On Board Diagnostics

Drive Style Analyst applicatie

Datum: juni 2011

Auteurs: Jeroen van Schelven
Erwin Smeets

AFSTUDEERVERSLAG VOOR FONTYS HOGESCHOOL ICT

Gegevens studenten:

Naam + voorletters student	J. van Schelven
Studentnummer	2123151
Opleiding	Technische Informatica
Afstudeerperiode	07-02-2011 t/m 01-07-2011
Naam + voorletters student	E.A. Smeets
Studentnummer	2122919
Opleiding	Technische Informatica
Afstudeerperiode	07-02-2011 t/m 01-07-2011

Gegevens Bedrijf:

Naam bedrijf/instelling:	TASS technology solutions
Plaats	Eindhoven

Gegevens bedrijfbegeleider:

Naam	Tiny Henst
Functie	People manager

Gegevens technisch begeleider:

Naam	Andre Schurer
Functie	Programmeur

Gegevens verslag:

Titel afstudeerverslag met subtitel	On Board Diagnostics “Drive Style Analyst applicatie"
Datum uitgifte afstudeerverslag	9-6-2011

Getekend voor gezien door bedrijfsbegeleider:

Datum:

De bedrijfsbegeleider,

Voorwoord

Dit verslag is geschreven naar aanleiding van de afstudeerstage die plaatsvindt in het vierde en daarbij laatste leerjaar van de opleiding Technische Informatica bij Fontys Hogescholen te Eindhoven. In de periode van februari 2011 tot en met juni 2011 hebben we 100 dagen stage gelopen bij TASS technology solutions te Eindhoven. Tijdens deze periode hebben we onderzoek gedaan naar "On Board Diagnostics" in auto's en naar aanleiding van dat onderzoek een applicatie ontwikkeld die automobilisten inzicht geeft in hun rijstijl.

In dit verslag zullen we verdere toelichting geven over de aanleiding van de opdracht, onze werkwijze en de gemaakte beslissingen. Ook het onderzoek en het gehele proces van ontwikkeling van het begin tot aan het opgeleverde eindproduct worden in deze scriptie naar voren gebracht.

We willen graag onze bedrijfsbegeleider, Tiny Henst en onze technisch begeleider, Andre Schurer bedanken voor hun betrokkenheid, technische ondersteuning en begeleidingsgesprekken. Ook willen we onze dank uiten naar Jos Verhagen, onze docent begeleider, voor het bewaken van het afstudeer proces en zijn waardevolle feedback op het afstudeerverslag.

Aangezien de stage uitgevoerd is in duo vorm willen we als laatst het voorwoord gebruiken om aan te geven door wie de hoofdstukken die volgen geschreven zijn.

1.	Inleiding:	Jeroen van Schelven & Erwin Smeets
2.	Het bedrijf:	Erwin Smeets
3.	De opdracht:	Jeroen van Schelven
4.	Onderzoek:	Erwin Smeets
5.	Requirements:	Erwin Smeets
6.	Ontwerpen:	Jeroen van Schelven
7.	Implementeren:	Jeroen van Schelven
8.	Conclusie:	Jeroen van Schelven & Erwin Smeets

Jeroen van Schelven
Erwin Smeets

Eindhoven, Nederland, 09-06-2011

Inhoudsopgave

Figurenlijst

Tabellenlijst

Samenvatting

TASS technology solutions is een dienstverlener op het gebied van technische en embedded software. TASS is oorspronkelijk een onderdeel van Philips en sinds 2007 een zelfstandige organisatie. Het merendeel van de activiteiten bestaat uit dienstverlening en dan specifiek gericht op de volgende gebieden: Healthcare & Cure, Mechanics & Control, Consumer Lifestyle, Safety & Security en Mobility & Automotive.

Een van de interessegebieden is daarom “On Board Diagnostics”(OBD). Deze “diagnostische stekker” bevindt zich sinds 2003 standaard in alle auto’s en maakt het mogelijk om diverse live en opgeslagen gegevens uit te lezen.

Tijdens de uitvoering van het project was de eerste focus het onderzoeken van de huidige status en eventuele standaard van OBD. Dit onderzoek is gedaan om te bepalen of de lijst met wensen van TASS haalbaar was. Uit de resultaten van dit onderzoek is gebleken dat het mogelijk is om met een omweg het merendeel van deze wensen te realiseren.

Om te kijken hoe dit het best vertaald kon worden naar een Android applicatie is er onderzoek gedaan. Na dit onderzoek is het concept van de “Drive Style Analyst” applicatie opgesteld. Dit concept is in overleg met de bedrijfsbegeleider verder uitgewerkt tot een requirements document waarin de gehele functionaliteit van deze Android applicatie beschreven staat. Deze applicatie heeft als doel om automobilisten te helpen zuiniger te rijden.

Hoe gaat deze applicatie de automobilist hiermee helpen? Aan de hand van de gegevens die via OBD worden uitgelezen wordt door middel van een aantal algoritmes de rijstijl beoordeeld. Omdat niet alle auto’s dezelfde eigenschappen hebben zijn deze algoritmes geschikt gemaakt voor verschillende auto types door normwaardes te gebruiken. Omdat niet vooraf bepaald kan worden wat deze normwaardes precies zullen zijn is hiervoor een zelf lerend systeem ontwikkeld.

Door alle ritgegevens online op te slaan heeft dit zelf lerend systeem toegang tot alle relevante informatie om zo tot nieuwe normwaardes te komen. Op deze manier kunnen voor alle verschillende auto’s dezelfde algoritmes gebruikt worden om de rijstijl te beoordelen.

Met de applicatie is het mogelijk om de zuinigheid en milieuvriendelijkheid van de rijstijl van automobilisten te bepalen. Door de scores en extra tips te geven kan de automobilist vervolgens zijn rijstijl verbeteren met als resultaat lagere brandstof kosten. Dit systeem is getest door minstens 10 TASS’ers en is hierdoor steeds preciezer geworden. Het idee is dus uitgewerkt tot een werkend systeem!

Summary

TASS technology solutions is a service oriented business, focused on technical and embedded software. TASS has been on the cutting edge of technology and development for over 30 years. Originally part of Philips as the internal software house for embedded and technical consulting, TASS became an independently operating company in 2007 as part of the international Dutch concern Total Specific Solutions. TASS mainly operates in the domains of Healthcare & Cure, Mechanics & Control, Consumer Lifestyle, Security and Mobility & Automotive.

One of the missions of TASS is to create more own products and do more in-house projects, especially in the direction of Automotive. This is why this project about On Board Diagnostics (OBD) is of extra interest. This diagnostics ability in the modern car has been available and mandatory for all cars produced since 2003 and opens a world of possibilities to get live information and other car specific data.

The first focus during this project was to research the current state of OBD and what kinds of standards were used with this technology. This was done to ensure the demands of TASS could be implemented with the use of OBD. With the results of this research there were several ideas for an Android demo application. Of which the Drive Style Analyst was the most promising.

This concept then was further discussed with the company supervisor and translated to a software requirements document in which all the details and functionalities of the application were included. This application is meant to inform car drivers of their driving style and to give advice on how to improve the way they drive. This is very interesting because certain driving styles can have very negative effect on fuel consumption as well as environmental impact.

How is this application going to help the driver to improve their driving style and save money? With the use of OBD to read all important vehicle information the application can generate a set of scores using sophisticated algorithms. Because all car models have different characteristics the algorithms are optimized to use standards generated for the specific car type. Because not all these standard values could be calculated in advance, a self learning system has been created.

This system uses all the drive data gathered from all the cars in a category and calculates the best possible default value. This way all different car types can use the same algorithm to assess the driving style.

With this implemented application it’s possible to determine how economic and environmentally friendly the driving style is. The system gives scores and advice to the drivers, which enables them to adapt their driving styles to reduce fuel consumption. The self learning system has been tested out by at least 10 TASS employees and has proven to be more accurate every time. At the end the idea has become a fully functional application!

Verklarende woordenlijst

API	Application Programming Interface. Een verzameling definities op basis waarvan een applicatie kan communiceren met een andere applicatie of onderdeel hiervan.
App	Korte aanduiding voor een applicatie, voornamelijk gebruikt op mobiele platformen.
DSA	Drive Style Analyst, de ontwikkelde Android applicatie.
DTC	Diagnostic Trouble Codes. Foutcodes die door het OBD systeem worden opgeslagen.
EOBD	Europese versie van de OBDII standaard.
HTTP	Hypertext Transfer Protocol. Het protocol wat het meest gebruikt wordt bij webcommunicatie.
HTTP GET	Een implementatie van het HTTP protocol waarbij extra gegevens worden meegestuurd via de URL.
HTTP POST	Een implementatie van het HTTP protocol waarbij gegevens als berichtinhoud worden meegestuurd.
JSON	JavaScript Object Notation, een datastructuur. Een deelverzameling van de programmeertaal JavaScript. Het wordt gebruikt voor de uitwisseling van datastructuren, voornamelijk in webapplicaties.
OBD	On Board Diagnostics. Een term die verwijst naar het voertuig managementsysteem en de interface voor het uitlezen van informatie over verschillende delen van het voertuig.
OBDII	De huidige standaard en opvolger van OBD.
OBDKey	Specifieke hardware die een Bluetooth verbinding mogelijk maakt met de OBD poort.
XML	Extensible Markup Language, een datastructuur waarmee het mogelijk is gestructureerde gegevens weer te geven in platte tekst.

1. Inleiding

Het totaal aantal personenauto's op de Nederlandse wegen nam in 2010 toe met 198.579. Het wagenpark van Nederland telde daarmee op 31 december 2010 8.002.579 personenauto's. Dat betekent dat ons onderzoek naar "On Board Diagnostics" (OBD) bij auto's en de applicatie die met behulp van dit onderzoek ontwikkeld is dus ruim 8 miljoen potentiële gebruikers telt.

Figuur 1: OBD connector

TASS technology solutions is een dienstverlener op het gebied van embedded software. Een van de pijlers van TASS is Mobility & Automotive. Omdat OBD een interessante technologie is wil TASS weten wat op dit moment de mogelijkheden van deze "diagnostische stekker" zijn. Op basis hiervan is ons deze afstudeeropdracht voorgelegd. Hierbij is er een onderzoek naar de huidige staat van OBD gedaan waaruit een beter beeld is gekomen over de mogelijkheden hiervan. Op basis van dit onderzoek is een voorstel geschreven met een aantal mogelijke ideeën die gebruik maken van de "diagnostische stekker". Na overleg met de klant is hieruit een opzet voor een toepassing gekomen die uitgewerkt is tot een volledige applicatie.

Tijdens dit verslag wordt in verschillende hoofdstukken het volledige proces uitgelegd. We beginnen met informatie over het bedrijf, waarna de oorspronkelijke opdracht wordt beschreven. Hierna gaan we verder in op het uitgevoerde onderzoek naar "On Board Diagnostics" en Android. Ook wordt er ingegaan op de mogelijke toepassingen en ideeën die uit dit onderzoek zijn gekomen.

Na het onderzoek gaan we in het hoofdstuk Requirements verder in op het definitieve idee en hoe dit wordt uitgewerkt tot een volledige applicatie, ook worden er motivaties gegeven waarom voor bepaalde functionaliteiten gekozen is. Om de requirements te vertalen naar een daadwerkelijk applicatie, is hiervoor een ontwerp gemaakt. Dit ontwerp wordt besproken in het hoofdstuk Ontwerp. Hierna gaan we in op de problemen die we tegen zijn gekomen tijdens het implementeren van het ontwerp ook wordt er in dit hoofdstuk toegelicht wat de verschillen zijn. Tot slot eindigen we dit verslag met een conclusie en een tweetal evaluaties.

Figuur 2: OBD app

2. Het bedrijf

TASS technology solutions is een dienstverlener op het gebied van embedded - en technische software. Een organisatie die al meer dan 30 jaar voorop loopt in de ontwikkeling van software voor technisch hoogwaardige producten. Eerst als onderdeel van Philips en sinds 2007 een succesvolle zelfstandige organisatie. Dit maakt TASS tot een dynamische en betrouwbare partner, die snel kan schakelen en tijdig weet in te springen op ontwikkelingen in de markt. Dankzij deze missie staat TASS momenteel in de top 3 dienstverlener voor technische - en embedded software.

TASS is een onderdeel van Total Specific Solutions (TSS). TSS bestaat uit negen onafhankelijke ICT-ondernemingen, waarbij elke onderneming zijn eigen marktgebied voor ogen heeft.

Het merendeel van de activiteiten bestaat uit dienstverlening en dan specifiek gericht op de volgende gebieden: Mobility & Automotive, Healthcare & Cure, Mechanics & Control, Consumer Lifestyle en Safety & Security.

Momenteel heeft TASS een viertal vestigingen, namelijk in Apeldoorn, Eindhoven, Leuven (BE) en Gent (BE). Bij TASS werken ruim 250 software professionals, waarvan het merendeel gedetacheerd is bij een van de vele klanten. Tijdens de afstudeerperiode hebben wij gewerkt in de vestiging te Eindhoven.

Figuur 3: TASS hoofdkantoor

3. De opdracht

3.1 Aanleiding

Bij TASS worden altijd nieuwe technologieën en de bruikbaarheid hiervan onderzocht. Met Automotive als een van de pijlers van TASS valt OBD hier ook onder. Er is geen duidelijke probleemstelling voor deze opdracht maar eerder een vraag naar kennis. Het is voor TASS interessant om te investeren in kennis van nieuwe en bestaande technologieën om in de toekomst hierop verder te ontwikkelen.

3.2 Doelstelling

Om te kijken wat OBD voor TASS kan betekenen, is de opdracht als volgt geformuleerd:

Specificeer, ontwerp en bouw een systeem waarmee we data van de OBD poort kunnen uitlezen en analyseren.

Op basis van deze opdracht zijn de volgende doelstellingen opgesteld:

Via de On Board diagnostics (OBD) poort in auto's is het mogelijk om allerlei informatie over de motor, de aandrijving e.d. op te vragen. Het doel van dit project is het onderzoeken welke informatie beschikbaar is via de OBD standaard waarna met deze gegevens een applicatie wordt bedacht, ontworpen en gerealiseerd voor Android. Deze applicatie analyseert en beheert deze gegevens.

3.3 Werkwijze

Tijdens dit project is er extra aandacht besteed aan het professioneel uitvoeren van de verschillende aspecten. Omdat er na het OBD onderzoek genoeg gegevens bekend waren om een uitgebreid requirements document en een gestructureerd ontwerp te maken is er voor het V model gekozen. Hierbij is het mogelijk om tijdens het ontwerpen met alle aspecten van de applicatie rekening te houden. Om tijdens dit proces tot de juiste keuzes te komen, zijn er een aantal deelonderzoeken naar Android gedaan. Bij het specificeren van de opdracht is er continue contact geweest met de klant om de eisen/wensen zo goed mogelijk te documenteren. Voor details over de werkwijze en de manier van contact met de klant zie bijlage A: Project Management Plan (PMP).

3.4 Randvoorwaarden

De belangrijkste randvoorwaarde die de klant heeft aangegeven is dat de applicatie ontwikkeld moet worden voor Android. Binnen TASS is er ruime kennis aanwezig van Android en de benodigde test hardware is hiervoor al beschikbaar. De overigen randvoorwaarden van dit project staan beschreven in bijlage A: Project Management Plan (PMP).

Figuur 4: Android

4. Onderzoek

Voor dit project zijn twee onderzoeken uitgevoerd, hierbij gaat het over OBD en Android. Van beide technieken is een minimale voorkennis aanwezig. Het doel van het Android onderzoek was om bekend te worden met de mogelijkheden hiervan. Om het OBD onderzoek gestructureerd te kunnen uitvoeren, is gekozen voor de aanpak via onderzoeksvragen. Hieronder staan de onderzoeksvragen voor het OBD onderzoek.

Figuur 5: OBDII poort

Welke gegevens zijn er beschikbaar via de OBD poort?

- Welke van deze gegevens behoren tot een standaard?
- Wat zijn de specifieke features geïmplementeerd in OBD per merk/model?
- Welke toepassingen zijn er mogelijk met de beschikbare gegevens?

Als hypothese wordt verwacht dat het resultaat op de onderzoeksvragen een groot vertrouwen geeft en dat er voldoende OBD gegevens beschikbaar zijn om hiermee een uitdagende toepassing te ontwerpen en te realiseren.

4.1 On Board Diagnotics

Aan het begin van het project was de kennis over OBD minimaal, om deze kennis uit te breiden is er als eerste onderzocht wat OBD precies inhoudt.

Figuur 6: OBD connector

Na onderzoek blijkt dat OBD in de jaren 80 is geïntroduceerd. OBD is ontwikkeld voor de autotechniek, hier wordt de OBD gebruikt om diagnoses te stellen bij een onderhoudsbeurt of bij defecten van een auto. Dankzij de gestandaardiseerde hardwareconnector is het mogelijk om bij elke auto via dezelfde hardware interface diagnoses op te stellen.

Vanaf 1996 is er in Amerika de OBDII geïmplementeerd. Deze opvolger van OBD heeft vele voordelen in zowel mogelijkheden als standaardisatie. Naast de hogere overdrachtssnelheid van OBDII, zijn de basisonderdelen van de OBD diagnoses gestandaardiseerd. Waar bij OBD per merk/model andere codes bestonden om gegevens uit te lezen, is deze bij OBDII deels gestandaardiseerd. Denk hierbij aan standaard onderdelen zoals de snelheid en het toerental. Hiervoor is een protocol ontwikkeld waaraan de autofabrikanten zich moeten houden. Mochten ze sensorgegevens beschikbaar willen maken dan moet dat volgens de standaard.

Het protocol wat gebruikt wordt bij de standaard van OBD is onderverdeeld in verschillende "modes". Hieronder worden de meeste gebruikte modes toegelicht:

- **Mode 1, standaard sensor informatie:** met deze mode is het mogelijk om sensorwaardes zoals snelheid, toerental en motor temperatuur uit te lezen.
- **Mode 2, freeze frame van de foutcodes:** Bij deze mode is het mogelijk om snapshots van alle sensor waardes op te vragen op het moment dat een fout opgetreden is.
- **Mode 3, opvragen foutcodes:** deze mode wordt gebruikt om alle foutcodes op te vragen die opgeslagen zijn in de auto.
- **Mode 4, verwijderen foutcodes:** met deze mode kunnen foutcodes verwijderd worden nadat deze door een monteur verholpen zijn.
- **Mode 9, basis voertuig informatie:** deze mode wordt gebruikt om voertuig informatie zoals het Vehicle Identification Number (VIN) uit te lezen.

Sinds 2001 voor benzine en sinds 2003 voor diesel voertuigen bestaat er een Europese variant van de OBDII standaard, namelijk Europe On Board Diagnostics (EOBD). De verschillen met OBDII zijn minimaal, deze liggen voornamelijk op juridisch gebied, daarom zal hierna tussen OBDII en EOBD geen onderscheid meer gemaakt worden.

Doordat de oorsprong van OBD uit de autotechniek komt, ging de documentatie hiervan veelal diep in op de autotechniek. Aangezien hierover een minimale kennis aanwezig was, is er tijdens het onderzoek een kleine uitstap gemaakt naar de autotechniek. Hiermee was het mogelijk om technische documentatie van OBD beter te begrijpen en gerichter onderzoek te doen naar de bruikbaarheid van de verschillende sensoren.

4.2 On Board Diagnostics hardware keuze

Om de OBD poort uit te lezen is er gekeken naar een geschikt hardwareapparaat. Hierbij zijn meerdere standaard hardwareoplossingen gevonden. Bij het zoeken naar hardwareoplossingen zijn een aantal criteria gebruikt waarvan hieronder een overzicht is weergegeven.

Tabel 1 OBD hardware

Naam	Prijs	SDK	USB	Bluetooth	Wi-Fi
OBDLink (Wi-Fi, USB)	$189,90	Gratis C SDK	X	-	X
OBDLink (BT, USB)	$149,90	Gratis C SDK	X	X	-
Kiwi Wi-Fi	$149,99	SDK voor $30	-	-	X
OBDKey Wi-Fi	€245,71	Gratis C# SDK en contact met ontwikkelaar	-	-	X
OBDKey Bluetooth	**€124,27**	**Gratis C# SDK en contact met ontwikkelaar**	**-**	**X**	**-**
OT-2 Wi-Fi	$179,00	Gratis SDK	-	-	X

Aangezien een draadloze oplossing voorkeur kreeg boven een fysieke oplossing. Vielen de USB oplossingen direct af.

Omdat bij een Wi-Fi oplossing de internetverbinding van de telefoon niet gebruikt kan worden en deze nodig is voor onderdelen van de applicatie, viel deze keuze ook af. Dit laat alleen de apparaten met Bluetooth over. Het ging hier om twee mogelijke hardware apparaten namelijk de OBDLink (BT, USB) en de OBDKey Bluetooth. Vanwege het bestaande contact met de ontwikkelaar en met de beschikbare C# SDK is er uiteindelijk gekozen voor de OBDKey Bluetooth.

4.3 Live testen

De eerste stap was om daadwerkelijk te bekijken wat er met de OBDKey mogelijk was. Op basis van de lijst met mogelijke sensoren is hiervoor een testapplicatie ontwikkeld in C# die de sensoren uit kon lezen. Hierbij is bij vijf verschillende automodellen geprobeerd elke sensor uit te lezen. In totaal zijn er 83 mogelijke sensoren die uitgelezen kunnen worden. In de praktijk bleek dat de beschikbare hoeveelheid sensoren veel lager lag dan verwacht. De uitkomst van deze tests zijn hieronder in een tabel weergegeven.

Tabel 2: Onderzoek auto's

Model	Beschikbare sensoren
Peugeot 207	14
Citroen C3	13
Toyota Auris	29
Fiat Punto	17
Volkswagen Passat	13

Figuur 7: Toyota Auris

Zoals zichtbaar in bovenstaande tabel is het minimaal beschikbare aantal sensoren rond de 13 á 14, waarbij de Toyota Auris positief uitschoot naar 29 aanwezige sensoren. Naast het sobere aantal beschikbare sensoren kwam uit de test ook naar voren dat de leessnelheid van de OBDkey rond de 6 á 7 lezingen per seconden bedraagt.

Op basis van de leessnelheid is er extra onderzoek gedaan naar de mogelijkheden om deze te versnellen. Hierbij is een oplossing gevonden om per lezing twee sensoren uit te lezen. Hiermee viel de leessnelheid terug naar 5 lezingen per seconden maar zorgt voor een totaal van 10 sensorwaardes per seconde, een optimalisatie van meer dan 50%.

4.4 Android

Bij het onderzoek naar Android was het de bedoeling om bekend te raken met de architectuur en specifieke eigenschappen. Het volgen van de "Hello World" tutorial samen met andere tutorials die worden aangeraden door de Google Android Developer's Guide heeft hier een grote rol in gespeeld. Nadat deze tutorials succesvol waren afgerond is er gekeken welke onderdelen van Android waarschijnlijk gebruikt zouden gaan worden in de te ontwikkelen applicatie. Denk hierbij aan database management, de Bluetooth communicatie en het gebruik van custom listviews. Een custom listview is een speciale lijst in de user interface waarmee het mogelijk is om door de ontwikkelaar zelf te bepalen waaruit een onderdeel bestaat.

4.4.1 Activity lifecycle

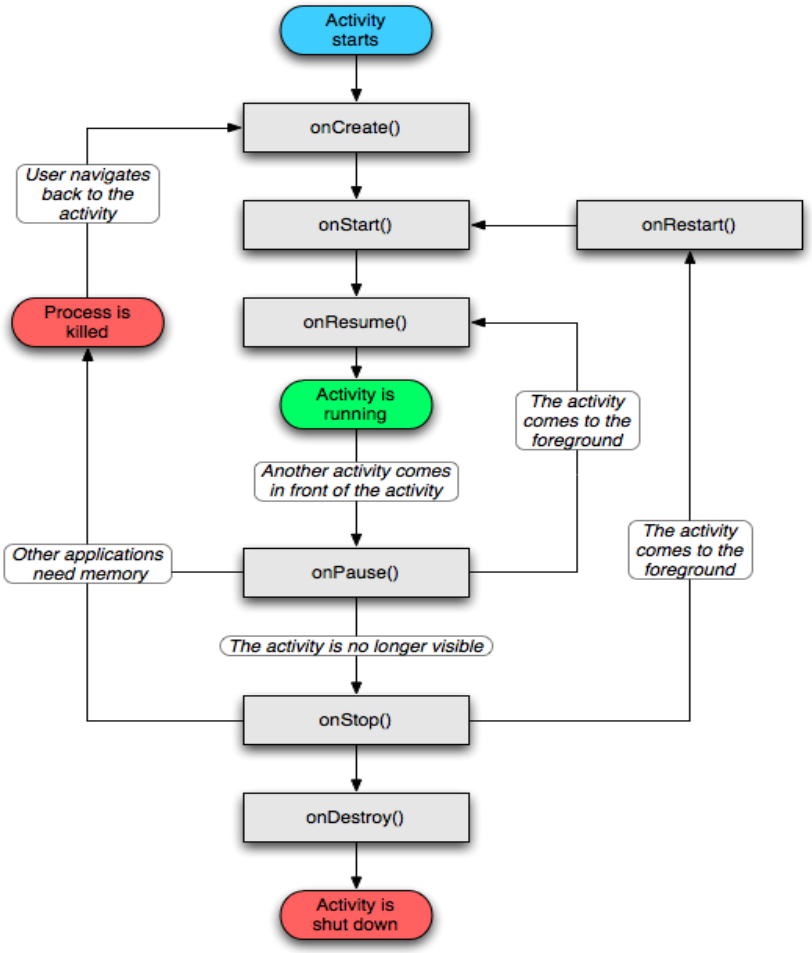
Onderstaande uitleg is een geparafraseerde beschrijving van de Android Developer's Guide.

Een Android applicatie bestaat uit een of meerdere activities (activiteiten). Een activity is een applicatie component dat zorgt voor de schermweergave waarop een gebruiker interactie kan uitvoeren. Denk hierbij aan een scherm om een telefoonnummer in te vullen of een mail te versturen. Om naar een ander scherm te gaan, bijvoorbeeld bij het opzoeken van contactpersonen, wordt een nieuwe activity gestart met nieuwe functionaliteiten. Om ervoor te zorgen dat activities het geheugen niet bezet houden, heeft Android per activity een eigen lifecycle. Door deze lifecycle kan een activity zich in een van de volgende drie statussen bevinden.

- **Resumed**: De activity is actief en de gebruiker kan via de user interface de applicatie bedienen. Deze status wordt ook wel Running genoemd.
- **Paused**: Een andere activity is actief, maar is niet full screen. De activity op de achtergrond blijft actief en is deels zichtbaar. De gebruikersinstellingen blijven bewaard, mocht de gebruiker terug keren naar deze activity.
- **Stopped**: Een andere activity is momenteel actief waardoor de huidige activity zich op de achtergrond bevind. Het activity object blijft in het geheugen staan, waardoor de gebruikersinformatie bewaard blijft. Zodra Android geheugen nodig heeft, is het mogelijk dat Android deze afsluit, hierover heb je geen controle als ontwikkelaar/gebruiker.

Dankzij deze feature van Android, beheert Android zelf het geheugengebruik van de diverse applicaties. Mocht een zware applicatie opgestart worden, dan zal Android uit zichzelf overbodige activities afsluiten.

Om deze wisselingen tussen bovenstaande statussen beheersbaar te houden, heeft Android een aantal vooraf geïmplementeerde functies. Deze worden standaard aangeroepen door Android. Zoals zichtbaar in Figuur 8: Android activity lifecycle zijn er een zevental standaardfuncties.



Figuur 8: Android activity lifecycle

In de volgende tabel wordt beschreven wat deze functies inhouden en wanneer deze worden aangeroepen.

Tabel 3: Android lifecycle

Methode	Omschrijving	Volgende
onCreate()	Deze functie wordt als eerste en eenmalig aangeroepen in zijn levensduur, hiermee is het mogelijk om verbindingen te openen.	onStart()
onRestart()	Deze functie wordt enkel aangeroepen als de applicatie eerder de onStop() heeft aangeroepen.	onStart()
onStart()	Deze functie wordt aangeroepen op het moment dat de activity zichtbaar wordt voor de gebruiker.	onResume() onStop()

onResume()	De onResume() functie wordt als laatste aangeroepen voordat de activity Resumed / Running is. Op dit moment wordt deze activity de activity in Android waar user input naar toe gaat.	onPause()
onPause()	Zodra de focus van de activity verandert, naar een andere activity of een pop-up, wordt de onPause() aangeroepen.	onResume() onStop()
onStop()	Zodra de gebruiker een andere activity opent, bijv. door het drukken op een knop, zal van de huidige activity de onStop() aangeroepen worden. Hierin is het mogelijk om gebruikersinput te bewaren.	onRestart() onDestroy()
onDestroy()	Deze functie wordt enkel aangeroepen op het moment dat de activity wordt gestopt, dit is tevens de laatste functie van de levensduur van een activity. De onDestroy() kan worden aangeroepen door de gebruiker (applicatie afsluiten) of door Android zelf, als deze activities afsluit voor extra geheugen.	-

Zoals hierboven beschreven, betekent het dus dat er maar slechts één activity actief kan zijn. Alle overige activities bevinden zich in de status Paused of Stopped.

4.5 Toepassing ideeën

Om een interessante toepassing met OBD te bedenken zijn hiervoor eerst diverse doelgroepen uitgekozen. Om te voorkomen dat er ideeën over het hoofd gezien werden, is er tijdens de brainstormsessie niet gekeken naar de mogelijke beperkingen van OBD. Uit deze sessie zijn de volgende ideeën tot stand gekomen:

Consumenten:
Als consumententoepassing is het mogelijk om deze op milieuvriendelijkheid of zuinigheid (Het nieuwe rijden) te betrekken. Daarnaast waren er ideeën zoals een schakelhulp of een snelheidsmonitor die voorkomt dat de gebruiker harder gaat dan de snelheidslimiet van zijn huidige weg. Het bijhouden van ritstatistieken, zoals afstand, verbruik, etc. is ook mogelijk, wat dan in combinatie met het vergelijken van gegevens van een andere rit of zelfs met ritten van andere gebruikers mogelijk is (zowel offline als online).

Lease/verzekeringsmaatschappijen:
Bij lease/verzekeringsmaatschappijen is het wellicht handig om te weten hoe de gebruiker met de auto omgaat, denk hierbij aan of er netjes met de auto wordt omgegaan of dat deze vaak de snelheid overtreedt. Op basis hiervan is het mogelijk dat er speciale acties komen voor de "nette" bestuurders.

Wegenwacht:
Voor de wegenwacht zijn de beschikbare foutcodes ideaal om onderweg beschikbaar te hebben. Daarnaast is het mogelijk om actuele sensorinformatie op te vragen. Op deze manier zou de wegenwacht problemen sneller kunnen detecteren.

Figuur 9: Logo ANWB

Na een verdere uitwerking van de ideeën, is er gekeken naar de realisatiemogelijkheden van de toepassingen. Op basis hiervan zijn enkele ideeën afgevallen, zoals de wegenwacht toepassing waarbij foutcodes uitgelezen worden. Ook bleek na onderzoek dat het niet mogelijk was om betrouwbare gegevens te vermelden over milieuvriendelijkheid. De overige ideeën zijn samengevoegd tot een applicatie waar verder op in wordt gegaan in het volgende hoofdstuk.

5. Requirements

In dit hoofdstuk zullen de ideeën uit beide onderzoeken worden uitgewerkt tot requirements voor een volledige applicatie. De reden waarom bepaalde onderdelen wel/niet in de toepassing zijn gekomen wordt hier vermeld. Aangezien de applicatie enkel wordt gebruikt voor demo doeleinden bij TASS is geen rekening gehouden met commerciële belangen van de toepassing.

5.1 Sensormogelijkheden

Zoals al in het vorige hoofdstuk beschreven, is er tijdens het onderzoek nagedacht over mogelijke toepassingen voor een applicatie. Deze gingen van een foutcode lezer voor de wegenwacht tot aan consumentenapplicatie waarbij het milieu centraal stond. Naast een goed idee voor een toepassing moet dit ook gebruik maken van OBD. Hiervoor is een overzicht opgesteld van de sensoren die aanwezig zijn in een gemiddelde auto[1] op basis van de beschikbare testresultaten.

- **Calculated engine load value:** door de auto berekende motorbelasting.
- **Engine coolant temperature:** de temperatuur van de koelvloeistof.
- **Engine RPM:** het toerental van de motor
- **Vehicle speed:** de snelheid.
- **Intake air temperature:** de temperatuur van de lucht waarmee de koelvloeistof gekoeld wordt.
- **Run time since engine start:** aantal secondes dat de motor draait.
- **Distance traveled since codes cleared:** gereden kilometers sinds de laatste keer dat de foutcodes verwijderd zijn.
- **Absolute load value:** de absolute motorbelasting.
- **Ambient air temperature:** de temperatuur van de buitenlucht.
- **Accelerator pedal position D:** de stand van het gaspedaal.
- **Accelerator pedal position E:** een tweede sensor voor de stand van het gaspedaal. Vaak gebruikt als back-up.

Zoals zichtbaar zijn enkel de basisgegevens over een auto beschikbaar, zoals toerental, snelheid en de motorbelasting. Hierdoor wordt de mogelijkheid om iets over de zuinigheid te bepalen een stuk lastiger. Hierover is geen directe informatie beschikbaar. Omdat er al een applicatie gebruikt wordt door de wegenwacht om OBD foutcodes uit te lezen is er voor gekozen geen soortgelijke applicatie te ontwikkelen.

Wat overbleef was een idee om gegevens tijdens een rit weer te geven aan een automobilist zodat hij deze live kon bekijken. Na een kleine zoektocht bleek dat hiervan al diverse applicaties beschikbaar zijn voor Android en andere platformen. Dit zijn vooral “domme” applicaties die sensorwaardes uitlezen en deze daarna weergeven op het scherm. Om dit principe te evolueren is de Drive Style Analyst applicatie (DSA app) bedacht.

Figuur 10: App logo

[1] Een gemiddelde auto bevat 50% van de benoemde sensoren.

5.2 Drive Style Analyst applicatie

Het doel van de DSA app is om de gebruiker bewust te maken van zijn huidige rijstijl en om deze te verbeteren. Door de rijstijl van de gebruiker te verbeteren, zorgt deze ervoor dat er minder schadelijke stoffen in het milieu komen. Daarnaast profiteert de gebruiker ook direct van een brandstofbesparing en zorgt hij voor meer veiligheid op de weg.

De rijstijl van de gebruiker wordt hiervoor gemonitord en gelogd. Tijdens een rit worden de actuele autogegevens gemonitord, denk hierbij aan de tijdsduur, de gereden afstand en het toerental. Op basis van deze gegevens worden er, tijdens de rit, informatieve berichten gegenereerd. Denk hierbij aan berichten op het moment dat een gebruiker een positieve of negatieve handeling heeft verricht.

Zodra de rit is beëindigd zal alle gelogde data worden geanalyseerd tot diverse statistieken en een ritscore. De statistieken bestaan uit gegevens over de huidige rit, dit kan bijvoorbeeld zijn: de totale tijd dat de auto heeft stilgestaan (wachttijd bij stoplichten), de gereden afstand, de gemiddelde snelheid, enz. De totale lijst van de statistieken is terug te vinden in het requirements document.

5.2.1 Rit types

Een rit wordt opgedeeld in drie vooraf bepaalde rit-types. Het makkelijkste kan het rit-type vergelijken worden met een landschap. De mogelijke rit-types zijn:

- “Snelweg” (snelweg).
- “Platteland” (buiten bebouwde kom).
- “Stad” (binnenstad).

Figuur 11: Rit-type iconen

Hiervoor is gekozen om per rit-type een andere analyse uit te voeren, waardoor de analyse specifiek op rit-type ontworpen kon worden. Om ervoor te zorgen dat het systeem dynamisch en flexibel blijft, is ervoor gekozen om per rit-type een aantal subscores te geven. De volgende subscores zijn mogelijk op basis van de beschikbare sensoren.

- Snelheid
- Toerental
- Motorbelasting
- Gaspedaal positie
- Koude motor gebruik

Dankzij het gebruik van deze subscores per rit-type is het mogelijk specifieke onderdelen van de rijstijl te analyseren en hierop punten toe te kennen. Denk hierbij aan grote snelheidsverschillen in de stad, of een stabiele gaspedaalpositie op de snelweg. Als, door het ontbreken van sensoren, een subscore niet berekend kan worden. Zal deze niet meegeteld worden in de score voor dit rit-type.

Naast deze ritscore is het mogelijk om specifieke gegevens hiervan te bekijken zoals behaalde subscores en rit-type specifieke statistieken. Ook is het mogelijk om de verkregen informatieberichten die tijdens de rit gegeven zijn opnieuw te bekijken. Hiermee kan de automobilist bewust worden van zijn huidige rijstijl en deze waar nodig verbeteren.

5.2.2 Ritscore bepaling

Zoals al eerder vermeld, wordt de rijstijl van de automobilist gemonitord en gelogd. Als eindresultaat krijgt de automobilist een ritscore (een waarde tussen 0-100), deze score is berekend op basis van gereden rit-types. Een rit-type is opgebouwd op basis van de eerder genoemde subscores (snelheid, toerental, motorbelasting, gaspedaal positie en koude motor gebruik).

Per rit-type en subscore zijn aparte formules ontworpen die aansluiten bij de mogelijke situaties van het desbetreffende rit-type. Extra informatie over deze formules die gebruikt worden bij de analyse zijn terug te vinden in bijlage B: Software Requirements Specification (SRS). Zo wordt er gekeken naar een constant toerental op de snelweg, waar er in de binnenstad extra aandacht wordt gegeven aan snelheidsverschillen (zowel positief als negatief) bijvoorbeeld bij het optrekken en afremmen in de buurt van stoplichten. Elke formule maakt gebruik van diverse standaardwaardes en enkele normwaardes. Deze normwaardes zijn opgebouwd op basis van het autotype en de eerder gelogde ritten met dat autotype. Hoe deze normwaardes tot stand komen, wordt later toegelicht in 5.3 Self Learning System.

Tegenwoordig zijn er diverse automerken, elk met weer allerlei verschillende modellen en mogelijke configuraties. Om ervoor te zorgen dat de normwaardes gebaseerd zijn op vergelijkbare auto’s is ervoor gekozen om de auto’s in de delen in verschillende autotypes. Bij deze indeling is rekening gehouden met het type brandstof en de motorinhoud van een auto. Er is een bewuste keuze gemaakt voor een onderverdeling in brandstof en motorinhoud aangezien deze twee parameters belangrijk zijn in de karakteristieke eigenschappen van de auto. Ook had het gewicht meegenomen kunnen worden maar hiervoor is niet gekozen omdat nooit precies vast te stellen is wat het daadwerkelijke gewicht van een auto is. Het aantal personen en bagage is namelijk een onbekende factor hierbij. Daarnaast zou dat resulteren in een veelvoud van auto types, een aantal wat dankzij deze oplossing tot 16 beperkt is gebleven.

Tabel 4: Autotype

Motorinhoud	<1.0	1.0 – 1.3	1.3-1.6	1.6-1.9	1.9-2.1	2.1-2.4	2.4>	Onbekend
Benzine	X	X	X	X	X	X	X	X
Diesel	X	X	X	X	X	X	X	X

Dankzij het kenteken van een auto, is het mogelijk om via internet informatie over de desbetreffende auto op te halen, op basis hiervan is het mogelijk om een auto in te delen in een van de bovenstaande autotypes.

5.3 Self Learning System

Het self learning system is ontwikkeld om gegevens van verschillende auto's binnen een categorie met elkaar te vergelijken. Hierdoor kan dit systeem de normwaardes die gebruikt worden bij de formules steeds verder specificeren. Hierdoor kan de DSA app alle categorieën op dezelfde manier te beoordelen. Hoe dit systeem functioneert wordt hieronder beschreven.

Voor elke combinatie van rit-type (3: snelweg, platteland, binnenstad), autotype (16: zie Tabel 4: Autotype) en subscore een normwaarde gebruikt. Als je even snel gaat rekenen kom je uit op 192 (3 * 16 * 4) verschillende normwaardes.

Zoals hierboven te zien, zijn er enkel 4 normwaardes voor de 5 verschillende subscores (snelheid, toerental, motorbelasting, gaspedaal positie en koude motor gebruik). Dit komt vanwege het feit dat de koude motor wordt berekend door de overige subscores te middelen, echter zijn de normwaardes voor deze berekeningen 15% lager/zuiniger. Het doel hiervan is de gebruiker extra te motiveren om aan het begin van de rit, als de motor nog niet op de ideale temperatuur is, rustig te rijden. Hieraan wordt extra aandacht gegeven aangezien het rijden met een koude motor resulteert in een exponentieel hoger brandstofverbruik en een grotere vervuiling voor het milieu.

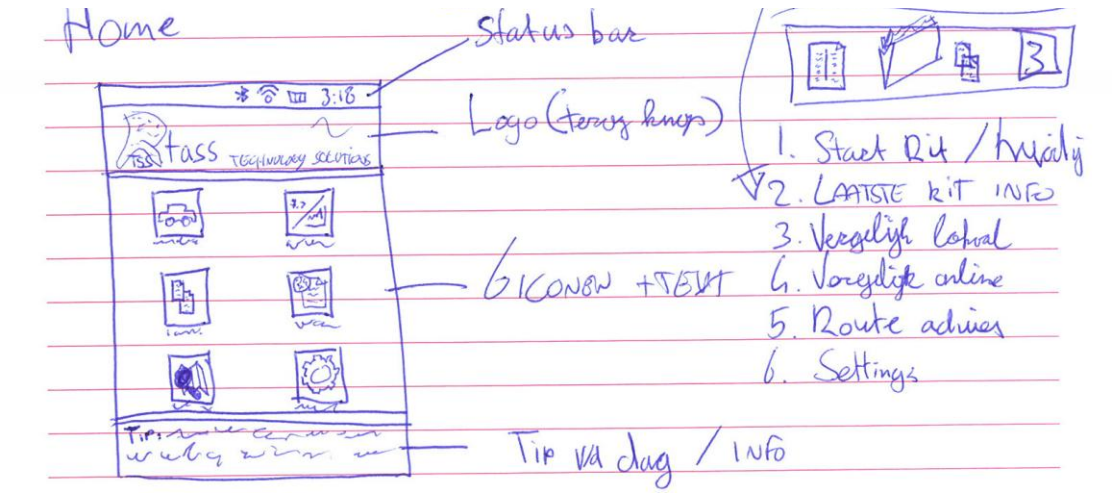
Nadat de DSA app een ritscore heeft berekend, zal deze rit geüpload worden. De opslag van ritten gebeurt om meerdere redenen online. Ten eerst is het op deze manier mogelijk om het self learning system van data te voorzien zodat deze nieuwe normwaardes kan genereren. Daarnaast kan je op deze manier vanuit de applicatie je eigen ritten met online ritten vergelijken. Ook is het mogelijk om in de toekomst een web interface te ontwikkelen waarbij gebruikers online ritinformatie kunnen bekijken. Om de privacy in acht te nemen, worden er geen gebruikerspecifieke onderdelen vrijgegeven vanaf de server.

Het self learning system wordt dagelijks automatisch geactiveerd op de server. Hierbij wordt gekeken van welke autotypes nieuwe ritten zijn geüpload, waarna de normwaardes van dit type opnieuw wordt berekend met alle beschikbare gegevens. De normwaardes van de subscores snelheid, toeren en motorbelasting worden online berekend.

De normwaarde van de vierde subscore, namelijk de gaspedaalpositie, wordt op het Android apparaat berekend. Dit komt vanwege de grote verschillen tussen de gaspedaalposities van auto's. Ook deze normwaarde wordt berekend op het moment dat de rit is afgerond.

5.4 User interface

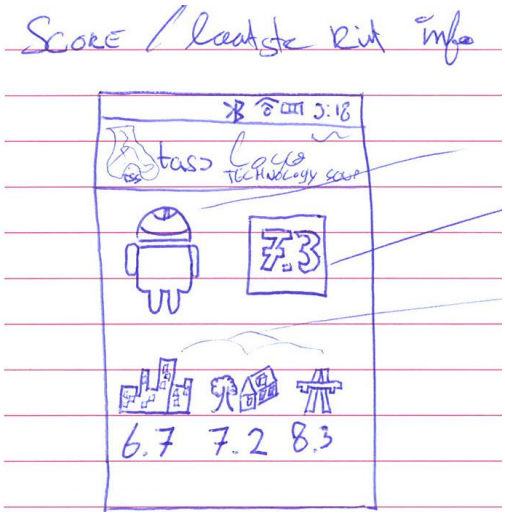
Een belangrijk onderdeel van de gebruikerservaring is de “look and feel”. Om de user interface (UI) op een gestructureerde wijze op te zetten, is ervoor gekozen deze tijdens een brainstorm sessie eerst op papier uit te werken. Tijdens het uitwerken van de UI is er bewust rekening gehouden met de relatief kleine schermen van mobiele apparaten. De user input gebeurd meestal via het touchscreen in tegenstelling tot de hardware knoppen die vaak niet beschikbaar zijn op Android apparaten. Hiermee is tijdens het ontwerpen rekening gehouden, zo zijn de knoppen extra groot en is er bewust nagedacht over de zichtbare inhoud. Naast eerder genoemde onderdelen is ook gebruik gemaakt van de User Interface Guidelines van de Android Developer’s website zodat de UI zo goed mogelijk aansluit bij de standaard Android applicaties. Op basis van een brainstormsessie waarbij de belangrijkste requirements en functionaliteiten zijn bepaald is er begonnen het ontwerp op papier uit te werken. Een voorbeeld van een dergelijke papieren schets is hieronder afgebeeld.



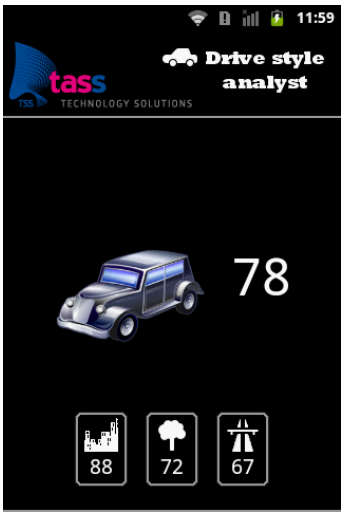
Figuur 12: Homescreen schets

Om alle schermen van de DSA app herkenbaar te maken is ervoor gekozen om een standaard lay-out te ontwikkelen. Op deze manier kunnen alle schermen van de applicatie op dezelfde manier gestructureerd worden. Deze lay-out bestaat uit een header, waarin het TASS logo en het logo van de DSA app in terugkomen en is het per scherm mogelijk om een footer toe te voegen. In deze footer is ruimte gereserveerd voor informatieberichten, die onder andere de functionaliteiten van de applicatie beschrijft in de vorm van tips. Ook kan deze footer gebruikt worden om informatieberichten over de huidige rit weer te geven. De overige ruimte (het middenstuk) wordt gebruikt om de functionaliteiten van het desbetreffende scherm weer te geven.

Om de gebruiker een zo optimaal mogelijke gebruikerservaring te geven, is ervoor gekozen om per functionaliteit een herkenbaar icoon (afbeelding) te gebruiken. Zo is het icoon voor de functie "Start rit" een auto, waar de vergelijkingsfunctie een icoon heeft met meerdere documenten (denkbeeldige ritstatistieken). Voor de rit-types is gekozen voor een skyline (stad), boom en boerderij (platteland) en snelwegbord (snelweg). Zie Figuur 13: Ontwerp schets van het ritscore scherm voor de uitwerking van deze iconen.



Figuur 13: Ontwerp schets van het ritscore scherm



Figuur 14: Digitale versie van het ritscore scherm

Nadat alle functionaliteiten zijn uitgewerkt in diverse schetsen, was de volgende stap om deze te digitaliseren. Hiervoor is gekozen om deze stap direct in de ontwikkelomgeving te doen i.p.v. een grafisch ontwerp programma zoals Photoshop. Deze beslissing is genomen omdat er weinig kennis van Photoshop, of andere tekenpakketten, beschikbaar was en beide mogelijkheden ongeveer evenveel tijd in beslag zouden nemen. Daarnaast konden de ontwerpen in Eclipse direct gebruikt worden in de uiteindelijke applicatie waardoor er voor een directe uitwerking in Eclipse is gekozen.

Om de user interface volledig te digitaliseren is er gekeken naar bestaande icon sets waarbij de gewenste iconen beschikbaar waren. Dankzij de website www.iconfinder.com, is er een complete icon set (abstracte zwart-wit iconen) gevonden, die het merendeel van de gewenste iconen bevatte. Enkele iconen zijn zelf ontworpen en/of aangepast zodat alle gewenste iconen aanwezig waren. Er is gekozen om deze zwart-wit stijl te verwerken in de gehele applicatie, aangezien het gebruikte Android ontwikkelapparaat ook met deze stijl was opgemaakt.

Dankzij gebruik te maken van een zwart-wit stijl, heeft de applicatie een professionele "look en feel" en is de stijl vergelijkbaar met de standaard stijl van het Android apparaat. Dit verhoogt de herkenbaarheid van de applicatie voor de gebruikers.

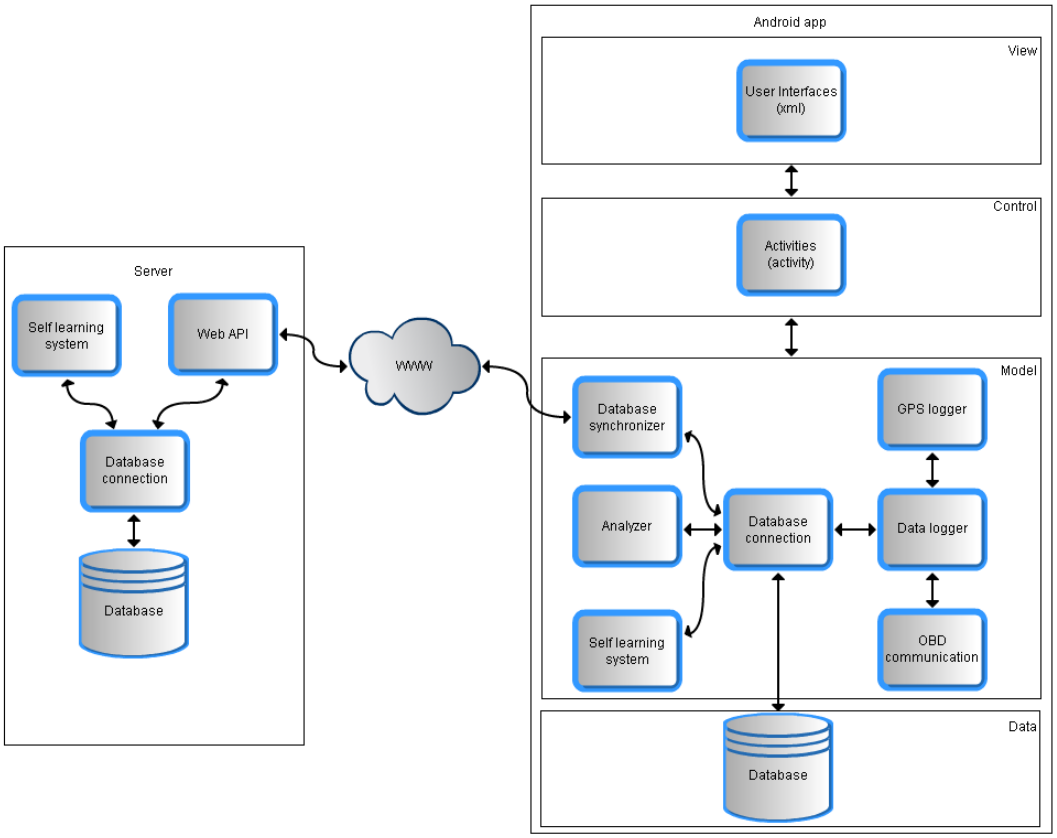
6. Ontwerpen

In dit hoofdstuk wordt ingegaan op het ontwerp van de applicatie. Als eerste zal het ontwerp besproken worden dat gebruikt is bij de implementatie van de applicatie. Hierin worden alle onderdelen van de applicatie uitgelegd, waarna een aantal interessantere onderdelen specifiek worden toegelicht.

Hierna wordt het proces van het ontwerpen en de beslissingen die hierbij gemaakt zijn extra naar voren gebracht. De volgorde van dit onderdeel van het hoofdstuk heeft dezelfde volgorde als waarin de applicatie ontworpen is en begint daarmee met een globaal overzicht van de volledige applicatie in de vorm van een zogenaamd Data Flow Diagram (DFD). Hierdoor wordt inzichtelijk gemaakt welke gegevensstromen binnen de applicatie gebruikt worden en tussen welke onderdelen de uitwisseling van gegevens plaatsvindt.

6.1 Ontwerp

Hieronder is een overzicht te zien van het ontwerp dat gebruikt is tijdens het implementeren van de applicatie. Dit ontwerp is het eindresultaat van het ontwerpproces en is na afloop van dit proces samen met een deskundige binnen TASS besproken en geverifieerd.



Figuur 15: Data Flow Diagram verbeterd ontwerp

Als toelichting op bovenstaand Figuur 15: Data Flow Diagram verbeterd ontwerp wordt elk onderdeel van beide applicaties hieronder beschreven.

Android applicatie

- **Database connection**: Dit onderdeel heeft als enige een directe verbinding met de database en bevat functies om deze data te selecteren, in te voeren, te updaten en te verwijderen. De database connection staat in verbinding met bijna alle andere onderdelen van het ontwerp. Dit omdat al deze onderdelen toegang nodig hebben tot de gegevens die in de database opgeslagen worden.
- **Data logger**: De data logger is verantwoordelijk voor het uitlezen van de juiste sensoren die passen bij het huidige rit-type (dit type wordt bepaald door de analyzer). Dit onderdeel heeft hiervoor een aantal voorgedefinieerde lijsten met sensoren die per rit-type uitgelezen worden. Dit onderdeel staat volledig los van de gebruiker en wordt gestart op het moment dat de applicatie gestart wordt. Naast de sensorinformatie wordt ook locatie-informatie uitgelezen. Dit wordt verzorgd door de GPS logger. Alle verkregen gegevens zullen door de data logger via de database connection opgeslagen worden.

Om de data logger op de achtergrond zijn werk uit te laten voeren, is er gekozen om dit onderdeel te implementeren als Android service. Wat hiervan de voordelen zijn en waarom hiervoor gekozen is staat verder beschreven bij hoofdstuk 6.2.4. Tevens heeft dit onderdeel ook een observable wat betekend dat deze updates kan leveren aan een observer. Wat hiervan de voordelen zijn en waarom hiervoor gekozen is staat verder beschreven bij hoofdstuk 6.2.3.

- **OBD communication**: het OBD communication onderdeel beheert alle communicatie met het OBD Bluetooth apparaat. Het is in staat om gegevens op te vragen van de ondersteunde sensoren en gebruikt hiervoor twee sub onderdelen; de Bluetooth verbinding en de OBD parser.
    - **Bluetooth verbinding**: dit onderdeel heeft als functie om een verbinding op te stellen met het OBD Bluetooth apparaat en hier gegevens naar toe te sturen en te ontvangen.
    - **OBD parser**: Na het ontvangen van een reactie van het OBD apparaat zorgt dit onderdeel dat het OBD protocol vertaald wordt naar een sensorData object. In dit sensorData object zijn de sensor gegevens opgeslagen. Deze sensor gegevens worden dan verder verwerkt door de data logger.
- **GPS logger**: Dit onderdeel is verantwoordelijk voor het verkrijgen van locatie informatie van het Android apparaat. Net zoals de OBD sensor gegevens zullen deze GPS gegevens door de datalogger verder verwerkt worden.
- **Analyzer**: De analyzer zal in een aparte thread gestart worden op het moment dat er een rit gestart is. Dit onderdeel zal gedurende deze rit elke 30 seconden een analyse uitvoeren op de data die in die 30 seconden opgeslagen is door de data logger. Op basis van deze gegevens worden de scores berekend die aan het eind van de rit weergegeven worden aan de gebruiker. Verder is dit onderdeel verantwoordelijk voor het bepalen van het rit-type. Aan het eind van de rit zal de laatste analyse uitgevoerd worden waarna de ritscores en statistieken berekend worden.
- **Self learning system**: Dit systeem wordt pas in gang gezet op het moment dat er een rit is afgerond. Het systeem zal dan alle gegevens van de ritten op het apparaat gaan analyseren om vanuit die gegevens een nieuwe normwaarde voor de berekening van de gaspedaalpositie subscore te bepalen.
- **Database Synchronizer**: Als een nieuwe rit is afgerond en er een actieve internetverbinding aanwezig is zal de database synchronizer zijn werk starten. Deze zal de communicatie met de server tot stand brengen en de data van de rit uploaden via de web API van de server. Deze staat hierna beschreven.

Server applicatie

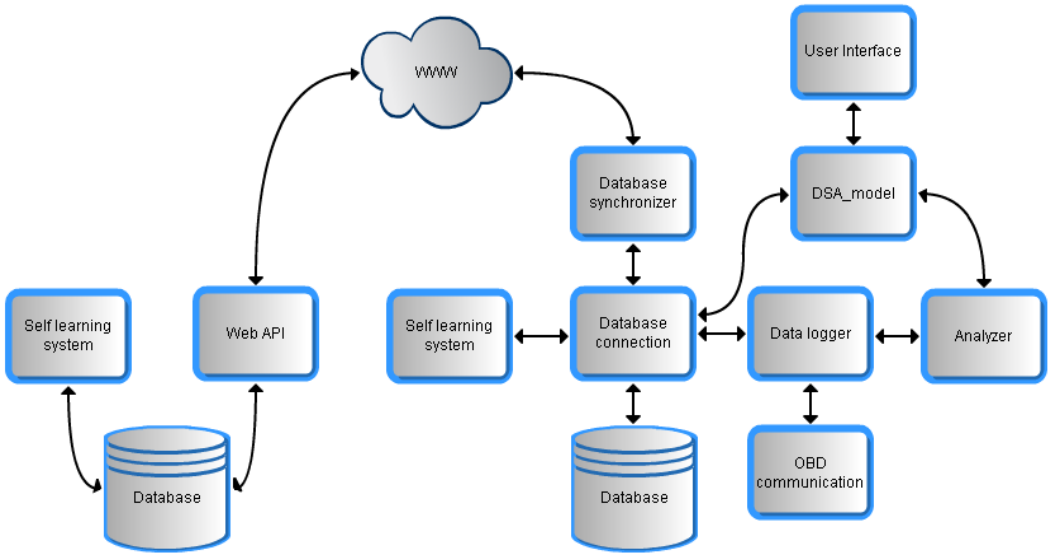
- **Database connection**: Dit onderdeel is net als bij de Android applicatie verantwoordelijk voor de connectie met de database zodat de gegevens geselecteerd, ingevoerd, geüpdate en verwijderd kunnen worden.
- **Web API**: De web API is toegankelijk via internet zodat de verschillende Android apparaten gegevens kunnen opvragen en versturen naar de server.
- **Self learning system**: Dit systeem wordt dagelijks gestart. Hierbij worden alle gegevens in de database geanalyseerd op basis van de verschillende autotypes. Voor de subscores die niet auto afhankelijk zijn, zal door dit system nieuwe normwaardes voor de berekeningen bepaald worden.

Voor elk van de bovenstaande onderdelen is ook een klassendiagram uitgewerkt zodat er tijdens het implementeren van de applicaties minimale tijd besteed hoeft te worden aan nadenken over functies en de naamgeving hiervan. Deze klassendiagrammen worden verder niet besproken in de scriptie maar zijn te bekijken in bijlage C: Technisch Ontwerp

6.2 Ontwerp proces

6.2.1 Data Flow Diagram

Er is tijdens het ontwerpen van de DSA app rekening gehouden met uitbreidbaarheid en beheersbaarheid. Zo zijn de verschillende functionaliteiten in logische modules verdeeld. Als basis van het ontwerp zijn we begonnen met een brainstormsessie waaruit een Data Flow Diagram (DFD) is opgezet. De eerste opzet ziet er uit als in Figuur 16: Data Flow Diagram tijdens ontwerpproces.



Figuur 16: Data Flow Diagram tijdens ontwerpproces

In bovenstaand DFD zijn de basiselementen van de applicatie opgenomen zodat elk onderdeel verder uit te werken is waarbij alle functionaliteiten uit het requirements document geïmplementeerd kunnen worden. Tijdens de opzet van het ontwerp is er rekening gehouden met Android specifieke features die bij de hoofdstukken 6.2.4 Threads en Services en 6.2.5 Notification manager verder toegelicht worden.

Na het maken van de eerste versie van het DFD is er gekeken naar hoe de software opgezet kan worden met een goede structuur. Hierbij is bewust nagedacht over welke software ontwerppatronen gebruikt konden worden om de applicatie een goede softwarestructuur te geven en de beheersbaarheid van deze applicatie hierdoor te verbeteren. Er is hierbij gekozen voor verschillende software ontwerppatronen die verder in dit hoofdstuk per stuk beschreven zijn inclusief de motivatie waarom dit patroon gebruikt is.

6.2.2 Model View Controller patroon

Dit patroon is gebruikt om de applicatie een betere structuur te geven. Hierdoor is de functionaliteit van de applicatie goed gescheiden van de data en de user interface. Hoe dit gedaan is en wat het patroon precies inhoud staat in de volgende paragrafen beschreven.

6.2.2.1 Uitleg ontwerppatroon

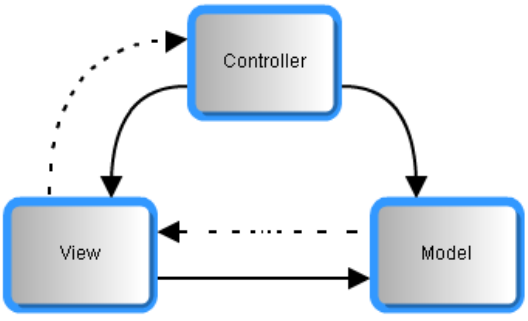
Het model view controller (MVC) ontwerppatroon is een ontwerppatroon dat gebruikt wordt om een complexe applicatie op te delen in drie onderdelen met eigen verantwoordelijkheden. De drie onderdelen zijn; het datamodel(model), de datapresentatie(view) en de applicatielogica(controller). Het grootste voordeel van het gebruik van het MVC ontwerppatroon is dat wijzigingen in de user interface niet direct betrekking hebben op het datamodel en vice versa. Ook zorgt dit ontwerppatroon er voor dat de code een stuk overzichtelijker wordt en is te hergebruiken.

Hieronder staat een korte omschrijving van de verantwoordelijkheden van de verschillende onderdelen van het ontwerppatroon.

Model
Definieert de representatie van de informatie waarmee de applicatie werkt. Aan ruwe gegevens wordt betekenis gegeven door relaties tussen data en logica toe te voegen. De daadwerkelijke opslag van data wordt gedaan met behulp van een persistent opslagmedium, zoals een database. De applicatie zal gegevens die gebruikt worden in het model, ophalen en wegschrijven van en naar de dataopslag via een datalaag. De datalaag is geen verplicht onderdeel van het MVC patroon.

View
Informatie wordt weergegeven via de View. User interface elementen zullen gedefinieerd zijn in dit onderdeel. De view doet geen verwerking zoals berekeningen, controles, of het aanpassen van de gegevens die getoond worden.

Controller
De controller reageert op events en handelt deze af. Deze events zijn meestal het gevolg van handelingen van de gebruiker. Een event is een gebeurtenis in een computerprogramma waarop dit programma kan reageren, dit kan bijvoorbeeld een gebruikersactiviteit zijn of een update van een observer.



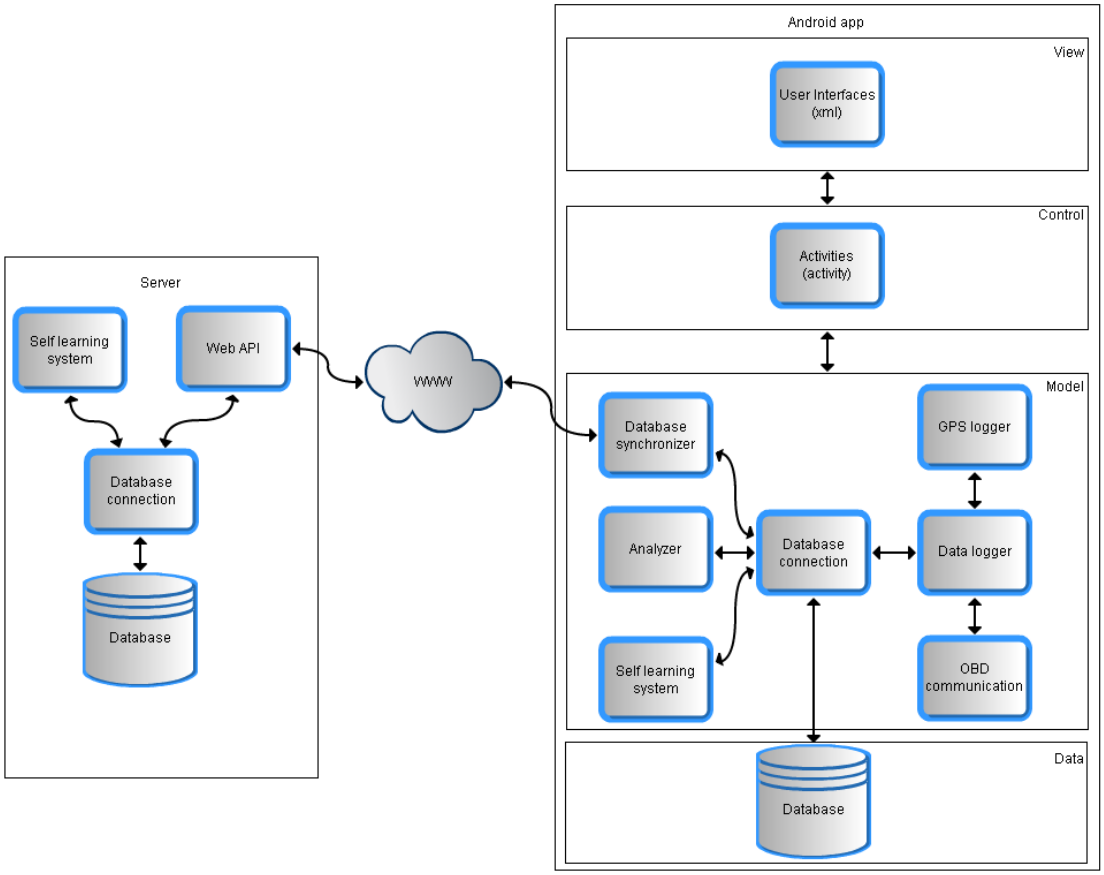
Figuur 17: MVC patroon

6.2.2.2 Toepassing in de applicatie

Het MVC ontwerppatroon is in de Android applicatie op een eenvoudige manier toe te passen. Android gebruikt standaard een zogenaamde activity om alle gebruikersinput op te vangen. Deze activities functioneren dus als controllerlaag van de applicatie. Om te bepalen hoe de gegevens weergegeven worden aan de gebruiker en hoe de elementen van de user interface opgebouwd zijn, wordt in Android gebruik gemaakt van XML files. Deze XML files bevatten alleen de weergave elementen van de user interface en zijn de view van het ontwerppatroon.

Alle onderdelen die data bevatten of verder de data van logische relaties voorzien, zijn onderdelen van het model. In dit model zit ook de datalaag van de applicatie die als enige de verbinding met het opslagmedium heeft.

Hieronder in Figuur 18: Data Flow Diagram ontwerp met toepassing MVC is te zien hoe het ontwerp er met de toevoeging van het MVC patroon uitziet.



Figuur 18: Data Flow Diagram ontwerp met toepassing MVC

6.2.3 Observer patroon

Dit patroon is toegepast om de user interface automatisch van nieuwe informatie te voorzien. Door dit ontwerppatroon te gebruiken kan dit op een efficiënte manier gebeuren. Hoe dit gedaan is en wat het patroon precies inhoud staat in de volgende paragrafen beschreven.

6.2.3.1 Uitleg ontwerppatroon

Het observer patroon is een ontwerppatroon dat op een efficiënte en ontkoppelde manier objecten in een programma kennis laat nemen van statusveranderingen of veranderingen van gegevens binnen andere objecten van hetzelfde programma.

Het observer patroon is van toepassing op situaties waarin een entiteit binnen een programma moet reageren op een gebeurtenis binnen hetzelfde programma, maar waarbij de toestandsverandering niet in die entiteit zelf plaatsvindt. Het klassieke voorbeeld hiervan is een spreadsheet: als de waarde in een cel verandert, moeten bijvoorbeeld grafieken die dezelfde waarde weergeven mee veranderen. Het doel van dit ontwerppatroon is dat een toestandsverandering in een object opgemerkt wordt in een ander object. En dit het liefst zonder dat de objecten sterk gekoppeld zijn (zodat de objecten vrij onafhankelijk van elkaar bestaan en wellicht hergebruikt kunnen worden). Ook is een belangrijke waarde van dit ontwerppatroon dat het opmerken van de statusverandering niet al te duur is. In het geval van een spreadsheet: om een grafiek actueel te houden, is het bij voorkeur niet nodig dat iedere microseconde de hele spreadsheet doorlopen wordt om veranderingen te detecteren.

6.2.3.2 Toepassing in de applicatie

Omdat de user interface door het MVC patroon ontkoppeld is van het model zorgt de controller ervoor dat de juiste data op het scherm getoond wordt. Dit is in de meeste gevallen een geschikte manier om de data uit het model op te halen en weer te geven. Echter zijn er een aantal functionaliteiten die eisen dat deze data regelmatig vernieuwd wordt. Om hierin op een efficiënte manier te kunnen voorzien, is er gekozen voor het observer patroon. Dit patroon wordt in de applicatie toegepast bij het vernieuwen van de gegevens die tijdens de rit weergegeven worden. In Figuur 19: Rit loggen weergave is te zien dat de huidige gereden afstand en tijdsduur worden weergegeven. Deze gegevens worden door de data logger aangeboden door middel van een observable. Ook laat dit scherm het huidige rit-type en de informatie berichten die door de analyzer worden gegenereerd zien. Deze gegevens worden door de analyzer ook door middel van een observable kenbaar gemaakt aan deze user interface. Op deze manier zal dus niet het hele scherm vernieuwd hoeven te worden op het moment dat één van deze observables nieuwe informatie beschikbaar heeft.

Figuur 19: Rit loggen weergave

6.2.4 Threads en Services

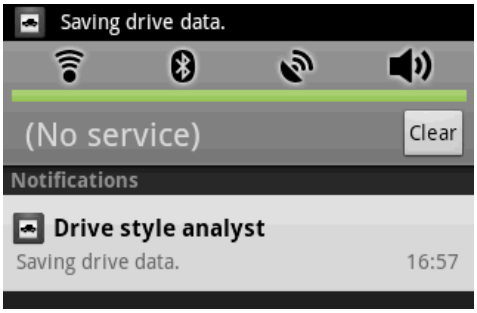
Een extra aandachtspunt tijdens het ontwerpen van de applicatie was het gebruik van threads en Android services. Over het algemeen heeft een simpele Android applicatie een thread die alle werkzaamheden uitvoert. Naar deze thread wordt meestal verwezen als user interface thread, dit omdat deze thread ook de gebruikersinput afhandelt. Bij het uitvoeren van langdurige taken is het verstandig om gebruik te maken van een extra thread zodat deze taak niet de gebruikersinput blokkeert.

Bij een taak die gedurende langere tijd op de achtergrond uitgevoerd dient te worden en geen extra input van de gebruiker vergt, kan gebruik gemaakt worden van een Android service. Een Android service is een component dat door Android aangeboden wordt en zorgt ervoor dat dit onderdeel niet zomaar door de maatregelen van Android afgesloten kan worden. Waarom kunnen applicaties of onderdelen hiervan eigenlijk zomaar afgesloten worden? Android heeft een aantal ingebouwde features die ervoor zorgen dat de applicaties op het apparaat (en dan voornamelijk de applicatie op de voorgrond) soepel draaien. Een van deze features is de garbage collector die ervoor zorgt dat objecten en resources die niet meer gebruikt worden, opgeruimd kunnen worden om meer geheugen beschikbaar te maken voor het huidige proces. Ook heeft het Android Operating System (OS) een component dat draaiende processen en threads af kan sluiten om meer geheugen vrij te maken op het moment dat deze threads of processen geen actieve relatie meer hebben met het voorgrondproces. Om ervoor te zorgen dat de data logger (beschreven in hoofdstuk 6.1) niet afgesloten wordt op het moment dat de gebruiker van het Android apparaat bijvoorbeeld een telefoontje krijgt, maken we hierbij dus gebruik van de Android service.

Een implementatie van een Android service die voor onze data logger nog beter van pas komt, is de zogenaamde IntentService. Deze klasse die door het Android framework aangeboden wordt, zorgt er zelf voor dat alle aanvragen naar de service opgestart worden in een aparte (worker)thread. Op deze manier verstoren de achtergrondwerkzaamheden van de data logger de gebruiker op geen enkele manier meer.

6.2.5 Notification manager

Om gebruikers op de hoogte te houden van gebeurtenissen van een applicatie heeft Android een notificatie manager geïmplementeerd. Deze klasse van het Android framework kan in de notification bar van Android een bericht plaatsen dat een bepaalde status verandering van de applicatie kan weergeven. Op het moment van selecteren van deze notificatie kan verwezen worden naar een bepaald scherm binnen die applicatie. Omdat een aantal functionaliteiten van de applicatie op de achtergrond draaien, is dit dus een erg handige manier om de gebruiker op de hoogte te stellen van de huidige status van deze onderdelen. Hierbij gaat het onder andere over de data logger die aan kan geven of er een actieve verbinding is met het OBD Bluetooth apparaat en of deze verbinding op dit moment gebruikt wordt om gegevens uit te lezen en op te slaan.



Figuur 20: Notification voorbeeld

6.3 Server

Naast de applicatie voor een Android apparaat, is er ook een server gedeelte ontwikkeld. Met dit onderdeel is het mogelijk om gegevens uit te wisselen. De server heeft een vergelijkbare database waarin gesynchroniseerde ritten opgeslagen worden.

Voor de communicatie tussen de applicatie en de server is gekozen om een eigen Application Programming Interface (API) op basis van het HTTP protocol en JSON te ontwikkelen. Om de API zo dynamisch mogelijk te ontwikkelen is er per functionaliteit gekozen voor een eigen functie binnen deze API.

Om gegevens naar de server te sturen, is er gekozen voor het de POST methode van het http protocol. De functionaliteit om gegevens op te halen van de server gaat via de GET methode van dit protocol. Dit protocol is de algemene webstandaard voor communicatie en is daarom gekozen voor de web API.

Als datastructuur is gekozen voor JSON in plaats van XML of een eigen datastructuur omdat JSON als voordeel heeft dat dit eenvoudige dataprotocol kortere berichten genereert dan XML. Voor XML en JSON zijn zowel voor web talen (zoals PHP en ASP.net) en voor Java (wat gebruikt wordt in Android) standaard generators en parsers beschikbaar. Om deze reden is het gebruik van een eigen datastructuur af te raden.

Het verschil in lengte van JSON ten opzichte van XML zit het hem voornamelijk in de dubbele XML tags die gebruikt worden. Deze tags zijn in JSON vervangen door een enkele hoofdtag die hierna niet meer per element herhaald hoeft te worden. Voor de vergelijking tussen deze twee structuren zie onderstaand voorbeeld.

JSON (90 tekens):

```
{"drive_data":[
                {"time":"1305185069635","data":"34"},
                {"time":"1305185107373","data":"13"}
            ]
}
```

XML (132 tekens):

```
<drive_data>
        <time>1305185069635</time>
        <data>34</data>
</drive_data>
<drive_data>
        <time>1305185107373</time>
        <data>13</data>
</drive_data>
```

Tabel 5: XML en JSON vergelijking

Aantal sensor waardes	XML	JSON	JSON verbetering
2	132	90	32%
18.000	1.188.000	666.017	44%

In Tabel 5: XML en JSON vergelijking hierboven is te zien hoeveel tekens het versturen van 2 en 18.000 sensor waardes verschilt bij het gebruik van XML en JSON. Zo is te zien dat zelfs bij het versturen van 2 sensoren JSON al 32% minder tekens gebruikt dan XML. Bij het versturen van 18.000 sensoren (wat gelijk staat aan een rit van 30 minuten) scheelt dit zelfs 44%.

Doordat de applicatie draait op een mobiel platform (met beperkte mogelijkheden) en door de standaard ondersteuning van JSON voor Android, is er gekozen om een API te ontwikkelen op basis van het HTTP protocol (GET / POST) met gebruik van de JSON datastructuur.

7. Implementatie

In dit hoofdstuk zal worden uitgelegd hoe de implementatie van de DSA app en de server applicatie is verlopen. Hierbij wordt vooral ingegaan op in welke mate de implementatie overeenkomt met het ontwerp van de applicatie. Ook is er beschreven hoe het programmeren op een professionele manier voorbereid en verlopen is, welke problemen naar voren zijn gekomen en wat voor ontwikkelomgeving gebruikt is bij het programmeren.

Tijdens het programmeren van de eerder ontworpen onderdelen is zo veel mogelijk een directe implementatie van de functies gemaakt zoals die ontworpen waren. Er zijn een aantal wijzigingen geweest in de manier waarop de implementatie gedaan is om de code overzichtelijker te maken en problemen te verhelpen die zijn ontstaan tijdens het programmeren.

7.1 Code styling

Tijdens het programmeren is er nog eens goed gekeken naar de naamgeving van de onderdelen. Omdat er geprogrammeerd wordt in Java wordt er gestreefd naar zo goed mogelijk de standaard te gebruiken van deze taal. Een van deze veranderingen op het ontwerp is daarom het gebruik van de code style. Hiervoor wordt de zogenaamde camelcase opmaak gebruikt. Deze manier van code opmaak is standaard bij Java maar was tijdens ons ontwerp niet meegenomen. Deze verandering heeft geen effect gehad op de logica of de indeling van de applicatie maar alleen de naamgeving is hierdoor veranderd. Deze verandering is gemaakt om zo veel mogelijk met de standaard van Android en Java te werken en de code ook voor andere developers overzichtelijk te houden. Omdat deze coding standaard al beschreven is door Oracle (zie bronnenlijst) zal hier niet verder op ingegaan worden.

7.2 Ontwikkelomgeving

Als ontwikkelomgeving is er gewerkt met Eclipse. Eclipse is een opensource ontwikkelomgeving waarmee het via een Integrated Development Enviroment (IDE) mogelijk is software te ontwikkelen. Bij het ontwikkelen van een applicatie voor Android op basis van Java is er gekozen voor de Java IDE. Om de ontwikkelomgeving in te richten met specifieke functionaliteiten voor Android ontwikkelaars, is hiervoor een aparte Software development kit (SDK) geïnstalleerd. Om ervoor te zorgen dat Eclipse met de Android SDK samenwerkt, is hiervoor een speciale plug-in beschikbaar, de Android Development Tools (ADT) plug-in. Dankzij deze combinatie is het mogelijk om Android specifieke onderdelen te gebruiken binnen Eclipse die het programmeren en debuggen van Android apparaten mogelijk maakt. Hieronder staat uitgelegd welke onderdelen dit zijn en waar deze voor gebruikt worden.

Figuur 21: Android ontwikkelomgeving

- **Android Debug Bridge (ADB):** dit onderdeel verzorgt de communicatie tussen de ontwikkelomgeving en het te programmeren Android apparaat of emulatie.
- **Dalvik Debug Monitor Server (DDMS):** Dit onderdeel is een verzameling van meerdere functionaliteiten waaronder het uitlezen van de log berichten die Android genereert (Logcat), het verkrijgen van thread en heap informatie van de draaiende processen, de mogelijkheid tot het maken van screenshots en het simuleren van locatie informatie en inkomende oproepen of berichten.

7.3 Versie beheer

Om eenvoudig aan een project te kunnen werken met meerdere personen zonder wijzigingen door te voeren die de wijzigingen van anderen weer ongedaan te maken is er gebruik gemaakt van het versiebeheersysteem SVN. Op deze manier worden alle wijzigingen als aparte versie opgeslagen en is het hierdoor mogelijk om de historie van deze wijzigingen te bekijken en eventueel terug te draaien.

7.4 Javadoc

Tijdens het programmeren zijn alle functies voorzien van Javadoc. Op deze manier zijn binnen Eclipse alle functies voorzien van documentatie bij het aanroepen hiervan. Hierdoor wordt het veel gemakkelijker om de juiste argumenten mee te geven aan een functie en is precies duidelijk wat de functie zal retourneren. Aan het eind van het ontwikkelen kan op deze manier ook een complete documentatie worden gegenereerd waarin de volledige code van de applicatie gedocumenteerd is. De complete Javadoc van het project is te vinden in bijlage D: CD.

7.5 Ontwerp VS Implementatie

Tijdens het implementeren van de applicatie zijn er een aantal dingen veranderd aan het ontwerp. In dit hoofdstuk wordt beschreven welke veranderingen plaats hebben gevonden en wat de reden is van deze veranderingen.

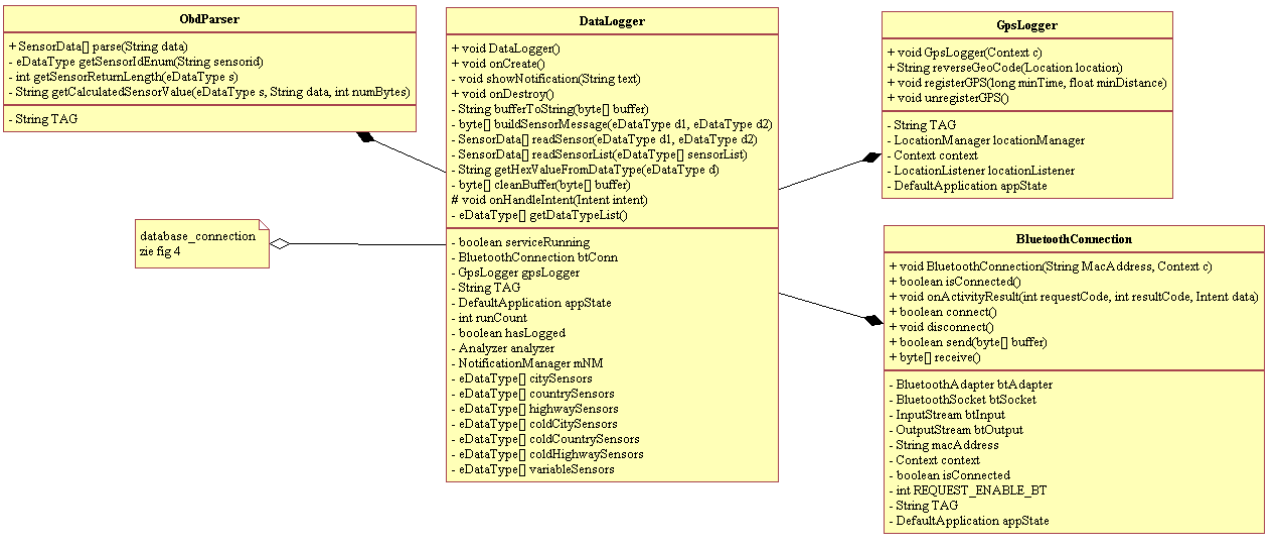
Een van de toevoegingen in bijna alle klassen behalve de user interface is het toevoegen van een “TAG”. Deze TAG bevat de naam van de klasse zodat dit gebruikt kan worden in het loggen van debug berichten. Door deze TAG toe te voegen aan elk debug bericht kan er gemakkelijk achterhaald worden waar het betreffende debug bericht vandaan komt en waar de eventuele foutmelding ontstaan is. Ook kan er op deze manier op een makkelijke manier gescheiden worden welke berichten weergegeven worden in Logcat (de debug console van Android).

Voor het delen van informatie tussen verschillende klassen is tijdens het implementeren gebruik gemaakt van de zogenaamde appState. Deze appState bevat objecten die over de volledige breedte van de applicatie opgeroepen en veranderd kunnen worden. Op deze manier kan de status van bijvoorbeeld de beschikbaarheid van Bluetooth en dat de gebruiker aangegeven heeft een rit te willen loggen op een centrale plek worden bijgehouden.

Omdat op deze manier ook maar één instantie van een object binnen de applicatie kan bestaan, is het gebruik van het singleton ontwerppatroon, wat gebruikt werd voor de database klasse, niet meer van toepassing. Dit mede met het feit dat Android onderdelen van de applicatie zou kunnen verwijderen, is er voor gekozen dit database object in de appState te plaatsen. Hierdoor is de database klasse minimaal veranderd en maakt de deze klasse geen gebruik meer van het singleton ontwerppatroon.

In het grootste gedeelte van de applicatie is er weinig verschil tussen het ontwerp en de uiteindelijke implementatie. Het enige onderdeel waarin redelijk veel veranderd is ten opzichte van het ontwerp is de data logger. Deze klasse heeft een aantal functionaliteiten overgenomen die tijdens het ontwerpen geplaatst waren in de Bluetooth klasse. Het gaat hier om de “readsensor” functies. Deze functies horen eigenlijk niet thuis in deze klasse omdat er al meer gebeurt met de gegevens die via Bluetooth opgehaald worden dan waarvoor deze klasse verantwoordelijk is. Om een goede scheiding aan te brengen tussen het ophalen van data en het interpreteren van data zijn deze functies verplaatst naar de data logger. Op deze manier is de Bluetooth klasse alleen verantwoordelijk voor de communicatie met het OBD apparaat. Ook kan deze klasse nu eventueel vervangen of aangevuld worden met een Wi-Fi klasse om Wi-Fi OBD apparaten te laten communiceren met de applicatie.

In de data logger zelf zijn redelijk veel veranderingen gemaakt om de complete klasse een stuk overzichtelijker te maken. De hoofdfunctionaliteit van deze klasse is niet veranderd maar de manier waarop deze geïmplementeerd is, heeft veranderingen ondergaan. In bijlage C: Technisch Ontwerp is te zien wat het originele ontwerp voor deze klassen was en hieronder in Figuur 22: Klassendiagram data logger staat de uiteindelijk geïmplementeerde klasse.



Figuur 22: Klassendiagram data logger

De status van de data logger wordt bijgehouden door meerdere status booleans in de appState van de applicatie, op deze manier kan de service continu blijven draaien en op basis van deze booleans de juiste taken uitvoeren. De BluetoothConnection klasse is nu alleen maar verantwoordelijk voor het verbinden en versturen/ontvangen van data met het OBD apparaat. Verder zijn er in de data logger een aantal ondersteunende functies bijgekomen in plaats van deze logica in een grote functie uit te voeren.

Bij de andere klassen van het ontwerp zijn minimale veranderingen geweest. Al met al is het gemaakte ontwerp goed overeengekomen met de uiteindelijke implementatie en heeft het minimale moeite gekost om de applicatie te programmeren. Dit omdat tijdens het ontwerpen alle grote beslissingen al zijn genomen.

7.6 Server

7.6.1 Omgeving

Bij TASS zijn enkele servers beschikbaar voor projecten. Voor dit project hebben we toegang gekregen tot een PHP server waarop een MySQL database beschikbaar is. Voor het PHP programmeren van de web API is gekozen voor het gebruik van NetBeans, een gratis ontwikkelomgeving op basis van Java met een PHP IDE. Deze keuze is gemaakt omdat met deze ontwikkelomgeving al eerder gewerkt is. Om de bestanden te uploaden naar de server was enkel het WebDAV protocol beschikbaar. Om gebruik te maken van dit protocol is er gebruikt gemaakt van het programma BitKinex. Om de database en tabellen op te zetten die benodigd zijn voor de web API is gebruik gemaakt van phpMyAdmin.

Figuur 23: Server ontwikkelomgeving

7.6.2 Implementatie

De implementatie van het server gedeelte verliep zonder grote problemen. De ontworpen documentatie functioneerde als een geschikte basis voor het uitwerken van de web API en het self learning system. De API is volledig getest dankzij de mogelijkheid om dit via de browser te testen. Nadat alle testen via de browser succesvol waren, is de web API ook geschikt voor de al ontwikkelde database synchronizer klasse. De database synchronizer functioneerde direct correct met de ontwikkelde web API waardoor er met zekerheid gezegd kan worden dat beide onderdelen van het ontwerp volledig geïmplementeerd zijn.

8. Conclusie

Uit het OBD onderzoek dat tijdens deze afstudeerperiode gedaan is. Waren de mogelijkheden van OBD lager dan verwacht. Vooral het aantal sensoren dat behoort tot de standaard viel tegen. Hierdoor waren de ideeën die van te voren bedacht waren een stuk lastiger te bereiken.

Ondanks tegenvallende onderzoeksresultaten heeft TASS in samenspraak met de afstudeerders besloten het project wel door te zetten. Om alsnog te voldoen aan de wensen van de klant is hiervoor de “Drive Style Analyst” applicatie bedacht. Met deze applicatie is het mogelijk om de zuinigheid en milieuvriendelijkheid van de rijstijl van automobilisten te bepalen. Door scores en tips te geven kan de automobilist vervolgens zijn rijstijl verbeteren met als resultaat lagere brandstof kosten.

Om ervoor te zorgen dat voor alle auto’s de scores vergelijkbaar zijn bij hetzelfde rijgedrag. Zijn er algoritmes bedacht die gebruik maken van normwaardes. Deze normwaardes worden door het zelf lerend systeem op de server bepaald op basis van alle vergelijkbare autoritten. Dit systeem is getest door minstens 10 TASS’ers en is hierdoor steeds preciezer geworden. Het idee is dus uitgewerkt tot een werkend systeem!

Door het uitgebreide onderzoek naar zowel OBD als Android is er voldoende kennis opgebouwd om een degelijk en goed doordacht ontwerp op te zetten. Hierdoor is de implementatie van de applicatie op een snelle en succesvolle manier verlopen. Door veel tijd te besteden aan het opstellen van de requirements en deze te verwerken in het ontwerp zijn alle eisen/wensen van de klant in de ontwikkelde Drive Style Analyst applicatie opgenomen.

Dankzij dit afstudeerproject heeft TASS een duidelijk beeld gekregen wat de mogelijkheden van OBD op dit moment zijn en welke mogelijke toepassingen dit kan hebben binnen het Automotive gebied.

Evaluatie Jeroen van Schelven

Aan het begin van de afstudeerperiode was alles natuurlijk nog nieuw maar al snel voelde ik me thuis bij TASS. Toen we begonnen aan de opdracht was er nog veel onduidelijkheid over het onderwerp “On Board Diagnostics”(OBD). Dit was een van de weinige afstudeerprojecten waarvan TASS zelf ook nog niet wist wat de exacte mogelijkheden waren. Het is ook daarom dat ik deze opdracht ook zo interessant vond.

Gedurende de eerste paar weken is er veel onderzoek gedaan naar OBD en werd er langzamerhand steeds meer duidelijk. Met alle resultaten is na een maand besloten door TASS en ons om het project door te laten gaan, wat tot die tijd nog elke keer de vraag was. Na deze tijd is het ook dat we de exacte formulering van de opdracht hebben vast gesteld.

Omdat ik het vermoeden had dat de technische uitdaging niet heel groot zou zijn van de eerste ideeën die we hadden, zijn we door gegaan met brainstormen tot er ook een leuke technische uitdaging was. Dit samen met de, voor mij, nieuwe Android omgeving heeft er voor gezorgd dat ik veel heb geleerd van deze periode.

Ook de professionele aanpak die we gebruikt hebben bij alle facetten van de opdracht heeft er voor gezorgd dat daar extra leermomenten zijn geweest. Onze begeleider, Tiny Henst, heeft ons prima geholpen en zeer goede feedback gegeven op deze facetten. Ook vond ik het erg prettig dat er extra feedback kwam van Tiny op onder andere deze scriptie zodat ik mijn uiterste best kan doen om mijn opleiding Cum Laude af te sluiten.

Omdat dit een duo afstudeeropdracht was kwam er natuurlijk ook meer samenwerking bij kijken dan andere stages. Ik ken Erwin vanaf het begin van de opleiding en we hebben een vergelijkbare achtergrond. We zijn allebei met MBO begonnen en zo kwamen we terecht bij de HBO-ICT stage van Fontys. Tijdens mijn opleiding heb ik al verscheidene keren samengewerkt met Erwin en dit is me altijd prima bevallen.

Ook tijdens deze afstudeeropdracht is op een paar kleine discussies alles vlekkeloos verlopen. De verdeling van taken en de onderlinge uitwisseling van ideeën heeft altijd geleid tot een beter resultaat. Het is daarom ook dat ik deze afstudeeropdracht met een prettig gevoel af kan sluiten.

Evaluatie Erwin Smeets

De afstudeerperiode bij TASS technology solutions heb ik als prettig ervaren, er was een goede werksfeer en gezellige collega's. De begeleiding van het project was goed verzorgd, mocht je vragen hebben kon je altijd langslopen.

De zelfgekozen afstudeeropdracht bood zowel een technische uitdaging, het On Board Diagnostics(OBD) onderdeel, als een softwarematige uitdaging. Deze uitdagingen zijn zowel softwarematig als technisch ingelost. Na een uitgebreid onderzoek over de OBD is hiermee een complete applicatie gerealiseerd, deze is vanaf de requirements, het ontwerp en de realisatie volledig ontwikkelt voor een Android apparaat. Het werken met Android was een persoonlijke wens van mij, om ervaring op te doen met dit relatief nieuwe platform. Deze wens is meer dan verwacht ingelost.

De afstudeerstage heb ik samen met Jeroen van Schelven doorlopen. Jeroen heb ik leren kennen via de Fontys, tijdens de opleiding hebben we eerder samengewerkt aan diverse opdrachten en projecten. Tijdens het project is de samenwerking positief verlopen. Er zijn weleens wat discussies geweest, maar hier zijn we altijd op een professionele manier uitgekomen. Terugkijkend op de gehele afstudeerperiode is de samenwerking goed geweest.

Gedurende deze afstudeerstage heb ik veelal de kennis toegepast welke tijdens de HBO opleiding aangeleerd was. Denk hierbij aan configuratie management, het goed opstellen van requirements tot aan het uitwerken van een ontwerp tot een daadwerkelijke applicatie.

Dankzij de gezellige collega's en de interessante technologische uitdagingen is deze afstudeerstage leerzaam voor mij geweest. Ik heb bij een groot software bedrijf, op een professionele manier software ontwikkelt en mijn kennis van het Android platform uitgebreid. Hiermee is aan mijn vooraf gestelde doelen voldaan, waarmee voor mij persoonlijk deze afstudeerstage geslaagd is.

Literatuurlijst

Boeken

1. Martin Fowler
UML Distilled, Third edition
Boston MA, Pearson Education, inc 2007
ISBN:0-321-19368-7

2. Reto Meier
Professional Android Application Development, First edition
Indianapolis IN, Wiley Publishing, inc:2009
ISBN: 978-0-470-34471-2

Internet

3. http://developer.android.com/
Google inc.
Referentie materiaal Android
http://developer.android.com/index.html
http://developer.android.com/guide/practices/ui_guidelines/index.html

4. http://groups.google.com/
Google inc.
Programmeurs forum met veel informatie over Android
http://groups.google.com/

5. http://www.achartengine.org/
The 4ViewSoft Company
Ontwikkelaar van een grafiek library voor Android.
http://www.achartengine.org/

6. http://www.cs.otago.ac.nz/
University of Otago (Nieuw-Zeeland)
Model view controller ontwerp patroon voor Android
http://www.cs.otago.ac.nz/cosc346/labs/COSC346Labs-10.pdf

7. http://www.iconfinder.com/
Icon finder
Website waarop gratis iconen zijn te downloaden
http://www.iconfinder.com/search/?q=iconset%3Aiconsweets2

8. http://www.oracle.com/
Oracle
Java code conventions
http://www.oracle.com/technetwork/java/codeconventions-150003.pdf

9. http://www.slimeuml.de/
MVM soft
UML plug-in voor Eclipse
http://www.slimeuml.de/

10. http://www.sqlite.org/
Public domain
SQLite informative en syntax.
http://www.sqlite.org/lang.html

11. http://www.stackoverflow.com/
StackOverFlow
Programmeurs forum met veel informatie over Android.
http://stackoverflow.com/

12. http://www.wikipedia.org/
WikiMedia Foundation
Informatie over OBD
http://en.wikipedia.org/wiki/On-board_diagnostics
http://en.wikipedia.org/wiki/OBD-II_PIDs
informatie over software ontwerp patronen
http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller
http://en.wikipedia.org/wiki/Observer_pattern

13. http://www.youtube.com/
Google inc.
Informatie auto techniek
http://www.youtube.com/user/markjhicks
http://www.youtube.com/user/ADPTraining
Google IO informatie over Android
http://www.youtube.com/user/GoogleDevelopers

Bijlagen

A: Project Management Plan (PMP)

Het volledige project management plan volgens de richtlijnen van TASS zoals opgesteld aan het begin van het project.

B: Software Requirements Specification (SRS)

Een beknopte versie van de software requirements die opgesteld zijn samen met de klant. Hierin staan de functionaliteiten van de applicatie beschreven.

C: Technisch Ontwerp

Het volledige technisch ontwerp zonder bijlagen. Dit document is toegevoegd als referentie voor het hoofdstuk ontwerpen.

D: CD

Een CD waarop de volledige scriptie inclusief bijlagen staat. Ook zijn op deze cd extra documenten en referenties toegevoegd om meer achtergrond informatie te bieden.