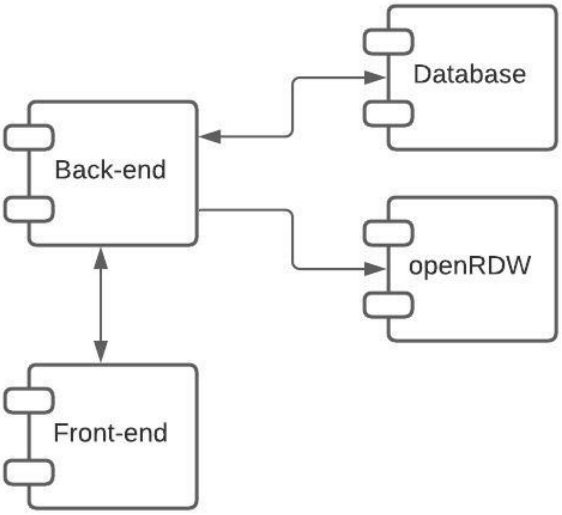
5.6.2 Technisch ontwerp

Het technisch ontwerp wordt beschreven in de onderstaande hoofdstukken. Er wordt eerst ingegaan op de systeemarchitectuur en de communicatie tussen de verschillende componenten. Vervolgens worden de verschillende componenten behandeld op basis van technieken, structuur en overige belangrijke punten. Zie bijlage 8, hoofdstuk 2.3 en hoofdstuk 3.6 voor een gedetailleerde uitleg en code voorbeelden van belangrijke functies in het systeem.

Architectuur

Het prototype bestaat uit 3 verschillende componenten:

1. De front-end
2. De backend
3. De database

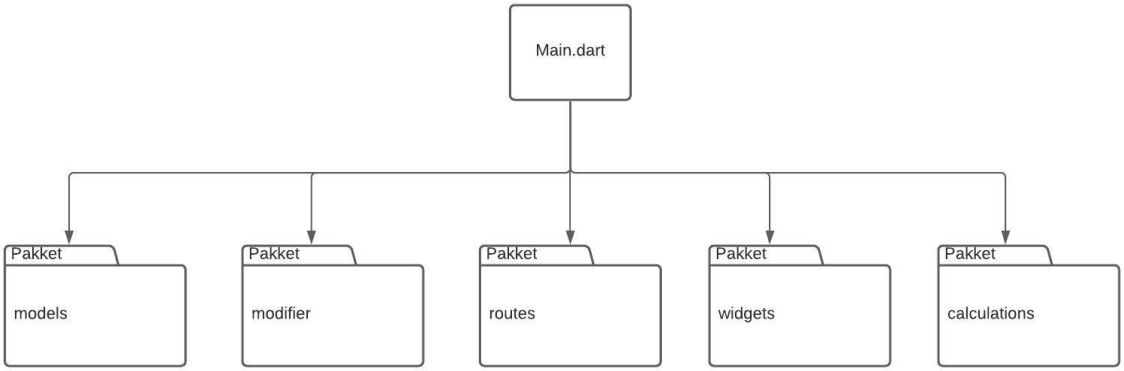


Figuur 4: Systeemarchitectuur

Zie figuur 4 voor de systeemarchitectuur en de communicaties tussen de verschillende componenten. Zoals te zien communiceert de front-end alleen met de backend. Dit komt omdat de front-end alleen data opvraagt van en verstuurt naar de backend. De backend communiceert met de front-end, database en de openRDW API. De communicatie met de front-end bestaat uit het versturen van opgevraagde data en het ontvangen van ontvangen data. Communicatie met de database bestaat uit ook uit het versturen van data en het ontvangen van opgevraagde data. Verder zal de backend alleen data opvragen van de openRDW API. Tot slot zal de database alleen data ontvangen van en versturen naar de backend.

Front-end

De front-end is opgebouwd in Dart in combinatie met Flutter. Door het gebruik van Flutter kan er een Android- en iOS app gegenereerd worden. De front-end is verantwoordelijk voor het weergegeven en samenstellen van nieuwe data voor een gebruiker. De front-end communiceert met die backend, voor deze communicatie wordt de GraphQL taal aangehouden. Hierbij worden query's op de front-end samengesteld waardoor er precies opgevraagd kan worden wat er nodig is. De functionaliteiten van de front-end zijn verdeeld over verschillende packages. Elke package bevat classes die dienen voor soortgelijke functies.



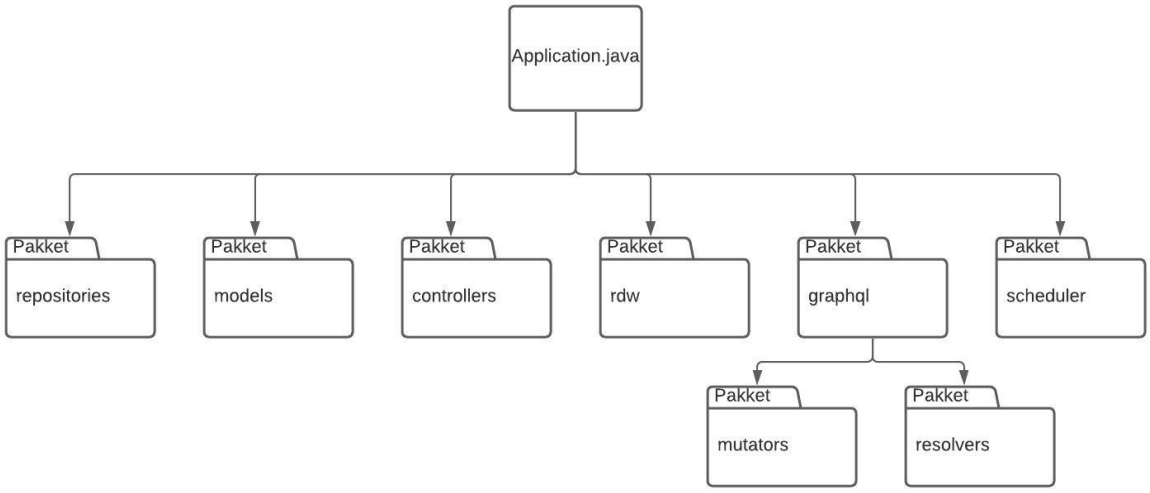
Figuur 5: Package structuur front-end

Zie figuur 5 voor de packagestructuur van de front-end. Zoals te zien maakt de Main.dart class gebruik van 4 verschillende packages. De *Main.dart* class is verantwoordelijk voor het opstarten van de applicatie. De *models* package bevat alle models die gebruikt worden in de applicatie. De *modifier* package bevat classes die binnenkomende data aanpast naar leesbare of bruikbare data. De *routes* package bevat alle de classes die elk scherm van de applicatie opbouwt. De *widgets* package alle bouwstenen van de classes in de *routes* package. Tot slot bevat de *calculations* package de classes die nodig zijn voor het berekenen van het verbruik van een auto voor de rit vergelijk functie

Backend

De backend is opgebouwd in Java in combinatie Spring Boot. De backend kan gezien worden een brug tussen de front-end en database en tussen het openRDW en de database. De applicatie is opgebouwd met de GraphQL structuur en taal, hierdoor worden de query's verschoven naar de front-end en wordt het ophalen van specifiek attributen afgehandeld op de achtergrond door het gebruik van de externe GraphQL library. De communicatie tussen openRDW en de backend wordt geactiveerd door middel van diverse REST calls. Er is hiervoor gekozen omdat de openRDW koppeling een aparte service is die los staat van de backend.

Ook in de backend zijn de classes met soortgelijke functies verdeeld over verschillende packages.



Figuur 6: Package structuur backend

Zoals te zien in figuur 6 gebruikt de *Application.java* class, die ook de applicatie opstart, met veel verschillende packages. De *repositories* package bevat classes die de communicatie met de database regelt, JPA en Hibernate dienen als ORM in deze context. De *models* package bevat alle entiteit classes die worden opgeslagen in de database. De *controllers* package bevat de REST controller die wordt gebruikt voor het activeren van de functies met betrekking tot openRDW. De *rdw* package bevat alle classes die worden gebruikt voor het ophalen en opslaan van de data van openRDW. De *graphql* package bevat de classes die de query's en mutaties koppelen aan bijbehorende functies, waarvan de classes zijn gedefinieerd in *mutators* en *resolvers* packages. Tot slot bevat de *scheduler* package de classes voor het afhandelen van geplande taken.

De werkstructuur van de backend is te verdelen over drie verschillend flows:

1. De front-end flow
2. De geplande flow
3. De REST flow

De front-end flow werkt via de GraphQL structuur en taal en zorgt voor het afhandelen van POST- en GET requests die binnenkomen vanaf de front-end. De geplande flow gaat over de afhandeling van de uitvoering van geplande functies, in de huidige context gaat dit om het binnenhalen van updates van openRDW. De REST flow handelt de API-calls af die het proces van de import van de volledige openRDW datasets kunnen activeren of stoppen.

Database

De databaseserver is van het type MySQL. De database component communiceert met de database om binnenkomende data op te slaan en opgevraagde data op te zoeken en te versturen. Zie figuur 2 voor de gebruikte databasestructuur in het prototype.

Tools

Voor het realiseren van het prototype zijn verschillende tools gebruikt. De gebruikte tools zullen in de onderstaande hoofdstukken beschreven worden. Hierbij zal benoemd worden wat de tool is, waar het voor is gebruikt en waarom hiervoor is gekozen.

IntelliJ IDEA

De gebruikte ontwikkelomgeving voor het schrijven van de front- en backend applicaties is IntelliJ IDEA. Er is gekozen voor IntelliJ IDEA omdat het ondersteuning biedt voor zowel Flutter als Java Spring Boot. Bovendien had de afstudeerder al ervaring met deze tool.

Docker

Docker is een programma wat zogenoemde containers kan draaien waardoor een applicatie direct 'up & running' is. In dit geval gebeurt dit dus bij de database server. Er is gekozen voor Docker omdat het handig in gebruik is en database makkelijk integreerbaar maakt.

Docker-compose

De backend applicatie bevat een docker-compose file die ervoor zorgt dat de MySQL database wordt opgezet. Hierin worden onder andere de database gebruikersnaam, wachtwoord en versie in vastgesteld.

Altair GraphQL client

Altair GraphQL client is een applicatie waarin calls naar een GraphQL API gemaakt kunnen worden. Het programma is gebruikt voor het testen van GraphQL calls die in ontwikkeling waren, hiermee is er gevalideerd of query's of mutaties ook echt werken op basis van response van de API.

5.6.3 Prototype

Met het prototype is een deel van het systeem gerealiseerd. Met het prototype is het mogelijk om een vergelijking op te stellen van het verbruik van twee verschillende auto's. De rit waarmee vergeleken wordt kan geregistreerd worden door het invullen van een formulier of door gebruik te maken van een GPS tracker. Om de code overzichtelijk en eenvoudig te begrijpen te maken, zijn er opmerkingen in de code toegevoegd waar nodig. Hierin wordt bijvoorbeeld uitgelegd hoe de binnenkomende data van openRDW wordt verdeeld over de threads. In tabel 11 op de volgende pagina is een precieze lijst van de wel en niet afgeronde Kanban taken met betrekking tot de implementatie te vinden. Zie hoofdstuk 7.1.3 voor uitleg waarom sommige taken niet zijn afgerond en zie bijlage 10 voor de code van het prototype.

Tabel 11: Afgeronde Kanban taken voor prototype

Nr.	Taak	Afgerond
1.	De ontwikkelaar moet een ontwikkelstraat opzetten voor de mobiele applicatie op GitLab zodat er validatie is van kwaliteit en correctheid van code op de repository.	✓
2.	De ontwikkelaar moet een ontwikkelstraat opzetten voor de API zodat er garantie is voor kwaliteit en correctheid van code en zodat de code doorgepushed wordt naar een testomgeving.	✓
3.	De gebruiker moet kunnen inloggen waarbij er gebruik gemaakt wordt van tokens.	
4.	De gebruiker moet een overzicht van auto's kunnen ophalen waarbij ook een foto van de auto wordt getoond.	✓
5.	Er moet een koppeling naar openRDW gemaakt worden waarbij de data uit hun database naar de eigen database gehaald wordt.	✓
6.	Er moet een koppeling naar AutoDisk gemaakt worden waarbij de data uit hun database naar de eigen database gehaald wordt.	
7.	De data van openRDW en AutoDisk moeten gecombineerd worden zodat het als 1 object opgehaald kan worden.	
8.	De gebruiker moet een gedetailleerd overzicht van autogegevens van verschillende auto's kunnen zien door op een auto uit het overzicht te klikken.	✓
9.	De gebruiker moet handmatig ritten kunnen registreren door middel van een formulier in te vullen. Hierbij moet de interval, afstand, ritsoort en intervalsoort worden ingevuld.	✓
10.	De gebruiker moet ritten kunnen registreren door het gebruik van gps-gegevens.	✓
11.	De gebruiker moet op basis van gps-gegevens de gesimuleerde actieradius kunnen zien.	✓
12.	De gebruiker moet op basis van gps-gegevens het rijgedrag kunnen bepalen.	
13.	De gebruiker moet zijn rijgedrag kunnen bepalen in het realtime overzicht.	
14.	De gebruiker moet een overzicht kunnen krijgen van het benodigde aantal tank- en oplaadbeurten op basis van de ingevoerde parameters bij een rit.	✓
15.	De gebruiker moet de hoofdaansluiting (meterkast) van de auto kunnen selecteren om de maximale laadsnelheid te kunnen bepalen.	
16.	De gebruiker moet een overzicht kunnen ophalen van de laadstations, afhankelijk van de hoofdaansluiting in de meterkast en de gekozen auto.	
17.	De gebruiker moet extra bijbehorende producten kunnen selecteren op basis van de gekozen auto en meterkast.	
18.	Het moet mogelijk zijn om meerdere producten te selecteren van een auto.	
29.	Het moet mogelijk zijn om een offerte op te vragen van de installatie van de geselecteerde laadpaal.	
20.	De gebruiker moet een mobiliteitsprofiel kunnen maken en kiezen. Dit profiel moet bestaan uit een geselecteerde auto	✓
21.	De ontwikkelaar moet een Docker-compose file maken voor het opzetten van de database.	✓

5.6.4 Testen

In de onderstaande hoofdstukken worden de verschillende soorten uitgevoerde testen beschreven en zal de code coverage van de front- en backend benoemd worden. Testen zijn geschreven en uitgevoerd na de implementatie van een functie. Hierbij is er rekening gehouden met dat de functionaliteit een requirement zou afdekken. Zoals eerder benoemd is de realisatie uitgevoerd op basis van Behaviour Driven Development en Acceptance Driven Development omdat het uiteindelijke resultaat van de requirements nog niet volledig duidelijk was en er veel onzekerheid voor de implementatie was door de grootte leercurve voor de afstudeerder. Zie tabel 12 en 13 voor een overzicht van de behaalde requirements, zoals de vergelijkingsberekening. De geschreven tests zijn te vinden in de map ‘tests’ van zowel de front- als backend.

Unit testen

Voor elke functionaliteit die gerealiseerd is zijn er unit testen geschreven voor zowel de front- als de backend applicatie. Voor de front-end gaan de testen vooral om het verifiëren of de berekeningen voor de rit vergelijking correct zijn en of de GraphQL calls correct worden uitgevoerd. Op de backend zijn unit testen geschreven voor het valideren van het verwerken van binnenkomende GraphQL calls, het aanpassen van data en de functies en classes die nodig zijn voor het importeer- en updateproces van de openRDW koppeling.

Widget(Gui) testen

Voor de front-end zijn er voor alle schermen widget testen geschreven. Hiermee wordt er getest of de front-end correct functioneert. De widget testen zijn in de vorm van integratie testen geschreven, hierdoor is de app van top tot teen doorlopen met test cases.

Database testen

Om de database calls te valideren zijn er databasetesten geschreven voor database operation die wordt toegepast in de backend. Dit is in plaats van met een remote database, getest met een in-memory database zodat de testen op elke omgeving uitgevoerd kunnen worden.

Exploratory testen

Buiten de geschreven testen om is er ook handmatig getest. Dit is bijvoorbeeld gedaan door tijdens het ontwikkelen te testen of het resultaat van een functie overeenkomt met wat er verwacht werd of door te kijken of een knop het gewenste gedrag uitvoert.

Acceptatie testen

Tijdens het project zijn er een aantal demo’s geweest om te valideren of de opgeleverde iteratie voldeed aan de wensen van de opdrachtgever. Dit is in de vorm van formele presentaties en informeel aan het bureau laten zien gedaan.

Code coverage

Bij elke regel code die getest wordt zijn er een aantal regels code die worden uitgevoerd omdat de test deze aanroept. Het percentage code wat door de testen is gebruikt ten opzichte van de totale hoeveelheid code is het code coverage percentage en kan als indicatie voor de code kwaliteit worden gebruikt.

In de meeste gevallen geld: Hoe hoger de code coverage, hoe beter de kwaliteit. Dit is in de meeste gevallen zo omdat code ook overzichtelijk en makkelijk te begrijpen moet zijn. De code coverage van het prototype is verdeeld over de front- en backend applicatie.

- Code coverage front-end: 99,1%
- Code coverage backend: 90,0%

De code coverage van de front-end is bekend door middel van een Flutter functie die uitgevoerd wordt door het commando 'flutter test –coverage' uit te voeren. Hierdoor zullen alle aanwezige tests in de 'tests' map worden uitgevoerd, waarbij het aantal uitgevoerde regels code wordt bijgehouden. Het resultaat hiervan is de code coverage. De code coverage van de backend is verkregen door een functie in IntelliJ IDEA. Net als bij Flutter, heeft IntelliJ een functie om alle tests van een Java-applicatie uit te voeren en het code coverage percentage te berekenen.

Behaalde requirements

Op basis van geschreven testen zijn er een aantal requirements officieel behaald. Hier wordt officieel benoemd omdat er met behulp van de testen verzekerd wordt dat de functies ook echt werken. Zie tabel 12 en 13 voor een overzicht van de behaalde requirements.

Tabel 12: Behaalde requirements

Nr.	Requirements	Behaald
1.	De gebruiker moet kunnen inloggen.	
2.	De gebruiker moet een overzicht van auto's kunnen krijgen op basis van externe gegevens (open RDW en AutoDisk).	✓
3.	De gebruiker moet een auto uit een overzicht kunnen selecteren waarbij ook een foto van de auto getoond wordt.	✓
4.	De gebruiker moet handmatig ritten kunnen registreren.	✓
5.	De gebruiker moet via gps-gegevens ritten kunnen registreren.	✓
6.	Het moet mogelijk zijn om een realtime overzicht te krijgen van gesimuleerde gegevens op basis van gps-gegevens.	✓
6.1	Het moet mogelijk zijn om op basis van gps-gegevens de gesimuleerde actieradius te zien.	✓
6.2	Het moet mogelijk zijn om op basis van gps-gegevens het gesimuleerde verbruik te zien.	
6.3	Het moet mogelijk zijn om de huidige snelheid te zien gebaseerd op gps-gegevens	✓
6.4	Het moet mogelijk zijn om het rijgedrag te bepalen in het realtime overzicht.	
7.	De gebruiker moet in een formulier de gegevens van minimaal 1 rit kunnen invullen. Welke auto, interval van de rit, afstand, wat voor soort rit (vakantie, vrije tijd, etc.)	✓
8.	De gebruiker moet in het formulier de laadopties kunnen selecteren. (Ochtend, middag, dag, nacht)	✓
9.	De gebruiker moet in het formulier de rijstijl kunnen selecteren. (Eco, normaal, sport)	✓

Tabel 13: Behaalde requirements (vervolg)

Nr.	Requirements	Behaald
10.	De gebruiker moet een overzicht kunnen krijgen van het benodigde aantal tank- en oplaadbeurten op basis van de ingevoerde parameters in het formulier.	✓
11.	De gebruiker moet de hoofdaansluiting (meterkast) kunnen selecteren. (Om de maximale laadsnelheid te kunnen bepalen)	
12.	De gebruiker moet een mobiliteitsprofiel kunnen selecteren en aanmaken waar de rij auto, laadopties en rijgedrag in vastgesteld zijn.	✓
13.	De gebruiker moet een overzicht kunnen ophalen van de optimale, maximale en op de toekomst voorbereide laadstations. (Afhankelijk van de hoofdaansluiting in de meterkast en de gekozen auto)	
14.	De gebruiker moet visueel een laadstation kunnen selecteren op basis van resultaten van requirement 12.	
15.	Het moet mogelijk zijn om minimaal meerdere product te selecteren. (Onder producten valt: schouwing van de hoofdaansluiting, laadpalen, laadpassen)	
16.	Het moet mogelijk zijn om een overzicht te krijgen van geselecteerde producten op basis van requirement 14.	
17.	Het moet mogelijk zijn om het resultaat te downloaden van de ingevoerde gegevens en berekeningen van de rit en geselecteerde producten.	
18.	Het moet mogelijk zijn om een offerte op te vragen van de installatie van de geselecteerde laadpaal.	
19.	De app moet minimaal iOS 11 ondersteunen	✓
20.	De app moet minimaal Android 6 (Marshmallow) ondersteunen	✓
21	De apps moeten voldoen aan de richtlijnen van Apple en Google (inclusief de laatste privacy eisen)	✓

Alle requirements die in tabel 12 en 13 zijn afgevinkt in de ‘behaald’ kolom, worden als behaald beschouwd. Voor deze requirements zijn de nodige functionaliteiten gerealiseerd en zijn de bijbehorende testen geschreven om de functionaliteit te valideren. Voor de volgende requirements gelden andere criteria, maar worden toch als behaald beschouwd:

- Requirement 19: Er zijn technieken gebruikt (Flutter) die minimaal iOS 11 ondersteunen. Bovendien is de applicatie is op een device met iOS 11 getest.
- Requirement 20: Er zin technieken gebruikt (Flutter) die minimaal Android 6 ondersteunen. Bovendien is de applicatie op en device met Android 6 getest.
- Requirements 21: Tijdens het ontwikkelen is er rekening gehouden met de algemene (privacy)richtlijnen van Apple en Google. Momenteel wordt hier rekening mee gehouden door een schaalbaar design zodat alle device formaten ondersteund zouden moeten zijn. Ook wordt de gebruiker om toestemming gevraagd waar nodig, zoals bij het opvragen van locatie gegevens. Verder worden er geen onnodige permissies gevraagd. Wat privacy betreft is er momenteel nog niets wat er wordt opgeslagen op gebied van persoonlijke informatie, bovendien wordt er alleen openbare data gebruikt waardoor het systeem voldoet aan de privacyrichtlijnen.

6. Conclusie

Op basis van het onderzoek kan er geconcludeerd worden dat de volgende componenten nodig zijn voor het realiseren van een systeem wat gebruikers via een mobiele omgeving kan adviseren of een elektrische auto bij hen past:

- Een front-end die is opgebouwd met Flutter.
- Een backend die is opgebouwd met Java Spring Boot.
- De GraphQL API-structuur en taal voor gestructureerde communicatie tussen de front- en backend.
- Een MySQL database met een databasestructuur zoals weergegeven in figuur 2.

Met het prototype is er ook gevalideerd dat de componenten in de praktijk functioneel werken. Ook is met het prototype bewezen dat gebruikers via deze mobiele omgeving geadviseerd kunnen worden over elektrische auto's. In tabel 12 en 13 zijn de behaalde en niet behaalde requirements voor het prototype weergegeven. Volgens dit overzicht zijn onder andere de requirements voor het registreren van ritten en het vergelijken van het verbruik op basis van een rit behaald. Dit zijn de hoofdfuncties die nodig zijn voor het adviseren van een consument in de huidige context. Zie bijlage 10 voor het prototype.

Na het trekken van conclusies op de deelvragen en het realiseren van een prototype kan er een conclusie op de hoofdvraag getrokken worden. De conclusie is dat het doormiddel van de geselecteerde componenten mogelijk is om een systeem te maken wat advies kan geven of een elektrische- of hybride auto geschikt is voor de gebruiker. Dit is mogelijk door het registreren van een rit waarbij vervolgens het verbruik van twee verschillende auto's wordt vergeleken op basis van de rit afstand, het rijgedrag en de gewenste laadopties voor een elektrische auto. De validiteit van deze conclusie wordt ondersteunt door het uitgevoerde onderzoek naar de verschillende deelvragen en de realisatie van het prototype.

7. Discussie en aanbevelingen

In het hoofdstuk *discussie en aanbevelingen* zal er gereflecteerd worden op het onderzoek. Onder hoofdstuk 7.1 zal er besproken worden of de resultaten valide zijn. Overigens zal er besproken worden wat voor obstakels er tijdens het project naar voren kwamen waardoor het resultaat afwijkt van wat er gevraagd werd. Tot slot zal er in hoofdstuk 7.2 een verzameling van adviezen worden gegeven aansluitend op de conclusies.

7.1 Discussie

In de onderstaande hoofdstukken wordt er gereflecteerd op de validiteit van het onderzoek, vervolgens op de punten die anders zijn gelopen dan verwacht. Er wordt ingegaan op resultaat ten opzichte van de verwachting, waarom dit zo is gelopen en hoe dit in de toekomst anders aangepakt zou kunnen worden.

7.1.1 Validiteit

Voor dit onderzoek zijn diverse onderzoeksmethoden gebruikt om te onderzoeken wat de benodigdheden zijn voor het ontwikkelen van een mobiel platform om gebruikers te adviseren over het gebruik van elektrische- en hybride auto's. Alle literatuuronderzoeken zijn gebaseerd op feiten van officiële documentaties. De interviews zijn uitgevoerd met ervaren software engineers en de observaties op intern gecreëerde projecten. Door de aanwezigheid van het corona-virus is het niet mogelijk geweest om externe interviews te houden met belanghebbenden. Ondanks dit zijn er alleen interne interviews gehouden met een CTO en software engineers die kennis hebben over het project. Op basis de feiten van het literatuuronderzoek en de contextuele informatie van de veldonderzoeken kan er geconcludeerd worden dat de resultaten van deelvragen 1, 2 en 3 valide en betrouwbaar zijn. Dit wordt bovendien ook ondersteund door het praktijkonderzoek in de vorm van het gemaakt Proof of concept.

Voor deelvraag 4 is er ook een literatuuronderzoek uitgevoerd gebaseerd op feiten. Overigens is het resultaat hiervan voortgekomen uit een werkplaatsonderzoek waarbij er meerdere versies van een ontwerp zijn gemaakt op basis van iteraties. Op basis hiervan kan gesteld worden dat door het itererend ontwerpen van een databasestructuur de resultaten van dit onderzoek valide zijn.

7.1.2 AutoDisk

Er zou een tweede externe koppeling aanwezig moeten zijn in het systeem. Deze zou naar AutoDisk gaan en zou de data die binnen wordt gehaald vanaf openRDW moeten aanvullen met ontbrekende punten. Het is niet mogelijk geweest om dit te realiseren vanwege vertraging van Emodz die de toegang naar AutoDisk mogelijk zou maken. Als alternatief hiervoor worden vaste waardes ingeladen voor de ontbrekende waardes die opgehaald zouden worden vanaf AutoDisk. Deze waardes worden niet opgeslagen maar worden handmatig toegevoegd in de front-end wanneer hierom gevraagd wordt. Dit betekent dat er in het vervolg, wanneer de AutoDisk koppeling wel mogelijk het volgende gedaan dient te worden:

- De AutoDisk data toevoegen aan de database.
- Zorgen dat deze data ook per tijdseenheid wordt geüpdate.

Door het wegvallen van deze koppeling zijn er ook een aantal andere ‘nice to have’ functies niet geïmplementeerd in de mobiele applicatie. Een voorbeeld hiervan is het weergeven van een foto van de auto in het auto overzicht scherm. In de gewenste situatie zou er een URL in de AutoDisk data zitten met die leidt naar een afbeelding van de desbetreffende auto. Momenteel is dit vervangen met het weergeven van een afbeelding van het merk van de auto.

In de toekomst zou een soortgelijk probleem anders aangepakt kunnen worden door opzoek te gaan naar een andere partij die dezelfde functionaliteit als AutoDisk levert. De mogelijkheid hiervan hangt overigens af van financiële factoren en de zaken die lopen met partnerbedrijven.

7.1.3 Afwijkend eindresultaat

De verwachting van het eindresultaat is afwijkend van het uiteindelijke resultaat. Dit komt omdat er een aantal implementaties niet af zijn gekomen, deze staan hieronder weergegeven:

- AutoDisk koppeling
- Rijgedrag bepalen op basis van gps-gegevens
- Producten selecteren
- Meterkast selecteren
- Offerte opvragen
- Loginfunctie

De reden dat deze functies niet afgekomen zijn heeft te maken met een te kleine tijdsinschatting van de gerealiseerde taken. Ook zat er een grote leercurve om het project voor Flutter, Java Spring Boot en GraphQL, bovendien duurde het implementeren van sommige functies langer dan verwacht. Wel zijn de meeste niet afgeronde taken ‘nice to have’ en zijn alle ‘must’ taken afgerond. Met uitzondering van de AutoDisk koppeling, zie hoofdstuk 7.1.2 voor meer uitleg over het wegvallen van de AutoDisk koppeling.

7.2 Aanbevelingen

In het hoofdstuk Aanbevelingen zullen er punten worden besproken voor het vervolg van het project. Het zal hier vooral gaan over vervolgpunten over het praktijkdeel van het project.

7.2.1 Front-end

In de onderstaande hoofdstukken zal er besproken worden wat adviezen zijn voor de vervolgimplementatie van de front-end.

Techniek

Flutter is gebruikt is framework voor de ontwikkeling van het product. Halverwege het project is het nieuwe Flutter 2 uitgebracht, dit zou een verbeterde versie van Flutter zijn die ook backwards compatible is met een Flutter codebase. Advies is om te overwegen om bij de doorontwikkeling van het product over te stappen op Flutter 2 voor een verbeterde ondersteuning.

Een ander voordeel van het gebruik van Flutter 2 is dat het ook ondersteuning biedt voor diverse varianten webapplicaties (Single page, progressieve web apps, mobile web apps), macOS, Windows en Linux.

Security

In de productie omgeving wordt momenteel gebruik gemaakt van SSL om de data tijdens transport te encrypten. Maar om dit nog een niveau hoger te brengen is een andere mogelijkheid het implementeren van JSON-web tokens die tijdelijk bruikbaar zijn en opgevraagd dienen te worden in de backend. Hierdoor zou iemand zonder token niet gebruik kunnen maken van de backend. Als laatste security optie zou er een loginfunctie toegevoegd kunnen worden die een gebruikers unieke token heeft, dit zou hetzelfde effect opleveren als een JSON-web token.

Performance/ datastreaming

De performance van Flutter is stabiel. Een kritiek punt waar wel naar gekeken dient te worden is het ophalen van data. Bij het ophalen van veel data kan de app vertraging oplopen. Dit zou opgelost kunnen worden door de implementatie van ‘lazy loading’, waarbij componenten pas worden geladen als dit nodig is. Een andere oplossing zou zijn om een stream te implementeren zodat data in secties wordt opgehaald. Dit zou de performance in theorie ook minder moeten belemmeren.

Overigens zou de combinatie van deze 2 adviezen een nog beter resultaat opleveren omdat er dan efficiënt gewerkt wordt op het binnenhalen van data en het verwerken van data in visuele componenten.

7.2.2 Backend

In de onderstaande hoofdstukken zal er besproken worden wat adviezen zijn voor de vervolgimplementatie van de backend.

Security

In de productieomgeving is er ook een SSL, overigens draaien de database en backend beide op dezelfde machine waardoor data niet onderschept kan worden. Als extra security kan er bijvoorbeeld persoonlijke data zoals wachtwoorden gehashed worden voor ze verstuurd en opgeslagen worden in de database.

Voor de functies voor het activeren en stoppen van de openRDW data import zou er authenticatie toegevoegd kunnen worden, zodat alleen bepaalde gebruikers hierbij zouden kunnen.

Datastreaming

De API haalt en verstuurt veel data. Momenteel is dit in de openRDW koppeling zo efficiënt mogelijk gemaakt door het gebruik van threads. Maar als de front-end veel data dient op te halen kan dit wat langer duren door de grote hoeveelheid data. Dit zou opgelost kunnen worden door het implementeren van datastreaming zodat data in etappes wordt verstuurd.

Deze techniek zou zowel op front- naar backend communicatie als backend naar database communicatie kunnen worden toegepast.

Auto zoek functie

De auto zoekfunctie is momenteel beperkt tot het zoeken op kenteken en het zoeken op de combinatie merk + model. Dit zou een groot verschil maken als het uitgebreid en verbeterd zou worden met flexibelere zoekopdrachten. Een mogelijke oplossing hiervoor is het gebruik van de software Elasticsearch. Dit is een opensource softwarepakket gemaakt om een zoekmachine mee te bouwen en is gefocust op snelheid en schaalbaarheid. Dit zou dus ideaal zijn om in de toekomst te gebruiken voor de auto zoek functie.

Implementatie vernieuwen

Java Spring Boot brengt om de zoveel tijd nieuwe updates uit voor de verbetering van al bestaande functies en classes. In de toekomst zou het goed zijn om te kijken of er een aantal functies en classes vervangen kunnen worden door nieuwere versies ervan om de performance en efficiëntie van de applicatie te verhogen. Tot slot is het verstandig om regelmatig dependencies te updaten om bij te blijven met security fixes.

8. Literatuurlijst

Apple. (z.d.). *Swift - Apple (NL)*. Apple (Nederland). https://www.apple.com/nl/swift/

Apple. (2013, 23 april). *Introduction*. Copyright 2018 Apple Inc. All Rights Reserved. https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ObjectiveC/Introduction/introObjectiveC.html#//apple_ref/doc/uid/TP30001163

B. (2020a, augustus 30). *Getting Started with GraphQL and Spring Boot*. Baeldung. https://www.baeldung.com/spring-graphql

Barrera, D. B. (2019, 13 augustus). *Building a simple application with Flutter and GraphQL*. Medium. https://medium.com/flutter-community/building-a-simple-application-with-flutter-and-graphql-5786764df102

Baseflow.com. (2021, 5 februari). *geolocator | Flutter Package*. Dart Packages. https://pub.dev/packages/geolocator

Concise Software. (2019, 26 augustus). *What is Flutter? Here is everything you should know*. Medium. https://medium.com/@concisesoftware/what-is-flutter-here-is-everything-you-should-know-faed3836253f

DEVATHON TEAM. (2020, 26 augustus). *Flutter vs Xamarin: A detailed comparison*. Devathon Blog. https://devathon.com/blog/flutter-vs-or-and-xamarin/#:%7E:text=The%20user%20experience%20provided%20by%20Xamarin%20vs%20Flutter&text=Xamarin%20is%20a%20more%20mature,Xamarin%20easier%20for%20UI%20development.

Flutter. (z.d.). *Build and release an iOS app*. https://flutter.dev/docs/deployment/ios

GraphQL Foundation. (z.d.). *GraphQL | A query language for your API*. GraphQL. Geraadpleegd op 10 maart 2021, van https://graphql.org/

I. (2020b, juli 28). *Flutter vs Native vs React-Native: Examining performance*. Medium. https://medium.com/swlh/flutter-vs-native-vs-react-native-examining-performance-31338f081980

IBM. (z.d.). *Voorbeeld van een SOAP message* [Afbeelding]. AlexSoft. https://content.altexsoft.com/media/2019/08/word-image-9.png

inVerita. (2020, 28 juli). *Flutter vs Native vs React-Native: Examining performance*. Medium. https://medium.com/swlh/flutter-vs-native-vs-react-native-examining-performance-31338f081980#:%7E:text=Native%20is%202%20times%20faster,6%20times%20slower%20than%20native.

Kaczorowski, M. (2021, 22 februari). *Java vs. Kotlin: Which One To Pick While Building An Android App?* Java vs. Kotlin. https://www.ideamotive.co/blog/java-vs-kotlin-which-one-to-pick-while-building-an-android-app

Lucid. (z.d.). *Tutorial database structuur en -ontwerp*. Lucidchart. Geraadpleegd op 23 maart 2021, van https://www.lucidchart.com/pages/nl/tutorial-database-structuur-en-ontwerp

MSc., P. J. (2020, 11 oktober). *Big Data: definitie, voordelen en voorbeelden (4x)*. Peter Joosten. https://www.peterjoosten.net/big-data/#:%7E:text=Big%20data%20wordt%20vooral%20gebruikt,verband'%20of%20'relatie'.

P, S. (2020, 28 mei). *When to use gRPC over REST - Sankar P*. Medium. https://medium.com/@sankar.p/how-grpc-convinced-me-to-chose-it-over-rest-30408bf42794

Populi, N. (z.d.). *5 Powerful Alternatives to REST APIs (2018 Update)*. LEAPGRAPH. Geraadpleegd op 9 maart 2021, van https://leapgraph.com/rest-api-alternatives

Redactie: Pantarein. (2019, 6 juni). *Wat is B Corp?* The Shift: Wat is B Corp? https://theshift.be/nl/inspiratie/wat-is-b-corp#:%7E:text=B%20Corp%20staat%20voor%20Benefit,ecologische%20prestaties%2C%20verantwoordelijkheid%20en%20transparantie.

Smith, C. (2020, 27 februari). *Advantages and Disadvantages of GraphQL*. Stable Kernel. https://stablekernel.com/article/advantages-and-disadvantages-of-graphql/

Socrata. (z.d.). *System Fields | Socrata*. SODA Docs. Geraadpleegd op 3 mei 2021, van https://dev.socrata.com/docs/system-fields.html

Statcounter. (z.d.). *Mobile Operating System Market Share Netherlands*. StatCounter Global Stats. https://gs.statcounter.com/os-market-share/mobile/netherlands

The gRPC Authors & The Linux Foundation®. (2020, 17 december). *Introduction to*. GRPC. https://grpc.io/docs/what-is-grpc/introduction/

Uptrends. (z.d.). *Wat is REST API? | Uptrends*. Uptrends.nl. Geraadpleegd op 10 maart 2021, van https://www.uptrends.nl/wat-is/rest-api

Wodehouse, C. (2015, 7 augustus). *Swift vs. Objective-C: A Look at iOS Programming Languages | Upwork*. https://www.upwork.com/. https://www.upwork.com/resources/swift-vs-objective-c-a-look-at-ios-programming-languages#:%7E:text=Swift%20is%20easier%20to%20read,has%20a%20more%20clunky%20syntax.&text=Also%2C%20Swift%20requires%20less%20code,interpolation%2C%20without%20placeholders%20or%20tokens.

Bijlagen

- Bijlage 1: Interview deelvraag 3
- Bijlage 2: Onderzoeksdocument
- Bijlage 3: Database diagram
- Bijlage 4: AutoDisk datavoorbeeld
- Bijlage 5: Mockups versie 1
- Bijlage 6: Mockups versie 6
- Bijlage 7: Functionele documentatie
- Bijlage 8: Technische documentatie
- Bijlage 9: App visualisatie