TI3800 Bachelorproject - Eindverslag

Jeroen Bareman(4035186), Bas Dado (4033736), Jordy van Kuijk (1320394)

6 juli 2013

eInsight

Bedrijf: Adecs Airinfra
Begeleider TU Delft: Dr. ir. A. J. H. Hidders
Begeleiders Adecs Airinfra: ir. A. van Helden, R. Overvliet
Coördinators Bachelorproject: H.-G. Gross, Dr. Martha Larson

Samenvatting

Dit verslag beschrijft de totstandkoming van de applicatie eInsight. eInsight heeft als doel de dienstenregistratie van vrachtwagenbedrijf Broekmans, gevestigd te Venlo, te digitaliseren waardoor kosten bespaard kunnen worden en de onderneming efficiënter te werk kan gaan. Er is een Android applicatie ontwikkeld voor de werknemers van Broekmans, die werkdagen, activiteiten, sub-activiteiten van werknemers, alsmede de locatie registreert en automatisch opslaat. Hierdoor wordt de registratielast minder en betrouwbaarder. Ook kunnen gebruikers van de mobiele applicatie een digitaal kasboek benaderen. Er is binnen het eInsight software-pakket een website ontwikkeld, genaamd Management Portal, waar alle diensten, die geregistreerd zijn met de Android applicatie, overzichtelijk weergegeven worden. Met behulp van de website kunnen ook activiteiten ingevoerd of aangepast worden, kan het digitaal kasboek van werknemers beheerd worden en kan er urenadministratie plaatsvinden. De Android applicatie en Management Portal voldoen na het einde van het project aan alle eisen zoals die zijn gespecificeerd, het project is succesvol verlopen.

Inhoudsopgave

1 Voorwoord

Dit project is uitgevoerd voor het vak TI3800 - Bachelorproject, onderdeel van de bacheloropleiding Technische Informatica aan de Technische Universiteit Delft. Het product eInsight is ontwikkeld voor Autohulpdienst Broekmans, gevestigd te Venlo, Nederland en is uitgevoerd als onderdeel van onze stage bij Adecs Airinfra, gevestigd te Delft, Nederland. Het project is gestart op 22 april 2013 en is afgerond op 12 juli 2013. Dit verslag documenteert de opdracht, de eisen, ontwerp-keuzes, de implementatie, het algemene verloop van het project en het resultaat.

We willen alle mensen die ons geholpen hebben bedanken, in het bijzonder:

- Andy van Helden, als stagebegeleider en voor de waardevolle begeleiding tijdens het project
- Robert Overvliet, voor de ondersteuning bij het ontwikkelproces
- Jan Hidders, voor de begeleiding van het project
- Martha Larson, voor het coördineren van het Bachelorproject en voor het bijwonen van onze presentatie
- Hans-Gerd Gross, voor het coördineren van het Bachelorproject en voor het bijwonen van onze presentatie

2 Inleiding

Autohulpdienst Broekmans is een bedrijf met ongeveer 25 werknemers dat werkzaam is in Venlo en omstreken. Zij leveren diensten als internationaal transport van bijvoorbeeld auto's, berging van voertuigen, het verhuren van auto's en pechhulp. De administratie van dit bedrijf vertoonde echter substantiële problemen. Adecs Airinfra, een bedrijf dat consultancy en software-tools biedt en werkzaam is in Delft, heeft daarom een oplossing aangeboden om deze te verbeteren: het softwarepakket eInsight. Dit pakket zal Autohulpdienst Broekmans in staat stellen om, via een Android applicatie en een via het web-bereikbare interface, alle activiteiten van werknemers te registreren en analyseren. Ook de uitgaven van chauffeurs kunnen met eInsight worden bijgehouden in een digitaal kasboek.

Werknemers van Autohulpdienst Broekmans zullen de app gaan gebruiken om de door hen uitgevoerde activiteiten te registreren. Deze activiteiten worden vervolgens opgeslagen op een server. De web portal, genaamd Management Portal, biedt het management inzicht in deze activiteiten en de mogelijkheid om analyses uit te voeren op de verzamelde data. Zo kan bijvoorbeeld het aantal gewerkte uren per werknemer worden berekend of kan afgeleid worden waaraan werknemers de meeste tijd kwijt zijn. Ook kan bijvoorbeeld berekend worden of een bepaalde rit winstgevend is geweest, iets dat momenteel niet mogelijk is.

De ontwikkeling van eInsight vormt het bacheloreindproject van Jeroen Bareman, Bas Dado en Jordy van Kuijk. Het is intern uitgevoerd bij Adecs Airinfra gedurende 3 maanden. In dit verslag wordt het volledige ontwikkeltraject van het softwarepakket beschreven. Zo zullen de keuzes die zijn gemaakt tijdens het traject worden toegelicht. Ook zal het ontwerp van het pakket worden beschreven en hoe dit uiteindelijk is geïmplementeerd.

Het gehele proces zal in chronologische volgorde worden beschreven. Zo zal in het eerstvolgende hoofdstuk, hoofdstuk 3, worden ingegaan op de specifieke problemen welke Autohulpdienst Broekmans ondervond. Op basis van de gevonden problemen is een lijst met eisen aan eInsight opgesteld. Deze eisen worden toegelicht in hoofdstuk 4. Zij zijn vervolgens verwerkt tot een ontwerp van de app en web portal. Dit ontwerp wordt besproken in hoofdstuk 5. De volgende stap in het ontwikkelingsproces is de implementatie van hiervan. Deze is toegelicht in hoofdstuk 6. Om te verifiëren dat het ontwerp naar behoren werkt is de software getest. Het test proces wordt beschreven in hoofdstuk 7. In hoofdstuk 8 wordt het resultaat van de laatste fase van het ontwikkelingsproces besproken: de verificatie dat aan alle gestelde eisen is voldaan. Om een idee te geven van de werkwijze gedurende het hele project wordt deze toegelicht in hoofdstuk 9. In hoofdstuk 10 zullen de projectleden afzonderlijk uitleggen hoe zij het project ervaren hebben. Tot slot zal in hoofdstuk 11 nog beargumenteerd worden of eInsight een succesvol project is geweest en zal hierover een conclusie worden getrokken.

3 Probleemanalyse

De problemen waarvoor in dit project een oplossing moet worden gevonden hebben te maken met activiteitenregistratie. Om de keuzes die zijn gemaakt te kunnen onderbouwen zal in dit hoofdstuk een analyse worden gedaan van de huidige situatie bij de klant. Vervolgens zullen de problemen in die huidige situatie worden beschreven.

3.1 Huidige situatie

De activiteitenregistratie binnen Broekmans gebeurt via zogenaamde rittenstaten. Op een rittenstaat wordt, met de hand, alle plaatsgevonden activiteiten met de start- en eindtijden, gebruikte auto's en kilometerstanden bijgehouden. Ook het aantal gewerkte uren en het totaal aantal afgelegde kilometers worden ingevuld. Ook eventuele betalingen die zijn gedaan worden op een dergelijke rittenstaat geschreven. Voorbeelden van dergelijke rittenstaten zijn te vinden in de bijlage 19.

Het berekenen van het aantal uit te betalen uren gebeurt met behulp van zogenaamde maandstaten. Hierin vult een werknemer per dag zijn gewerkte uren in. De boekhouder voert deze informatie in in een Excel spreadsheet. Deze spreadsheet zou naast de rittenstaten moeten worden gelegd als een vorm van controle. Vervolgens moet de informatie in een andere tool worden ingevoerd, welke eventuele toeslagen bepaalt.

Een ander belangrijk deel van de administratie binnen Autohulpdienst Broekmans is het verantwoorden van de gemaakte kosten. Aan werknemers van de Autohulpdienst Broekmans wordt vaak contant geld meegegeven voor het geval er kosten moeten worden gemaakt vanuit het bedrijf. Ook heeft elke werknemer betaalkaarten van het bedrijf. Voor alle kosten die worden gemaakt moeten bonnetjes worden meegenomen zodat de kosten kunnen worden gedeclareerd.

3.2 Probleembeschrijving

Het verwerken van alle ingevulde rittenstaten kost erg veel tijd. Hierdoor gebeurt de controle van ingevulde gegevens slechts zeer minimaal. Dit heeft tot gevolg dat werknemers hun rittenstaten onvolledig in gaan vullen: er wordt immers toch niet naar gekeken. Als gevolg hiervan heeft Autohulpdienst Broekmans geen manier om bij te houden waar zijn werknemers mee bezig zijn, informatie die cruciaal voor een manager. Op het moment dat dit project van start ging was er zelfs een administratieve achterstand van ongeveer een jaar.

Ook de maandstaten, waarop het salaris gebaseerd is, worden niet volledig gecontroleerd. Dit omdat hier simpelweg geen tijd voor is, met als gevolg dat het gemakkelijk wordt om fraude te plegen. En zonder volledige rittenstaten is er geen manier om erachter te komen of een werknemer te veel boekt, of altijd zijn gewerkte uren verplaatst zodat ze net binnen toeslag tijden vallen.

Met het bijhouden van het kasboek ondervond Autohulpdienst Broekmans vergelijkbare problemen. Omdat gemaakte kosten verspreid over rittenstaten of helemaal niet zijn terug te vinden is het vrijwel onmogelijk om deze te verantwoorden. Ook de stapel met bonnetjes van gemaakte kosten is zodanig omvangrijk dat een bepaalde transactie niet meer terug te vinden is. Bovendien raken bonnetjes snel kwijt.

4 Eisen

In dit hoofdstuk wordt gekeken naar de eisen die gesteld zijn aan de portal en de app. Dit zijn de eisen die gesteld zijn door de klant op basis van de problemen die gevonden zijn in het hoofdstuk probleem analyse (hoofdstuk 3). De technische eisen die zijn afgeleid uit de door de klant gestelde eisen zijn terug te vinden in de derde paragraaf van dit hoofdstuk.

In overleg met de klant is het idee ontstaan om een smartphone-app te maken die kan helpen bij het verminderen van de gevonden problemen. Deze smartphone app kan automatisch de begin en eindtijden van activiteiten registreren met een druk op de knop. Omdat de meeste smartphones tegenwoordig over een GPS chip beschikken kan ook de locatie waarop start- en eindtijden worden vastgelegd bijgehouden worden. Dit is handig bij het controleren van de correctheid van de ingevoerde start en eind tijden. Ook kan met behulp van een dergelijke app het kasboek worden bijgehouden. Om het management inzicht te geven in de door de app verzamelde data is een web front-end nodig. Deze kan vervolgens automatisch alle gewerkte uren uitrekenen op basis van de met de app ingevoerde start- en eindtijden en werkzaamheden. Ook kunnen de verlof-, vakantie- en ziektedagen met een web portal makkelijk worden bijgehouden.

4.1 App

Er is in overleg met de klant besloten om een Android-applicatie te maken. Dit omdat Android een veel gebruikt smartphone operating system is waardoor er veel betaalbare telefoons voor te verkrijgen zijn. Om een goed inzicht te krijgen aan de eisen die de klant aan de app stelt maken we gebruik van het MoSCoW-requirements model.

Must have	De mogelijkheid om alle uitgevoerde activiteiten te registreren.
	Cash transacties en het huidige verwachte saldo moeten inzichtelijk zijn
	Cash, Bankpas, Creditcard, DKV en Schreurs pas transacties moeten kunnen worden toegevoegd
	Als er geen internetverbinding is moet het toch mogelijk zijn de app te gebruiken.
	De app moet werken op de Motorola Defy+
Should have	Bijhouden van de locatie van de gebruiker wanneer deze activiteiten of werkdagen start (en/of eindigt)
Could have	Automatische synchronisatie wanneer er een internetverbinding beschikbaar is
	Mogelijkheid om de app te gebruiken op andere telefoons dan de Motorola Defy+
Would like	Registratie van defecten aan auto's

4.2 Portal

Het web-gebaseerde *back-end* van de applicatie, oftewel de Management Portal, heeft als doel alle geregistreerde data, die naar de database gestuurd is vanuit de Android-applicatie, te tonen, bewerkbaar, en inzichtelijk te maken. Zoals toegelicht in het plan van aanpak (Bijlage I, is er na de oriëntatie-fase een bezoek gebracht aan de opdrachtgever. Voorafgaand aan dit bezoek zijn er

mockups en prototypes gebouwd om de klant een beter idee te geven van hoe de applicatie eruit zal zien. De mockups in bijlage C (sectie 14) tonen hoe de verschillende schermen opgebouwd zijn. De schermen zijn, in samenspraak met de projectbegeleider, opgesteld de hand van een eerder opgestelde lijst van eisen. Aan de hand van het MoSCoW-requirements model, zijn de volgende functionaliteiten als doel gesteld:

Must have	De database en de Management-Portal moeten kunnen draaien op een door de opdrachtgever aangeschafte Windows-server
	De weergegeven activiteiten en uren- en ziekte-data moeten kunnen worden geëxporteerd naar een Microsoft Excel bestand
	Er moet een digitaal kasboek aanwezig zijn, waarin overschrijving met contant geld, DKV passen, creditcards en Schreurs tankpassen inzichtelijk zijn
	De door het personeel geregistreerde uren, toegekende toeslagen en overuren moeten berekend kunnen worden
	Registratie van ziekteverzuim en verlofuren van werknemers
Should have	De mogelijkheid om het wagenpark (auto's en aanhangwagens) te beheren
	De mogelijkheid om personeelsleden toe te voegen
	De mogelijkheid om DKV- en Schreurs passen toe te wijzen aan voertuigen
	De mogelijkheid om creditcards toe te wijzen aan werknemers
	De applicatie moet authenticatie gebruiken
	De applicatie kan omgaan met verschillende soorten gebruikersaccounts: administrator, kantoor-medewerker en standaardgebruiker
	Ondersteuning van de volgende browsers: Internet Explorer 9, Google Chrome
Could have	Het registreren van defecten aan auto's en of deze al dan niet opgelost zijn
Would like	Het plannen van activiteiten

4.3 Technische eisen

Uit de eisen die de klant heeft gesteld aan de app en management portal kunnen een aantal technische eisen afgeleid worden. Ook deze zullen in het MoSCoW requirements model geanalyseerd worden.

Must have	Activiteiten moeten verzonden kunnen worden naar de centrale database
	Volledige functionaliteit op Android 2.3.7 (dit is de Android versie op de Motorola Defy+)
	Ondersteuning van de volgende browsers: Internet Explorer 9, Google Chrome
Should have	De applicatie wordt ontwikkeld met behulp van het *Model, View, Controller* design-patroon
Could have	Versleuteld opslaan van wachtwoorden
	Paginering voor pagina's waar veel data in kan komen (zoals het kasboek)
Would like	Foutenlog voor administratoren
	Opslaan van logboek van communicatie tussen de server en de Android app
	Ajax (Asynchronous JavaScript and XML) oproepen voor schermen die modaal ingeladen worden

5 Ontwerp

De app en website hebben allebei dezelfde visuele gebruikersbeleving. Het design van de visuele interface, hetgeen wat op het computerscherm getoond wordt, stond bij opdrachtgever hoog in het vaandel. Er zijn handmatig plaatjes voor alle knoppen gemaakt die dicht bij belevingswereld van de gebruiker staan. De kleur van de opdrachtgever, Broekmans groen, is gekozen als uitgangspunt. Groen geeft de belangrijkste accenten, aangevuld met grijstinten om ervoor te zorgen dat de visuele interface niet overweldigend wordt.

Dit hoofdstuk is onderverdeeld in 3 secties. In paragraaf 5.1 wordt het ontwerp van de app toegelicht. paragraaf 5.2 en paragraaf 5.3 behandelen respectievelijk het ontwerp van de database en website.

5.1 App

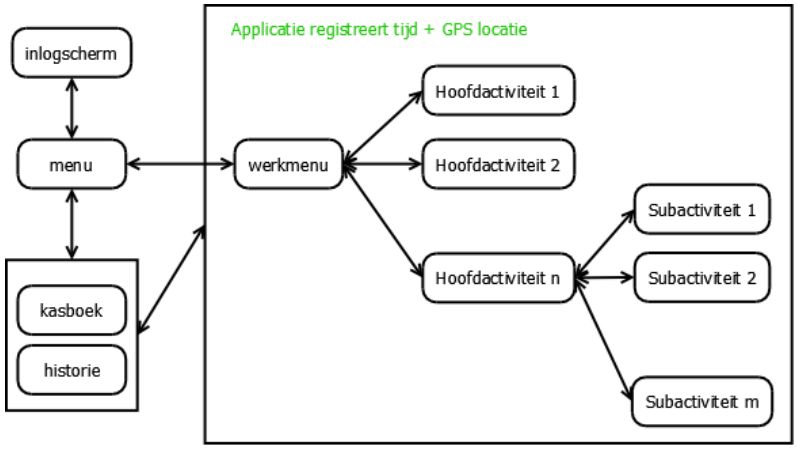
Weinig mensen worden 's ochtends wakker met het voornemen om de hele dag enquêtes in te vullen. Maar dat is eigenlijk wel wat de applicatie is, één grote enquête. Tijdens een werkdag vul je bij elke dienst lijstjes in. De applicatie had aan alle requirements voldaan als er een groot spreadsheet was gemaakt met alle diensten en eigenschappen van diensten. Maar om ervoor te zorgen dat de gebruikersbeleving van de applicatie die van een enquête overstijgt, zijn bij het ontwerp van de applicatie de volgende richtlijnen nageleefd:

- Maak alles zo simpel mogelijk
- Maak gebruik van interactie
- Zorg dat de gebruiker niet verdwaald
- Wees niet te streng

5.1.1 Maak alles zo simpel mogelijk

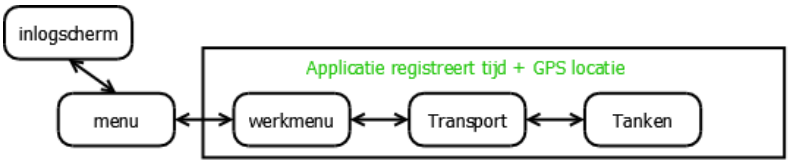
Er is gebruik gemaakt van de kracht van de beperking, er zijn zo min mogelijk verschillende opties per keer weergegeven. Als een gebruiker keuze heeft uit zes in plaats van 30 opties, zal hij bij zes opties meestal sneller weten welke keuze bij zijn intentie past.

Om alle functionaliteit van het programma te bieden en tegelijkertijd zo min mogelijk in beeld te hebben, is er gebruik gemaakt van 56 schermen, beeldvullende interfaces. Elk scherm is binnen maximaal vier acties te bereiken en de bevat gemiddeld zeven knoppen. In figuur 1 zijn de schermen schematisch weergegeven. Om alle verschillende schermen toe te lichten volgt hier een



Figuur 1: Ontwerpmenustructuur

voorbeeldroute van inlogscherm naar een subactiviteit. Dit voorbeeld is ook visueel weergegeven in figuur 2



Figuur 2: Voorbeeld route van inlogscherm naar subactiviteit.

Nadat er is ingelogd via inlogscherm kom je in het menu. Hier kan je kiezen een werkdag te starten of statistieken te bekijken in het kasboek- of (dienst)historiescherm. In het werk-menu kan er een hoofdactiviteit gekozen worden, bijvoorbeeld transport. Het hoofdactiviteitscherm transport geeft alle verschillende subactiviteiten weer die bij transport horen, bijvoorbeeld tanken. Een subactiviteit scherm is gevuld met invoervelden. In de tanken subactiviteit zijn er vakjes voor liters, bedrag en betaalmethode. Een volledig overzicht van alle hoofd- en subactiviteiten is te vinden in figuur 3.

De applicatie registreert de tijd en GPS-locatie op elk moment dat van scherm gewisseld wordt. Het invullen van de enquête wordt hierdoor eenvoudiger omdat de gebruiker niet zelf deze tijden hoeft in te vullen. Bovendien is er hierdoor meer plek per invoerscherm voor de andere velden.

5.1.2 Maak gebruik van interactie

Voor de interactie met de gebruiker zijn 56 schermen en pop-ups gebruikt. Als de gebruiker een activiteit start, toont de applicatie een nieuw scherm. Dit geeft de gebruiker feedback van het drukken op de knop. De gebruiker zal door de aandachtsverschuiving die hierdoor ontstaat hopelijk direct aangezet worden om lege invoervelden in te vullen. Als er op een een invoerveld wordt gedrukt, wordt er een pop-up getoond waarmee het veld kan worden gevuld. Deze pop-up kan een keuzelijst zijn of een tekstbox bevatten. In het geval van een tekstbox zal er dus ook een klein toetsenbord op het scherm verschijnen om de nodige informatie in te vullen. Door bij elke actie van de gebruiker wat nieuws op het scherm te tonen hopen we de enquêtes wat tot leven te brengen.

5.1.3 Zorg dat de gebruiker niet verdwaalt

Om te voorkomen dat een gebruiker niet weet hoe hij naar een bepaald scherm moet komen, hebben we overgang van schermpje naar schermpje beperkt tot een boomstructuur, afgebeeld binnen de rechthoek in figuur 1. Vanuit het werkmenu kan met één klik een van de hoofdactiviteiten schermen bereikt worden. Vanuit een hoofdactiviteit kan met één klik een van de subactiviteit schermen of het werkmenu berijkt worden. Vanuit een subactiviteit kan met één klik worden teruggekeerd naar de bijbehorende hoofdactiviteit.

Het werkmenu, de hoofdactiviteiten en subactiviteiten zijn de 3 lagen waarin de schermpjes zijn onderverdeeld. Schermwisselingen zijn alleen toegestaan van laag naar laag. De boomstructuur is binnen de informatica een bekent begrip en we gaan ervan uit dat na een korte gewenning de gebruiker dit als natuurlijk zal beschouwen.

Elke menulaag is voorzien van een een eigen stijl zodat de gebruiker direct weet of hij in het menu-, werk-, hoofdactiviteit- of subactiviteit scherm is. Er is een plaatje van de ontwikkeling van de afwijkende gebruikers interfaces weergegeven in Bijlage A Bovenaan elk schermpje is bovendien het navigatie pad weergegeven als broodkruimels(menu >hoofdactiviteit >subactiviteit), zodat de groep gebruikers die hieraan gewend zijn dit direct kunnen gebruiken.

5.1.4 Wees niet te streng

Er is rekening gehouden met fouten van de gebruiker door een undo knop in elk scherm te bouwen. Als er per ongeluk ergens op geklikt is of een dienst onder een verkeerde naam is opgeslagen kan de undo knop gebruikt worden. Diensten wijzigen en toevoegen kan alleen via de website en enkel gebruikers die administrator rechten hebben zijn hiertoe in staat. Hier is voor gekozen, omdat de urenberekening is gebaseerd op de met de app geregistreerde activiteiten. Het is niet gewenst dat werknemers hun eigen werkdagen kunnen aanpassen omdat dit fraude eenvoudiger zou kunnen maken.

5.2 Database

In dit hoofdstuk wordt beschreven welke data wordt opgeslagen en waarom gekozen is om die data op te slaan. De daadwerkelijke structuur van de database en de gebruikte technieken staan beschreven in het hoofdstuk over implementatie (paragraaf 6.2). Om het geheel overzichtelijk te houden wordt een scheiding gemaakt tussen informatie die te maken heeft met activiteiten, het kasboek en algemene informatie.

5.2.1 Activiteiten

Tijdens het eerste gesprek met de klant is ontdekt dat per activiteit andere data moet worden bijgehouden. Zo moet bij een tankbeurt bijvoorbeeld worden bijgehouden hoeveel liter er is getankt en voor welk bedrag en moet er bij het uitgeven van een huurauto worden bijgehouden om welke huurauto het gaat. Er is besloten om een onderscheid te maken tussen hoofd- en subactiviteiten, zodat gemakkelijk een overzicht kan worden verkregen van waar werknemers over het algemeen mee bezig zijn, en het toch mogelijk is om een gedetailleerder overzicht te krijgen van de activiteiten van werknemers. Er zijn in totaal 7 hoofdactiviteiten: Transport, Berging, Kantoor, Werkplaats, Huurauto, Divers en Pechhulp. Elke hoofdactiviteit heeft een andere set met subactiviteiten die elk hun eigen velden hebben. Een overzicht van alle hoofd- en subactiviteiten en welke velden er moeten worden opgeslagen is te vinden in figuur 3. Over dat schema is tijdens het tweede gesprek een overeenstemming bereikt met de klant, en het database ontwerp is hier dus op gebaseerd.

Voor de urenberekening moeten ook de werkdagen worden bijgehouden. Hierbij zijn vooral de start en eindtijd belangrijk. Er is besloten om de locatie ook bij te voegen. Op die manier kan bijvoorbeeld ontdekt worden wanneer werknemers te laat, als ze bijvoorbeeld al thuis op de bank zitten, een werkdag afsluiten of te vroeg een werkdag beginnen. Omdat niet alle werknemers een full-time contract hebben moet voor de urenberekening ook het huidige contract van elke werknemer worden opgeslagen.

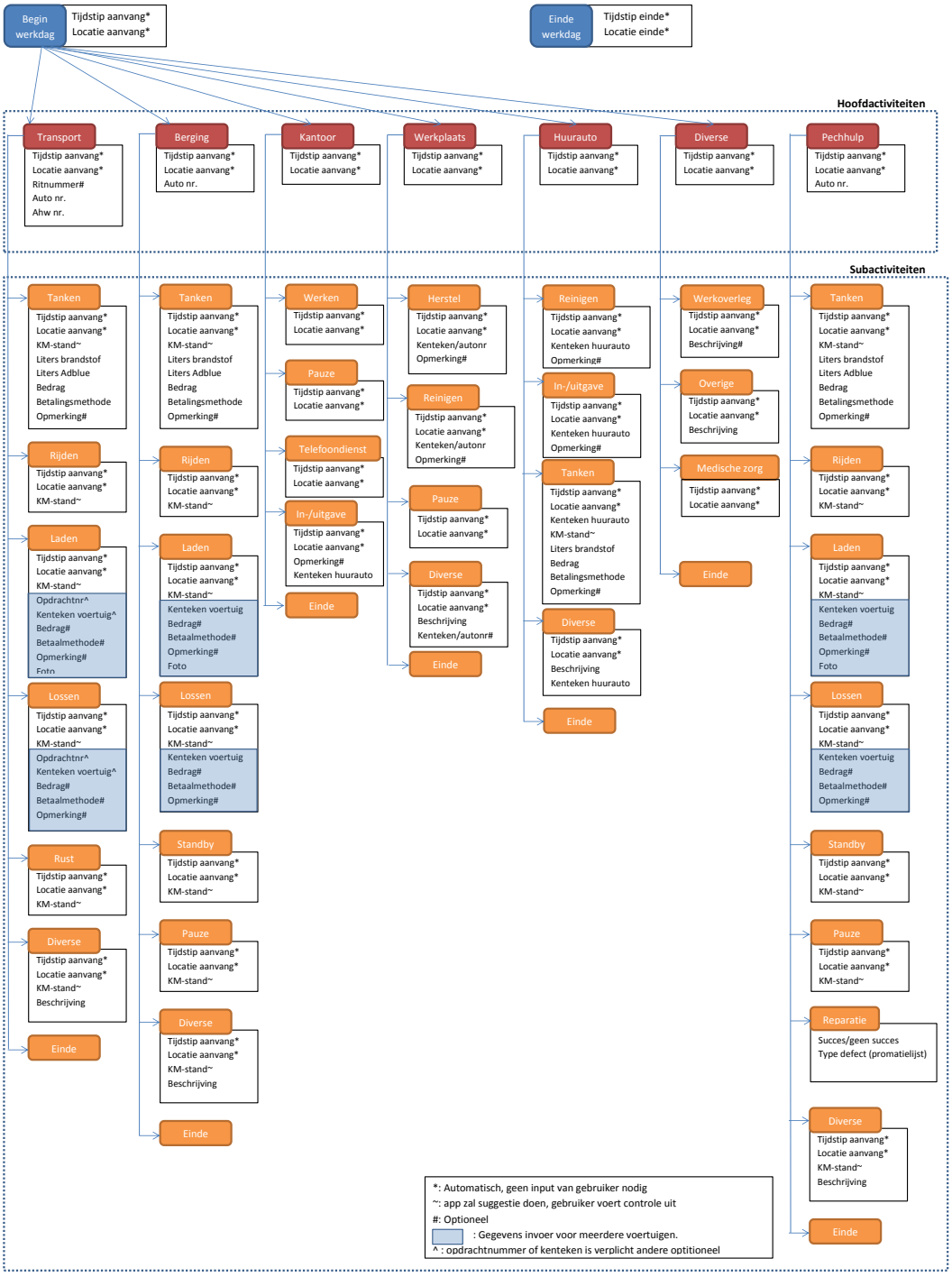
Buiten standaard werkdagen met activiteiten zoals hierboven besproken, zijn er ook nog declarabele uren zoals verlofdagen, ziekte, vakantie etc. Ook deze moeten opgeslagen kunnen worden. Wat hierbij belangrijk is is dat Autohulpdienst Broekmans een bedrijf is dat 24/7 actief is. Als een werknemer dus verlof boekt op een bepaalde dag betekend dat dat hij de gehele dag vrij verwacht. Het aantal te registreren uren moet dus opgeslagen worden bij verlof, omdat van de 24 uur verlof die een werknemer krijgt slechts 8 uur binnen de werkdag vallen.

5.2.2 Kasboek

Het kasboek is in de basis een lijst met transacties per werknemer. Voor elke transactie moet in de database worden bijgehouden door wie en wanneer de transactie is gedaan. Ook moet het bedrag worden bijgehouden en moet het mogelijk zijn een opmerking toe te voegen om aan te geven waaraan het bedrag is uitgegeven.

Binnen het bedrijf zijn verschillende soorten betalingsmethoden actief. Hieronder staat een lijst van de verschillende betalingsmethoden en hoe deze gebruikt worden:

- Contant: werknemers krijgen contant geld mee in een persoonlijke portemonnee. Als ze kosten moeten maken vanuit het bedrijf kunnen ze die portemonnee hiervoor gebruiken. Er



Figuur 3: Overzicht van alle attributen binnen de hoofd- en sub-activiteiten die moeten worden bijgehouden.

moet dus per werknemer worden bijgehouden welke transacties hebben plaatsgevonden. In deze portemonnee kan door de werkgever zowel geld uitgenomen als bijgestort worden. De werkgever kan ook besluiten om het bedrag in de portemonnee te controleren (en daarmee te bevestigen dat elke transactie verantwoord is). Deze controle momenten moeten ook opgeslagen worden met daarbij het moment van controleren en of het verwachte bedrag aanwezig was of niet.

- Creditcards: Werknemers hebben ook creditcards en bankpassen waarmee ze kunnen betalen. Ook dit moet ingevoerd kunnen worden. In principe kan bij bankpassen en creditcards alleen geld worden afgenomen en niet worden gestort.
- DKV kaart: Om te tanken kunnen werknemers DKV kaarten gebruiken. Elke auto in het bezit van Broekmans heeft een eigen DKV kaart. Anders dan creditcards zijn deze dus gekoppeld aan de auto en niet aan de persoon. Ook bij DKV kaarten kan in principe alleen geld afgeschreven worden.
- Schreurs tankpas: Dit is een alternatief voor de DKV kaart. Net als de DKV kaart bestaan er ook tankpassen per auto en kan ook hier alleen geld afgeschreven worden. Ook Schreurs tankpassen kunnen alleen gebruikt worden om geld te betalen. Er kan dus geen geld op gestort worden.

5.2.3 Algemene informatie

Ten eerste moet natuurlijk een lijst met werknemers worden bijgehouden met hun gebruikersnamen en wachtwoorden. Hierbij dient het wachtwoord op een veilige manier opgeslagen te worden zodat het niet mogelijk is om onrechtmatige toegang tot het systeem te verkrijgen. Om duidelijk te maken welke werknemer welke gebruikersnaam heeft moeten ook de voor- en achternaam worden opgeslagen.

Zowel in het kasboek als bij de activiteiten is een lijst met auto's nodig. Elke auto binnen Autohulpdienst Broekmans heeft een nummer toegewezen gekregen. Deze kunnen dus prima gebruikt worden om de lijst met auto's in de database op te slaan. Dankzij deze lijst is het voor een gebruiker gemakkelijk om de juiste auto te selecteren in bijvoorbeeld de app. Ook de aanhangers zijn bekend en hebben dergelijke interne nummers. Zij kunnen dus ook zonder problemen in de database worden opgeslagen.

In tegenstelling tot gewone auto's en aanhangers is de lijst met huurauto's niet bekend. De klant verhuurt namelijk ook soms auto's voor andere partijen. De lijst met auto's die verhuurt kunnen worden is dus niet van tevoren bekend.

5.3 Portal

Dit hoofdstuk beschrijft de zogenaamde Management Portal, de web-gebaseerde interface die de data afkomstig van de Android-applicatie weergeeft. De Portal heeft de volgende doelen:

- Het inzichtelijk maken van de geregistreerde activiteiten van de werknemers.
- Het bewerken, toevoegen en verwijderen van activiteiten van werknemers.
- Het inzien en beheren van een kasboek.
- Het bijhouden van verlof- en ziekte-uren.

Om bovenstaande doelen te bereiken zijn tijdens het ontwerp vaak bestaande documenten van autohulpdienst Broekmans gebruikt als uitgangspunt. Zo is bijvoorbeeld het maandoverzicht van de urenadministratie sterk gebaseerd op een formulier dat is aangeleverd door Broekmans. De uiteindelijke versie van de mockups zijn te vinden in Bijlage C.

Er zijn vier belangrijke schermen te onderscheiden in de applicatie, waarvan het ontwerp in dit hoofdstuk verder toegelicht zal worden: het activiteitenoverzicht, de urenadministratie, het

kasboek en de data-scherm. Bijkomend zijn er het inlog-scherm en het menu, die apart besproken worden. Voordat er ingegaan wordt op het ontwerp van deze schermen is het noodzakelijk kort het (algemeen) design van de applicatie toe te lichten, omdat de keuze van het design in sommige gevallen heeft geleid tot keuzes in het ontwerp van de schermen.

5.3.1 Design

De Management Portal maakt gebruik van het Twitter Bootstrap framework [2]. Er zitten in dit framework een aantal modules en componenten die schermen binnen onze applicatie voor een deel hebben vormgegeven of in ieder geval mee hebben gespeeld in het ontwerpproces van de applicatie. Denk hierbij aan *breadcrumbs*, zogenaamde *modals* (pop-ups) en de *navbar* (navigatiebalk). Het navigatiemenu van de applicatie is gebaseerd op de navigatiebalk uit het framework. Er is tijdens het ontwerp-proces gekeken naar welk "gereedschap"we tot onze beschikking hadden om deze vervolgens te integreren in ons ontwerp.

Hoewel de Bootstrap bibliotheek een goede basis biedt voor welbekende componenten binnen de applicatie (menu's, knoppen, iconen, breadcrumbs, etc.), kunnen in sommige gevallen andere bibliotheken goed te pas komen. Een gebruiker zal bijvoorbeeld moeite hebben met het invoeren van een datum als deze gepresenteerd wordt in het UNIX datetime formaat ("dd-MM-YYYY HH:mm:ss"). Er bestaan bibliotheken die dit probleem verhelpen door een component aan te bieden dat het invoeren van datums vereenvoudigt. Voor dit soort problemen hebben we de bibliotheken gebruikt die weergegeven zijn in tabel 1.

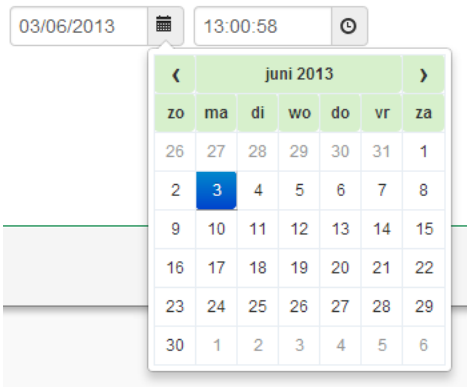
Naam bibliotheek	**Functie**
Bootstrap	Algemeen sjabloon voor de applicatie
Bootstrap Datepicker	Component dat het invoeren van datums vereenvoudigt
Bootstrap Modal	Component dat het bestaande modale venster in Bootstrap vervangt door een iets geavanceerdere variant met meer mogelijkheden.
Bootstrap Select	Select-elementen worden niet veranderd door het Bootstrap framework. Om de interface consistent te houden hebben we gebruik gemaakt van deze bibliotheek om de selectie-elementen te stijlen naar het design van Bootstrap.
JQuery NiceScroll	Het vervangen van scroll-balken binnen de browser. Bevordert de consistentie in het design en voegt enkele handige features toe.
Leaflet	Een open-source JavaScript bibliotheek voor interactieve kaarten

Tabel 1: Deze tabel geeft aan welke bibliotheken er zijn gebruikt die van invloed zijn geweest op het ontwerp van de applicatie.

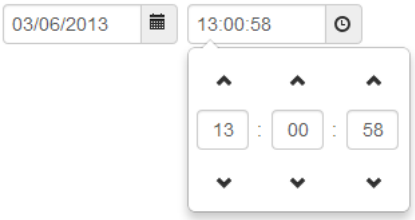
Bij het ontwerp-proces zijn er veel partijen betrokken: de projectbegeleider, de klant en de programmeurs. Het is daarom toepasselijk een set van richtlijnen op te stellen die helpen bij het ontwerpen van de schermen:

- Wees consistent, dit zorgt ervoor dat de gebruiker niet voor verrassingen komt te staan tijdens het gebruik van de applicatie en dat deze zijn of haar kennis van het gebruik van andere toepassingen op deze applicatie kan toepassen.

- Gebruik iconen waar nodig. Een aantal argumenten waarom het gebruik van iconen van goede invloed op het design van de applicatie kan zijn:
  - Effectieve visuele communicatie
  - Makkelijke identificatie van informatie
  - Schermruimte besparen door het vervangen van tekst
- Schenk (voldoende) aandacht aan de plaatsen van de elementen op het scherm.
- Houd rekening met de invoer van data. Maak dit makkelijker voor de gebruikers, als het kan (bijvoorbeeld door het gebruik van *date/time-pickers*).



Figuur 4: Voorbeeld van het *datepicker* component dat de gebruiker helpt een datum in te voeren.



Figuur 5: Voorbeeld van het *timepicker* component dat de gebruiker helpt een tijd in te voeren.

5.3.2 Inloggen

Het inlogscherm is ontworpen om eenvoudig in gebruik te zijn. Het scherm bestaat uit twee invoervelden en een knop om in te loggen. Het logo van de klant, Autohulpdienst Broekmans is ook verwerkt in het scherm, dat er voor zorgt dat de klant de website herkent. Het scherm is vrij eenvoudig opgebouwd en is gemaakt naar een ontwerp dat vaker binnen Adecs Airinfra wordt toegepast. Wanneer de gebruiker probeert in te loggen met incorrecte inloggegevens, krijgt hij of zij feedback hierop in de vorm van een melding die op het scherm verschijnt.

5.3.3 Menu

Het menu speelt een belangrijke rol in het ontwerp van de portal. Het laat de gebruiker navigeren door de voornaamste schermen van de applicatie. In Figuur figuur 6 is een overzicht van de opbouw van het menu getoond.

Figuur 6: Overzicht van de structuur van het menu in de management-portal.

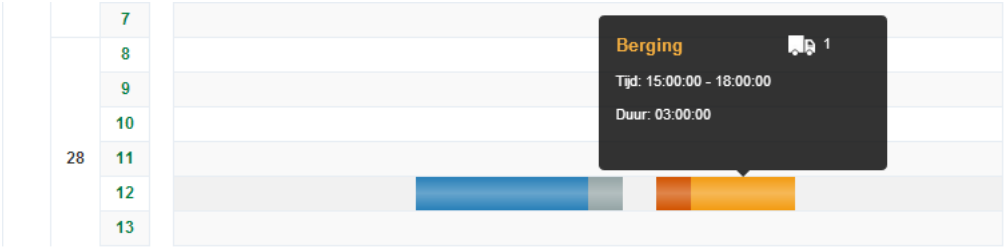
De *eInsight* knop, helemaal links in het menu, fungeert als "home"knop. De gebruiker keert hiermee terug naar de beginpagina van de applicatie. Hierna volgen knoppen voor de vier voornaamste functionaliteiten van het scherm. Elk item in het menu wordt voorafgegaan door een icoon. Dit zorgt ervoor dat de gebruiker makkelijker de menustructuur kan herkennen en onthouden.

5.3.4 Activiteiten

In deze sectie wordt het ontwerp van de schermen die activiteiten en sub-activiteiten weergeven behandeld. De schermen spelen een belangrijke rol in de management-portal, ze maken data van activiteiten, geregistreerd met behulp van de eInsight App, inzichtelijk. Het ontwerp van de schermen doorging tijdens het verloop van het project enkele veranderingen, ook deze worden besproken in dit hoofdstuk.

Overzicht

Het activiteitenscherm, genaamd overzicht, is verantwoordelijk voor het tonen en inzichtelijk maken van informatie over activiteiten die geregistreerd zijn door de eInsight App. Dit scherm bestaat voornamelijk uit een tabel waarin rijen dagen voorstellen en kolommen gebruikers. Een cel binnen de tabel representeert een werkdag van een werknemer. Deze cellen worden ingekleurd op basis van activiteiten die voor die werknemer op die dag zijn geregistreerd. De breedte van de vakjes die activiteiten aanduiden wordt bepaald door de duur van activiteit, gedeeld door de duur van de werkdag. Als een activiteit bijvoorbeeld een halve dag wordt geregistreerd, dan zal er in de tabel een cel zijn die voor de helft is ingekleurd. Het begin van de dag (00:00) wordt voorgesteld door de linker kant van het vakje. De rechterkant stelt 23:59 voor. Dit omdat Autohulpdienst Broekmans een bedrijf is dat 24/7 actief is. De cellen beperken tot alleen werktijden zou dus geen goed overzicht bieden. Door op de legenda knop te klikken kan de gebruiker zien welke kleuren horen bij welke activiteiten. De gebruiker kan door met de muis over de desbetreffende activiteit heen te gaan meer informatie zien over een bepaalde activiteit die heeft plaatsgevonden op die dag. Deze wordt weergegeven in een tooltip. Een voorbeeld hiervan is weergegeven in subparagraaf 5.3.4.



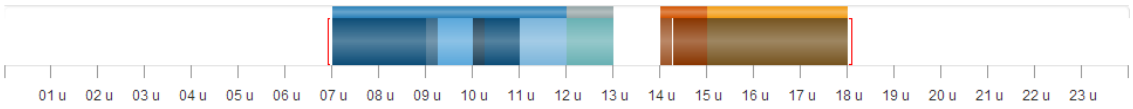
Figuur 7: Een screenshot van een werkdag in het overzicht scherm. Het getal 28 staat voor de week, de andere getallen in het groen representeren de dagen. In dit geval staat de gebruiker van de portal met de muis op het gele vak, wat een *berging*-activiteit is. In de zogenaamde *tooltip* wordt extra informatie weergegeven.

Het scherm biedt ook de mogelijkheid de tijdspanne van het overzicht aan te passen. Het scherm kent vier varianten voor het filteren op tijdseenheden: dagoverzicht, weekoverzicht, maandoverzicht en jaaroverzicht. Bij elke van deze keuzes kunnen de dag, week, maand en jaar respectievelijk worden ingesteld. Ook kan er gefilterd worden op werknemer. Wanneer een werknemer is geselecteerd, wordt alleen data van deze persoon getoond.

Dagoverzicht

Wanneer een gebruiker op een cel in het activiteiten overzicht klikt, verschijnt een scherm met het detailoverzicht van de dag en werknemer die bij de gekozen cel horen. Dit scherm noemen we het dagoverzicht. Hierin zijn alle werkdagen, activiteiten en sub-activiteiten van de gekozen dag zichtbaar met start- en eindtijd en de bijbehorende details. In de mockup die hiervan gemaakt is (zie Bijlage C, is alleen een tabel zichtbaar met die informatie. Echter bleek een dergelijke tabel weinig overzicht te geven in de gewerkte tijden en is besloten om het scherm uit te breiden. In het nieuwe ontwerp staat bovenin het scherm een diagram dat de dag representeert, de zogenaamde tijdlijn. Horizontaal toont dit de uren van die dag. Werkdagen worden getoond als dunne rode balkjes die het begin en het einde van een werkdag aangeven. Activiteiten en sub-activiteiten zijn zichtbaar als gekleurde balken die precies dat deel van de dag overspannen dat ze actief zijn. Wanneer een gebruiker met zijn muis over een dergelijke balk beweegt verschijnt er net zoals in het activiteitenoverzicht een *tooltip* met daarin de details van die activiteit of sub-activiteit. Door het toevoegen van de tijdlijn krijgt de gebruiker snel een idee van hoe de gekozen dag voor de werknemer er heeft uitgezien. Het ontwerp van de tabel is gebaseerd op aangeleverde rittenstaten van de werknemers van Autohulpdienst Broekmans .

Onder dit figuur staat een tabel met daarin een rij voor elke activiteit en sub-activiteit (zie



Figuur 8: Een voorbeeld van een werkdag die weergegeven wordt in de tijdlijn. Kleuren geven aan welke activiteit(en) hebben plaatsgevonden. De rode balken aan de linker- en rechterkant van de activiteiten markeren het begin en einde van een werkdag respectievelijk. De smalle balken die de hoge balken overspannen geven de hoofdactiviteiten aan. In dit voorbeeld zijn er vier hoofdactiviteiten geregistreerd. De balken aan de onderkant stellen sub-activiteiten voor die binnen een hoofdactiviteit hebben plaatsgevonden.

9). De tabel heeft kolommen voor de activiteit of sub-activiteit naam, starttijd, eindtijd, duur, extra beschikbare informatie, en eventuele opmerkingen. Met extra beschikbare informatie worden specifieke velden bedoeld die niet voor alle soorten activiteiten en sub-activiteiten bestaan, zoals het aantal liters brandstof bij een tankbeurt. In elke rij zijn er een knoppen die het mogelijk maken om de activiteit in detail te bekijken te bewerken of verwijderen. Werkdagen worden getoond als "vlaggetjes" die over de activiteiten heen liggen. Als een werknemer veel activiteiten heeft op een bepaalde dag kan er voor gekozen worden om sub-activiteiten te verbergen door op de - knop te drukken. De sub-activiteiten worden dan verborgen in de tabel.

Activiteit	Verrichting	Duur	Starttijd	Eindtijd	Details	Opm.	Opties
+ Transport		05:00:00	07:00:00	12:00:00	(Mercedes, 1, BV-SH-09, Atego) Barents (WF-75-HL)		
- Pechhulp		01:00:00	12:00:00	13:00:00	(Mercedes, 1, BV-SH-09, Atego)		
	Reparatie	01:00:00	12:00:00	13:00:00	Aandrijving/koppeling 117300 Nee AS-12-RT (-)		
- Werkplaats		01:00:00	14:00:00	15:00:00			
	Herstel	00:15:00	14:00:00	14:15:00	(Renault, 4, BR-LX-25, Mascott) ja? (-)		
	Reinigen	00:45:00	14:15:00	15:00:00	(Mercedes, 2, BV-ZS-31, Atego) :((-)		
- Berging		03:00:00	15:00:00	18:00:00	(Mercedes, 1, BV-SH-09, Atego)		
	Stand-by	03:00:00	15:00:00	18:00:00	0		

Figuur 9: Deze tabel is in het dagoverzicht onder de tijdlijn geplaatst (8) en laat de geregistreerde activiteiten en sub-activiteiten in detail zien. De kleuren aan de linkerzijde van de tabel komen overeen met de kleuren in Figuur 8.

Als een gebruiker op een activiteit of sub-activiteit, of op het bekijken symbool (het oogje) in de tabel klikt, verschijnt er een pop-up met daarin een gedetailleerd overzicht van het geselecteerde item. Ook is hierin de locatie van de activiteit, als deze beschikbaar is, zichtbaar op een kaart. Onderin dit scherm staat een aanpassen knop die het mogelijke maakt om alle zichtbare informatie aan te passen. Binnen deze pop-up wordt ook informatie getoond over de sub-activiteiten die vallen onder deze activiteit. Ook kan er een nieuwe sub-activiteit worden toegevoegd vanuit dit scherm.

In zowel het overzicht als het dagoverzicht is er een knop toegevoegd om de weergegeven data te exporteren naar Microsoft Excel. Alle werkdagen, activiteiten en sub-activiteiten worden dan achter elkaar in een Excel document geplaatst met hun bijbehorende informatie. De gebruikers van de Management Portal kunnen deze data verder bewerken in de Excel omgeving.

5.3.5 Urenadministratie

Het registreren van verlof en ziekte zou met een soortgelijk scherm moeten werken als het bewerken en bekijken van activiteiten. Ook hier is een tabel waarin de cellen dagen van werknemers representeren. Alleen is er besloten om hier voor elke werknemer een rij toe te voegen en voor elke dag een kolom. Dit maakt het mogelijk om smallere kolommen toe te voegen waardoor de hele

tabel op een pagina past zonder dat horizontaal scrollen nodig is. Er kan hier geklikt worden op werknemers waarna een overzicht van alle verlof en ziekte activiteiten voor die werknemer getoond wordt. Nieuwe verlof en ziekte uren kunnen toegevoegd worden vanuit dit scherm.

In de mockup was het mogelijk om met een selectie lijst te selecteren om wat voor soort verlof het ging. In het uiteindelijke design is echter gekozen om per type verlof een tabblad te maken. Zo is het makkelijk voor een gebruiker informatie te filteren. Op elk tabblad is een knop om een verlof activiteit van het type dat bij dat tabblad hoort toe te voegen. Als een gebruiker hier op klikt verschijnt een pop-up waarin de start- en eindtijd van het verlof en een opmerking kan worden ingevuld. Ook is er besloten om een invoer veld "te registreren uren" toe te voegen. Dit is nodig omdat als iemand een dag verlof boekt (dus 24 uur) hiervan slechts 8 uur als verlof uitbetaald worden. Het aantal "te registreren uren" is dan dus 8. Ook is er besloten het totaal aantal overuren, compensatieuren en verlofuren per type te tonen in een aparte "overzicht" tabblad en is er een tabblad toegevoegd waar de uitbetaalde uren kunnen worden toegevoegd. Deze worden getoond in de jaaroverzicht tabel.

Het overzichten menu uit de originele mockups is samengevoegd met verlof- en ziekte registratie onder het kopje "urenadministratie". Het jaar- en maandoverzicht in urenadministratie biedt een overzicht van de totaal gewerkte uren per periode voor elke werknemer. Dit overzicht wordt weergegeven als een tabel met daarin hoeveel uren de werknemer gewerkt heeft in de gekozen periode, hoeveel verlof uren hij heeft in die tijd, en hoeveel uren er onder extra tarieven vallen. Ook het aantal overnachtingen wordt hierin getoond. Deze kolommen zijn gebaseerd op de oude administratie tabellen die Broekmans gebruikt voor hun urenadministratie. Zowel het jaar- als maandoverzicht kunnen naar een Excel document geëxporteerd worden.

5.3.6 Kasboek

In het kasboek is voor elke betalingswijze een menu onderdeel toegevoegd. De gebruikersinterfaces die bij deze menu items horen hebben ongeveer dezelfde lay-out. Met name de schermen voor bankpassen, DKV passen en Schreurs tankpassen zijn vrijwel identiek. Beide bestaan uit een tabel met daarin in de eerste kolom alle werknemers (voor bankpassen) of auto's (voor DKV en Schreurs passen). In de tweede kolom staan alle passen naast de werknemer of auto waaraan ze zijn toegewezen. In het bankpassen overzicht is er nog een derde kolom die het type bankpas aangeeft. De laatste kolom toont het saldo van de pas. Wanneer een gebruiker klikt op een werknemer, auto of pas wordt een tabel met daarin alle transacties die bij de gekozen entiteit horen weergegeven.

De kasboek pagina voor contant geld is meer complex omdat het hierin ook mogelijk is om geld bij te storten of in te nemen. Bovendien is het mogelijk om het saldo van een werknemer te controleren, zodat bijgehouden kan worden dat werknemers niet onverantwoord geld uitgeven. Als een gebruiker in het menu op kasboek en daarna cash klikt verschijnt een tabel met alle werknemers, hun huidige saldo, en wanneer de laatste controle heeft plaatsgevonden. In de laatste kolom van deze tabel is tijdens de implementatie fase nog een indicatie toegevoegd die aangeeft of de laatste controle "OK" was.

Wanneer een gebruiker op een van de werknemers in het cash overzicht klikt opent een scherm waarin alle transacties van die werknemer zichtbaar zijn in een tabel. Per transactie staat in deze tabel een rij met daarin de datum, het bedrag en een opmerking als die beschikbaar is. In dit scherm is ook een tabblad waarin alle controles voor de gekozen werknemer inzichtelijk zijn gemaakt. Onderin staan knoppen om geld te storten, in te nemen en het saldo te controleren. Wanneer op een van die knoppen gedrukt wordt, komt er een pop-up naar voren waarin de benodigde informatie ingevuld kan worden.

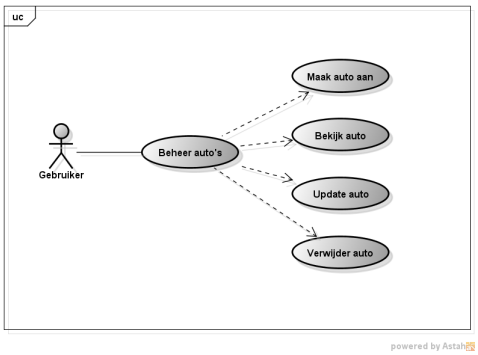
5.3.7 Data

De verzameling van schermen die onder data staan gegroepeerd hebben voornamelijk CRUD (Create, Read, Update en Delete) functionaliteit. Hiermee kan de gebruiker weliswaar data manipuleren die elders binnen het systeem gebruikt wordt. Het betreft voornamelijk statische lijsten, zoals

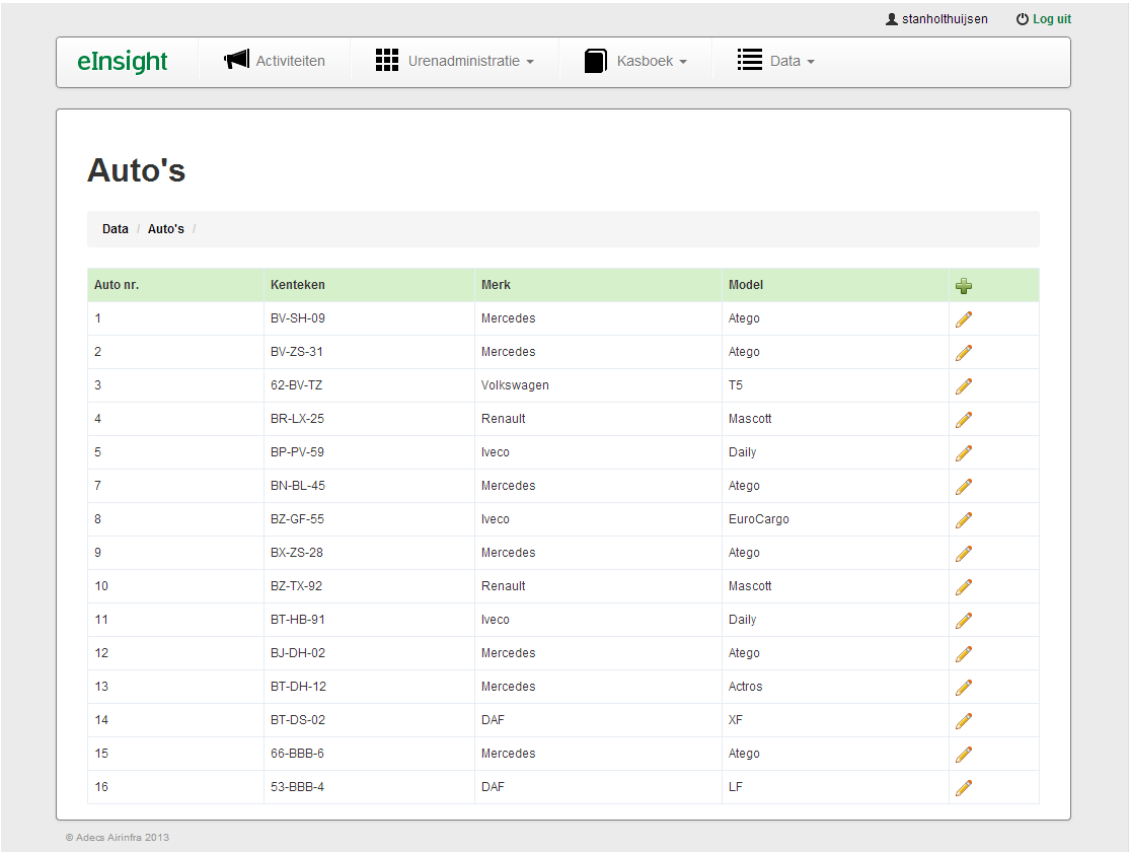
de lijst met werknemers, auto's en aanhangwagens. De schermen *Fouten log* en *Synchronizer log* zijn hier uitzonderingen op en worden alleen gebruikt door de ontwikkelaars.

- **CRUD-schermen**
  Een use case van de crud schermen is gegeven in figuur 10, een voorbeeld van een crud-scherm is te zien in figuur 11. Deze schermen zijn veelal hetzelfde opgebouwd. Bovenaan staat de titel met daarin de naam van het scherm. Daaronder staan de zogenaamde *breadcrumbs*, die navigatie bieden binnen de CRUD-omgeving. Ten slotte presenteert een tabel de data van het model is kwestie. In deze tabel wordt in de header de attributen van het model getoond. In de rijen van de tabel worden vervolgens de records uit de database weergegeven. Wanneer er een nieuw record kan worden toegevoegd, of wanneer de rij in kwestie kan worden aangepast, staat dit aangeduid met een icoon in de laatste kolom van de tabel. Wanneer de gebruiker op een rij in de tabel, of op het aanpassen-icoon klikt, komt deze terecht op een nieuwe pagina. Hierin worden de attributen van het model getoond als invoer-velden en kan de gebruiker aanpassingen doorvoeren of een nieuw record aanmaken.

- **Log-schermen**
  De log schermen hebben als taak het inzichtelijk maken van logs die worden opgeslagen door zowel de synchronisatie van data vanuit de app, als bugs binnen de website. Ze bevatten een tabel met daarin records uit de fouten log en synchronisatie-log respectievelijk. Wanneer de gebruiker op een rij in de tabel klikt, verschijnt er meer informatie over de desbetreffende bug in een pop-up venster.



Figuur 10: Een usecase voor het gebruik van een CRUD scherm.



Figuur 11: Een voorbeeld van een zogenaamd CRUD-scherm voor de lijst Auto's in de management portal.

6 Implementatie

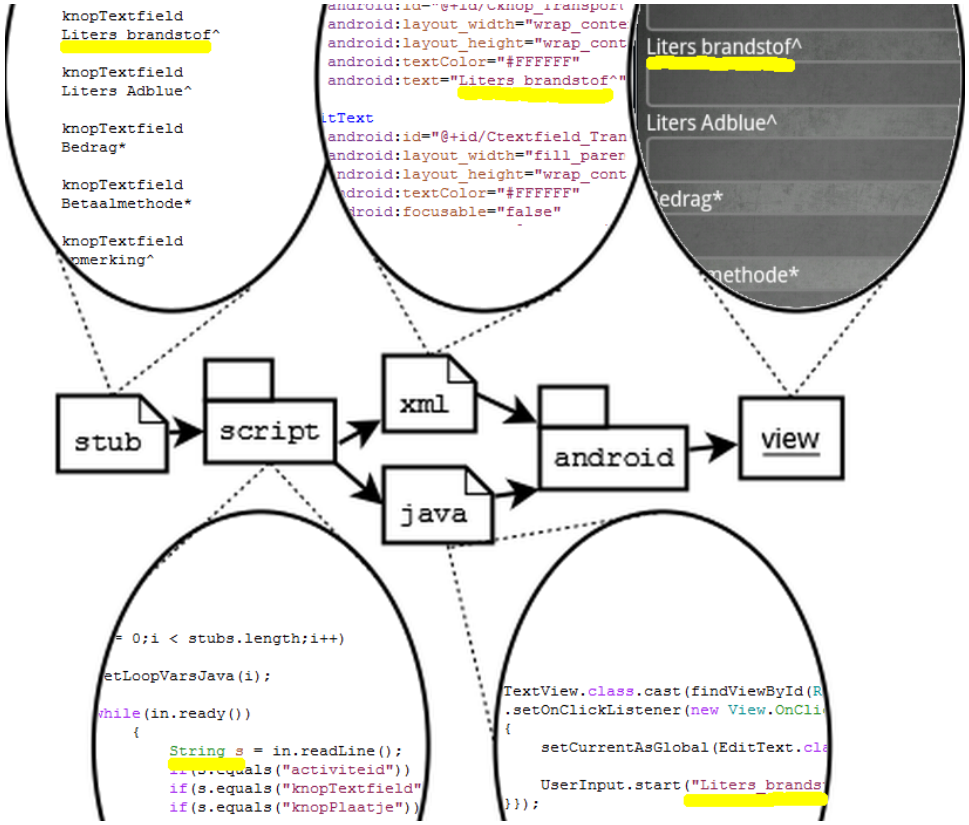
In dit hoofdstuk zal beschreven worden hoe eInsight is geïmplementeerd. Omdat de implementatie te uitgebreid is om volledig in dit verslag te beschrijven zijn alleen specifieke delen in detail uitgediept. Deze delen zijn geselecteerd op hun niveau van complexiteit en kwaliteit van de implementatie. Om het overzicht te behouden is een scheiding gemaakt tussen de implementatie van de app, database, web portal en synchronisatie-service. Voor elk van deze hoofdonderdelen bestaat een paragraaf in dit hoofdstuk. Ten slotte is er nog een paragraaf die in gaat op de feedback van de Software Improvement Group (SIG) en hoe daarmee is omgegaan in eInsight.

6.1 App

6.1.1 52 verschillende diensten

De opdracht was om 52 verschillende diensten bij te houden waarbij voor elke dienst een aantal gegevens ingevuld dienen te worden. Om deze 52 schermpjes te maken is er gebruik gemaakt van een script die met behulp van sleutelwoorden de app schermpjes maakte. In figuur figuur 12 staat een voorbeeld geïllustreerd van een tekstveld op een activiteitenschermpje. Stub is een tekstbestandje gevuld met codewoord, dit wordt gelezen door het script.

Het script zet deze codewoorden om in XML: Android layout definities, en java: Android code. De Android compiler en Android-runtime omgeving zetten dit dan weer om in een schermpje. Met behulp van het script is het mogelijk om in een paar minuten een compleet nieuwe werkzaamheid toe te voegen.



Figuur 12: Van codewoorden naar android schermpje

6.1.2 AndroidOS stopt applicatie

Android OS is een besturingssysteem met een eigen filosofie, dit is te merken als je de applicatie af wilt sluiten, System.exit(0) is afgevangen en werkt niet. Het idee is dat eigenlijk iedere applicatie die draait op het besturingssysteem actief blijft, zodat er snel te schakelen valt. Er kan gebruik worden gemaakt van de home knop om uit de applicatie te gaan en een andere te starten. Je kan dit zien als een soort alt-tab. Een nadeel van deze filosofie is dat als er meer capaciteit nodig is voor de ene applicatie, de andere applicatie wordt afgesloten. Onze applicatie registreert werktijden en kan niet zomaar worden afgesloten. De app is zo geïmplementeerd dat bij elke actie van een gebruiker de huidige staat van de app wordt opgeslagen. Als de app wordt afgesloten en overnieuw wordt opgestart, wordt die staat dan hersteld. De staat van de app wordt bepaald door onder andere: de huidige werkdag, activiteit, subactiviteit, ingevulde tekstvelden, starttijden, GPS-locaties, gebruikersnaam en wachtwoord.

Een voordeel van de best-practise om de app niet af te sluiten is dat als de gebruiker op de afsluitknop drukt, ja iedere gebruiker verwacht een afsluitknop, de synchronisatie met de web-server gewoon doorgaat. De automatische synchronisatie- en GPS-threads sluiten zichzelf na de synchronisatie af om batterij te besparen.

6.1.3 Automatische synchronisatie

Om het voor de gebruiker makkelijker te maken is ervoor gezorgd de synchronisatie automatisch te laten verlopen. De synchronisatie verloopt enkel via wifi zodat er geen databundel nodig is.

De applicatie heeft een asynchrone taak(andere thread) lopen die elke minuut controleert of er wifi verbinding is. Als er wifi en nieuwe data is wordt deze gesynchroniseerd. Na de synchronisatie wacht de taak 2 uur en start zichzelf dan overnieuw. Pseudocode is weergegeven in tabel 2

```
while(!finished)                        app niet is afgesloten
    als(laatstesync ¿ 120 && wifi)      langer dan 2 uur geleden gesyncet en wifi
        synchroniseer
        laatstesync = 0
    anders
        laatstesync ++                  is een minuut bijgekomen
    wacht minuut
```

Tabel 2: Automatische sychronisatie pseudocode

De synchronisatie wordt ook automatisch aangeroepen bij het beëindigen van een werkdag. Dit zorgt ervoor dat de synchronisatie ook plaatsvind als de auto-sync korter dan 2 uur geleden nog gesynchroniseerd bleek. Het leek ons netter het zo op te lossen dan de autosynchronisatie hiervoor te laten pollen. De netste oplossing is om ook gebruik te maken van een on-wifi event, dit is niet uitgezocht omdat het weinig oplevert en na een minuut googelen niet snel implementeerbaar leek.

6.1.4 GPS

Op het moment vullen alle chauffeurs van autohulpdienst Broekmans alle locaties met de hand in. Een harde eis van de app was om GPS te gebruiken zodat de locaties automatisch worden ingevuld. GPS locaties zoeken vraagt veel stroom van de telefoon. De uitdaging van de implementatie is om een zo goed mogelijk signaal te vinden in zo kort mogelijke tijd.

De implementatie van de GPS is als volgt. Als er een activiteit gestart wordt, wordt de laatst bekende GPS-locatie op een locatie-lijst gezet. De locatie-lijst wordt door een selectiemethode gehaald. De selectiemethode sorteert de lijst op nauwkeurig locatie-fix in meters + seconden locatie-fix verkregen na starten activiteit. Als de beste fix niet ouder dan 5 minuten is met een nauwkeurigheid van minimaal 150 meter wordt deze fix geschreven in een positie variabele anders wordt deze variabele leeg gemaakt.

Nadat de selectiemethode is aangeroepen wordt er naar een GPS-signaal gezocht. Er wordt elke 500 milliseconden een update gevraagd totdat: er een locatie gevonden is met een bepaalde nauwkeurigheid binnen een bepaalde tijd. De Java-implementatie van voorwaarden wanneer de gps kan stoppen met zoeken is weergegeven in figuur 13. Als de gps gestopt is wordt nogmaals de selectiemethode aangeroepen zodat de positie-variabele misschien verbeterd wordt als er een betere fix is gevonden. Op het moment dat een activiteit beëindigd wordt, wordt de positie-variabele uitgelezen en toegevoegd aan de activiteit.

```
private static long sec         = 1000;     // 1 seconde = duizend milliseconde
private static long stopTijd10  =  1 * sec; // stoptijd bij vinden signaal
                                            // binnen 10 meter
private static long stopTijd30  = 10 * sec; // stoptijd (...) 30 meter
private static long stopTijd50  = 15 * sec; // stoptijd (...) 50 meter
private static long stopTijd100 = 30 * sec; // stoptijd (...) 100 meter
private static long maxZoektijd = 50 * sec; // max tijd gps aanstaat
private ArrayList<Location> lijst;          // alle gevonden gpslocaties

public void onLocationChanged(Location l)
{
    lijst.add(l);
    Long tijdVerstreken = System.currentTimeMillis() - startTijd;
    if(((lijst.bevatNauwkeurigheid( 10) && (tijdVerstreken > stopTijd10 )) ||
       ((lijst.bevatNauwkeurigheid( 30) && (tijdVerstreken > stopTijd30 )) ||
       ((lijst.bevatNauwkeurigheid( 50) && (tijdVerstreken > stopTijd50 )) ||
       ((lijst.bevatNauwkeurigheid(100) && (tijdVerstreken > stopTijd100)) ||
       (tijdVerstreken > maxZoektijd)) stop(); // stop gps
}
```

Figuur 13: Stuk Java code uit gps implementatie

De GPS-implementatie is zo opgezet dat er makkelijk wijzigingen zijn aan te brengen in de verschillende criteria. In hoofdstuk 16 is een kaartje te zien met verzamelde GPS-locaties tijdens een dienst. En in figuur 14 is te zien hoe de applicatie een locatie vindt.

6.1.5 Portemonnee controle

Tijdens een werkdag van een werknemer kan zijn werkportemonnee gecontroleerd worden. De portemonnee wordt o.a. gebruik voor het afrekenen van benzine. We hebben ervoor gekozen om elke werkdag ongedaan te kunnen maken, hierdoor staan op de website niet de transacties tijdens een werkdag. Dit is gedaan als tijdsbesparing, zodat er geen update en delete request methoden in de synchronisatie nodig zijn. Echter voor de controle is wel vereist dat de cancelbare transactie op de website staan.

De implementatie is als volgt: Bij een kasboek controle wordt eerst normaal gesynchroniseerd. Daarna worden alle transacties van de huidige werkdag van de app naar de online database gestuurd. De transacties worden daar toegevoegd met de flag tijdelijk. Nu kan de controle uitgevoerd worden. Bij het begin van de volgende synchronisatie worden alle tijdelijke transacties weggegooid uit de online database en worden ze overnieuw toegevoegd als de app de werkdag heeft afgesloten, of niet toegevoegd als de werkdag is geannueleerd.

6.1.6 Wachtwoord controle

Deze laatste implementatieparagraaf is speciaal gericht aan de java programmeur, er wordt een voorbeeld gegeven hoe het static keyword gebruikt kan worden. De java code is te vinden in figuur 15. In de volgende alinea wordt de code uitgelegd.

Als op de login-knop wordt gedrukt werkt de passwordverificatie als volgt: Vanuit het inlogscherm wordt de passwordverificatie aangeroepen door PW.check(username,password). De statische check methode pakt het PW object die hoort bij de username uit de hasmap en roept hier vervolgens de niet statische overloaded check methode aan. Deze methode gebruikt de variabelen

```
14:51:17 gps aangeroepen
Locatie geset fix: time 50:3 m = 98   // laatst bekende signaal voldoet aan
    voorwaarden, signaal wordt in variabele geschreven en er wordt poging gedaan
    een nauwkeuriger signaal te vinden.
Gpsfix: t= 51:22 m = 1197.0 p= netw.
Gpsfix: t= 51:22 m = 1197.0 p= netw.  // 5 seconden verstreken: signaal van
    netwerk, straal 1200m nauwkeurig.
Gpsfix: t= 51:34 m = 181 p= gps
Gpsfix: t= 51:34 m = 181 p= gps
Gpsfix: t= 51:35 m = 98 p= gps
Gpsfix: t= 51:35 m = 98 p= gps
Gpsfix: t= 51:37 m = 98 p= gps
Gpsfix: t= 51:37 m = 98 p= gps
Gpsfix: t= 51:39 m = 98 p= gps
Gpsfix: t= 51:39 m = 98 p= gps
Gpsfix: t= 51:40 m = 98 p= gps        // gps warmt op en wordt
Gpsfix: t= 51:40 m = 98 p= gps        // steeds nauwkeuriger.
Gpsfix: t= 51:41 m = 98 p= gps
Gpsfix: t= 51:41 m = 98 p= gps
Gpsfix: t= 51:42 m = 98 p= gps
Gpsfix: t= 51:42 m = 98 p= gps
Gpsfix: t= 51:43 m = 98 p= gps
Gpsfix: t= 51:43 m = 98 p= gps
Gpsfix: t= 51:44 m = 98 p= gps
Gpsfix: t= 51:44 m = 98 p= gps
Gpsfix: t= 51:45 m = 45 p= gps        // 27 seconden verstreken, 45 meter
    nauwkeurig genoeg, gps wordt afgesloten om batterij te besparen.
Locatie geset fix: time 51:45 m = 45. // Gevonden signaal is nauwkeuriger dan het
    laatst bekende signaal dus variabele wordt hiermee overschreven.
```

Figuur 14: De applicatie vind een locatie

van het PW object om een sha1-hash te berekenen en te vergelijken met de opgeslagen sha1-hash overgestuurd van de server.

6.2 Database

In dit hoofdstuk wordt de implementatie van de database beschreven. De gebruikte technieken zullen uiteengezet worden en het database ontwerp zal beschreven worden. Het doel van dit hoofdstuk is het beschrijven van de volledige tabelstructuur van de database.

6.2.1 Entity Framework code first

Zoals beschreven in het oriëntatieverslag is gebruik gemaakt van het Entity Framework en code first. Het maken van de juiste modellen is echter belangrijk en niet triviaal. Daarom is in eerst instantie een ERD (Entity Relationship Diagram) gemaakt dat de basisstructuur van de database beschreef. Dit diagram was veel beperkter dan het uiteindelijke datamodel. Er was bijvoorbeeld slechts een activiteit type in uitgewerkt. Op basis hiervan zijn de C# modellen gemaakt die de volledige database beschrijven.

6.2.2 Database design

Een van de belangrijkste tabellen in deze database is de `Employee` tabel. Hierin staan alle werknemers met hun gebruikersnamen en wachtwoorden. Elke werknemer heeft een of meerdere rollen toegewezen gekregen. Het is namelijk noodzakelijk dat gewone werknemers geen toegang hebben tot de management portal. Dit om te voorkomen dat ze hun eigen activiteiten of gewerkte uren kunnen aanpassen. De tabellen die hiervoor verantwoordelijk zijn zijn de `EmployeeRoles` en `EmployeeRole` tabel. In de `EmployeeRole` tabel staat een lijst met alle mogelijke rollen, zoals 'admin'. De `EmployeeRoles` tabel houdt vervolgens bij welke rollen een werknemer heeft.

```
private static HashMap<String,PW> data;
private String salt, sha1;

public static boolean check(String username, String password)
{
    // pak pw object uit hashmap en return niet-statische check()
    return ((PW)data.get(username)).check(password);
}

private boolean check(String pw)
{
    // sha1 hash van webserver afkomstig
    byte[] sha1json = Base64.decode(sha1, Base64.DEFAULT);
    // sha1 salt van webserver afkomstig
    byte[] ssalt = Base64.decode(salt ,Base64.DEFAULT);
    // password ingevoerd door user
    byte[] password = pw.getBytes("UTF-8");

    // init sha1 bereken object
    MessageDigest digest = MessageDigest.getInstance("SHA-1");
    // toevoegen van password aan sha1 object
    digest.update(password);
    // toevoegen van salt en sha1hash berekenen
    byte[] sha1hash = digest.digest(ssalt);
    // kijk of de berekende sha1hash overeenkomt met sha1hash van de webserver
    return toHex(sha1hash).equals(toHex(sha1json));
}
```

Figuur 15: Stuk code uit PW.java

Om bij te kunnen houden welke werknemers op dit moment in dienst zijn, en hoeveel contracturen ze hebben bestaat de `Contract` tabel. Het bijhouden van de daadwerkelijk gewerkte uren per werknemer gebeurt in de `WorkDay` tabel.

Uitgevoerde activiteiten binnen een werkdag staan in de `Activity` tabel. Om alle informatie van alle soorten activiteiten, zoals ze in het diagram in figuur 3 staan, op te slaan wordt gebruik gemaakt van inheritance. Dat wil zeggen dat de superklasse `Activity` algemene informatie bevat die voor alle activiteiten moet worden opgeslagen, zoals de start en eindtijd, de werkdag waaronder ze vallen en de locatie. Voor de verschillende typen activiteiten, zoals transport, berging, kantoor, huurauto, werkplaats en pechhulp, bestaat een subklasse van het type `Activity`. Omdat opviel dat zowel transport als pechhulp altijd een auto nummer verwachten, en dat deze verplicht is, is er nog een tussenliggende subklasse toegevoegd: `CarActivity`. Transport en pechhulp zijn subklassen van `CarActivity` in plaats van dat ze direct subklassen zijn van `Activity`.

Voor subactiviteiten (de oranje blokken in het schema in figuur 3) is eenzelfde structuur gebruikt als voor de activiteiten. Ook hier bestaat een superklasse `SubActivity`. Voor elk type activiteit bestaat namelijk een `SubActivity` subklasse. De namen van deze subklassen corresponderen met het type activiteit. Zo bestaat er bijvoorbeeld `TransportSubActivity`. Voor alle subactiviteiten die bij transport horen bestaan er een subklasse voor het type `TransportSubActivity`. In de implementatie van het code first model voor `SubActivity` is deze klasse abstract gemaakt omdat het niet de bedoeling is dat er ooit een subactiviteit zonder type, en dus zonder subklasse, wordt aangemaakt. Om te voorkomen dat een subactiviteit die bij transport hoort kan verwijzen naar een activiteit van type kantoor is de verwijzing naar `Activity` vanuit de `SubActivity` klasse ook abstract gemaakt. Hierdoor wordt deze niet meegenomen in de tabel die EF Code First genereert voor dit model. De daadwerkelijke verwijzingen naar activiteiten gebeuren in subklassen van `SubActivity`. Voor transport subactiviteiten, bijvoorbeeld, staat de verwijzing naar `TransportActivity` in de `TransportSubActivity` klasse.

Elk veld dat specifiek bestaat voor een soort subactiviteit is toegevoegd als property in de subklasse die bij dat subactiviteit type hoort. Sommige van deze velden bestaan voor elke subactiviteit die bij een bepaald type activiteit horen. Een voorbeeld hiervan zijn berging subactiviteiten. Deze

hebben allemaal een optioneel veld "kilometerstand". Dit veld is dus verplaatst naar de superklasse van alle subactiviteiten van berging, het type `StorageSubActivity`.

Omdat verlof, ziekte, cursus en bijzonder verlof niet binnen een werkdag vallen is daarvoor een aparte tabel gemaakt die direct naar het contract verwijst: de tabel `LeaveActivity`. Deze tabel is bijna identiek aan de Activity tabel, behalve dat er geen locatie wordt opgeslagen. Ook staat het aantal te registreren uren als kolom in deze tabel, de kolom `CalculatedHours`.

Een ander groot onderdeel van het datamodel zijn transacties. Er zijn, zoals in de ontwerpfase al bleek, 4 soorten transacties: contant geld, via een bankpas, via een DKV kaart of via een Schreurs tankpas. Bovendien horen bankpassen bij personen, maar horen DKV en Schreurs tankpassen bij auto's. Voor elk van deze soorten passen is daarom een tabel in de database die bij houdt welke passen er bestaan. Koppeltabellen houden bij aan welke auto/persoon zo'n pas op een moment toegewezen is. Er bestaat dus voor elk type pas een koppeltabel. De informatie uit deze tabellen wordt gebruikt om te bepalen welke pas is gebruikt bij transacties die via de app ingevoerd zijn. Een gevolg hiervan is dat elke werknemer slechts één pas van elk type toegewezen kan hebben. Anders is immers onduidelijk welke pas bedoeld wordt bij het invoeren van de transactie in de app.

Omdat Schreurs een lokaal bedrijf is, is er nog een tabel met namen van andere (alternatieve) bedrijven toegevoegd die dergelijke tankpassen kunnen leveren. Dit is de `OilCompany` tabel. Deze zou eventueel in de toekomst gebruikt kunnen worden als Schreurs vervangen wordt maar momenteel is er in zowel de Android app als de website onvolledige of geen support voor andere tankpassen dan Schreurs en DKV.

Om bij een transactie op te slaan welke betalingsmethode gebruikt is wordt wederom inheritance gebruikt. Er is een superklasse voor transacties waarin onder andere het bedrag van de betaling wordt opgeslagen, wie er heeft betaald en wanneer er is betaald. Voor elke betalingsmethode is een subklasse waarin de pas staat die gebruikt is voor de betaling.

Verder zijn er nog een aantal simpele tabellen die niet verantwoordelijk zijn voor de hoofdfunctionaliteiten in eInsight. Deze tabellen zijn:

- `Car`: In deze tabel worden voertuigen opgeslagen die in het bezit zijn van Broekmans. Deze informatie wordt gebruikt om te bepalen uit welke auto's gekozen kan worden bij het toevoegen van een activiteit. Ook wordt hiernaar verwezen om bij te houden aan welk voertuig een DKV of Schreurs pas is toegekend.

- `Trailer`: In deze tabel worden aanhangwagens opgeslagen die in het bezit zijn van Broekmans. Deze wordt net als de `Car` tabel gebruikt om te bepalen uit welke aanhangwagens gekozen kan worden bij het toevoegen van een activiteit.

- `ProcessedVehicle`: Bij veel activiteiten die werknemers van Broekmans uitvoeren komen externe voertuigen voor. Om het mogelijk te maken om in de toekomst alle activiteiten te vinden die hebben plaatsgevonden met een bepaalde auto is hiervoor een tabel gemaakt. Bij het invoeren van een kenteken van een extern voertuig in de applicatie of website wordt een rij toegevoegd aan deze tabel.

- `RentalCar`: Broekmans neemt soms huurauto's in, of geeft ze uit, die in het bezit zijn van andere bedrijven. De kentekens van deze auto's worden opgeslagen in de `RentalCar` tabel. Het is mogelijk om extra informatie over dergelijke huurauto's toe te voegen vanuit de portal, maar deze komt nergens terug.

- `DefectType`: Bij het repareren van auto's is het mogelijk om aan te geven wat er kapot is. De selectie van mogelijke defecten komt uit deze tabel.

- `PublicHoliday`: In deze tabel dienen alle feestdagen te staan zodat deze kunnen worden gebruikt voor de urenberekening.

- `Picture`: Als extra feature is ooit bedacht dat het mogelijk moet zijn om foto's te maken van `ProcessedVehicle`s. Deze foto's zouden in de `Picture` tabel terecht komen. Deze feature is echter nooit geïmplementeerd.

- `ErrorLog`: In deze tabel worden alle fouten bijgehouden die op de website of tijdens de synchronisatie optreden. Dit is handig om uit te vinden waardoor een bug of crash is veroorzaakt zodat deze op een snelle en correcte manier kan worden opgelost.
- `SynchronizerLog`: Een andere tabel die alleen bestaat om debug redenen. Alle requests en responses van de synchronisatie SOAP service worden hierin opgeslagen (zie sectie 6.4)
- `LeaveActivityOverviewCache`: Het berekenen van het aantal overuren is een dure berekening: alle werkdagen uit het verleden moeten doorlopen worden. De uitkomst hiervan wordt opgeslagen in deze tabel, zodat de berekening niet elke keer dat de pagina wordt geladen moet worden uitgevoerd. Op de pagina is een vernieuwen knop om de waarden te actualiseren.

Het volledige Entity Relationship Diagram (ERD) van de database is te vinden in Bijlage B (sectie 13).

6.2.3 Gebruiker authenticatie

Om ervoor te zorgen dat alleen werknemers van Broekmans in kunnen loggen op de portal en in de app is authenticatie met een gebruikersnaam en wachtwoord toegevoegd. Het is niet veilig om deze informatie zonder encryptie op te slaan in de database, zeker met in het achterhoofd dat de synchronisatie service het mogelijk maakt om alle werknemers met hun gebruikersnamen en wachtwoorden op te vragen. Daarom is encryptie noodzakelijk. We hebben gekozen voor SHA1 encryptie met een SALT voor elke werknemer. Zowel de SHA1 hash als de SALT worden opgeslagen in de `Employee` tabel.

De encryptie werkt als volgt:

1. De gebruiker kiest een wachtwoord, bijvoorbeeld “geheim”.
2. Het systeem zet dit wachtwoord om in bytes met behulp van UTF8 encoding. In hexadecimaal wordt “geheim” dan:

   0x67656865696D
3. Er wordt een willekeurig byte array gegenereerd. Dit noemen we de SALT. In eInsight worden SALTs met een lengte van 64 bytes gebruikt. Een dergelijke SALT ziet er als volgt uit:

   0x2C5A821771A5212F673E8A596DEAF1FA269ECF8899EC24E8EEE29FFA2D18DE1
   31284E493A7A76684248D73CD9BCDEE84B6164B5C8C4C8D73F5941A84AFA82F18
4. De SALT wordt achter het byte array van het wachtwoord geplaatst. In het voorbeeld zou dit tot de volgende string leiden:

   0x67656865696D2C5A821771A5212F673E8A596DEAF1FA269ECF8899EC24E8EEE29FFA2
   D18DE131284E493A7A76684248D73CD9BCDEE84B6164B5C8C4C8D73F5941A84AFA82F18
5. Het hele byte array wat (dus wachtwoord + SALT) wordt nu als input voor het SHA1 hashing algoritme gebruikt. De uitkomst hiervan wordt opgeslagen in de database als het wachtwoord. In dit voorbeeld zou dit leiden tot de volgende byte array:

   0xC3840AC08C69E304BA56A986BDE31B82DEC1A1AB

Het verifiëren van een wachtwoord gaat op dezelfde wijze als het genereren van het wachtwoord en de SALT, alleen wordt nu de SALT gebruikt die in de database staat in plaats van dat er een willekeurige SALT gebruikt wordt. Als het uiteindelijke byte array gelijk is aan het byte array dat in de database staat dan is het wachtwoord correct.

Het voordeel van deze encryptie is dat het niet mogelijk is voor een hacker om in een redelijke tijd het wachtwoord te achterhalen, zelfs als hij beschikking hebt over het SHA1 gehashte wachtwoord en de gebruikte SALT. Als er geen gebruik wordt gemaakt van SALT dan kan de hacker

een lookup table gebruiken. Dat is een soort Dictionary met voor elke hash het bijbehorende wachtwoord. Maar omdat de SALT samen met het wachtwoord geëncrypt wordt, zou een hacker ook nog voor elke mogelijke SALT een lookup table moeten genereren. Dit is niet in een reële tijd te doen: er zijn immers ongeveer 10^{154} mogelijke SALTs bij een SALT van 64 bytes [3].

6.3 Management Portal

Voor de eInsight management portal is gebruik gemaakt van ASP.NET met MVC3. Dat wil dus zeggen dat er in principe drie soorten klassen zijn: model-, view- en controller klassen. De model klassen voeren in principe geen acties uit. Ze worden alleen gebruikt om informatie op te slaan en dienen als het communicatie medium tussen de controllers en de views. De controllers zijn de klassen die de data ophalen uit de database, verwerken, in models stoppen en doorsturen naar de view. De view klassen zijn verantwoordelijk voor het op een overzichtelijke, gebruikersvriendelijke manier weergeven van de informatie uit de models: zij zijn verantwoordelijk de gebruikersinterface. Een aantal controllers in eInsight hebben overeenstemmende onderdelen functionaliteit. Om deze functies om een plek te houden bestaan er een aantal helpers. Voor al deze soorten klasse staat hieronder een gedetailleerdere uitleg.

6.3.1 Model

Omdat voor eInsight Entity Framework Code First is gebruikt voor het database ontwerp leek het in eerst instantie een goed idee om de models die hiervoor zijn gemaakt ook te gebruiken als models voor het MVC framework. Al snel bleek echter dat er velden waren die niet expliciet in deze models terug kwamen, maar die wel in de view beschikbaar moesten zijn. Bovendien zijn sommige delen van het datamodel, zoals hoe subactiviteiten worden opgeslagen, erg complex. Om deze informatie uit te lezen zou veel logica nodig zijn in de view en dat is niet gewenst. Daarom is er besloten om een nieuwe set models te maken die gebruikt worden voor de communicatie tussen de controller en de view. De models die gemaakt zijn voor Entity Framework Code First, ook bekend als de datamodels, worden nu uitsluitend gebruikt voor communicatie tussen de database en de controller.

In de uiteindelijke implementatie bestaan meer dan 50 models. Hiervan zijn een aantal models speciaal ontwikkeld voor een bepaalde view. Zo bestaat het `ActivityMonthOverviewModel` waarin alle informatie zit die het activiteiten overzicht nodig heeft: een lijst met de activiteiten in de gekozen periode, informatie over de gekozen periode, en informatie over alle werknemers die getoond dienen te worden. Models van dit type worden gevuld in de controller methode die bij die view hoort, of door een helper methode.

Andere models, zoals `CarModel` en `TrailerModel` corresponderen vrijwel één op één met hun bijbehorende datamodels. Deze models worden bijvoorbeeld gebruikt in de `DataController` (zie subparagraaf 6.3.3) om alle informatie over een auto of werknemer op een simpele manier naar de view te sturen. Ook de view specifieke models maken gebruik van deze op de datamodels gebaseerde models. Zo heeft het `ActivityMonthOverviewModel` een lijst van `EmployeeModel`s zodat de view zelf kan bepalen welke informatie over werknemers getoond wordt.

Extension methods in C# maken het mogelijk om methodes toe te voegen aan een bestaande klasse zonder dat de code hiervoor direct in de klasse zelf staat. Volgens de `mvc` structuur horen models geen functionaliteit te bevatten. Extension methods maken het mogelijk om in de code een duidelijke scheiding te houden tussen de models en functionaliteit, terwijl er wel duidelijk is welke functies bij een bepaald model horen. Alle models die direct corresponderen met een datamodel bevatten een extension method `ToModel` die in staat is om een datamodel om te zetten tot een model. De meeste hiervan bevatten ook een `Save` methode die, gegeven een instantie van de `DataContext`, het model omzet in een datamodel en op slaat in de database.

Een aantal modellen met bijbehorende extension methods verdienen nog speciale aandacht:

- `SubActivityModel`: In het datamodel heeft elk type subactiviteit zijn eigen subklasse (zie paragraaf 6.2). Als in de models dezelfde structuur zou zijn gebruikt zou er in de views

veel gecast moeten worden om de juiste informatie te tonen. Dat is niet handig. Daarom is ervoor gekozen om het slechts één model te gebruiken voor alle typen subactiviteiten. In dit model is een property die aangeeft wat voor type subactiviteit het is. Voor de verschillende velden van elk type subactiviteit bestaat een (nullable) veld in dit model. Voor elke property die een `SubActivity` subklasse heeft dient het een interface te implementeren om aan te geven dat het deze property heeft. In de `ToModel` methode om een `SubActivity` om te zetten tot een `SubActivityModel` wordt daarna gecontroleerd welke interfaces de `SubActivity` implementeert, en op basis hiervan wordt het `SubActivityModel` gevuld. Om op deze manier bijvoorbeeld het aantal liters brandstof dat bij een tankbeurt gebruikt is in het `SubActivityModel` op te slaan wordt de volgende code gebruikt:

```
public static SubActivityModel ToModel(this SubActivity sub, int? activityID
    = null)
{
    if (sub == null)
        return null;
    model = new SubActivityModel
    {
        // Velden invullen die elke subactiviteit hebben
    };
    var litersFuelSub = sub as ILitersFuel;
    if (litersFuelSub != null)
        model.LitersFuel = litersFuelSub.LitersFuel;
    // Voor alle mogelijke velden bestaat een dergelijke cast, nullcheck en
        setter
    return model;
}
```

Bij het opslaan wordt een soortgelijke methode gebruikt. Er wordt een instantie van de juiste `SubActivity` subklasse aangemaakt die bij het type subactiviteit hoort dat wordt opgeslagen. Vervolgens wordt dit model gecast na alle mogelijk invoer veld interfaces om de bijbehorende velden in te vullen. Voor sommige properties, zoals geladen voertuigen, die niet via een interface gevuld kunnen worden, omdat het bijvoorbeeld een many-to-many relatie betreft, wordt de methode `FillSubActivity` gebruikt met een overflow voor dat subactiviteit type. In code:

```
public static SubActivity Save(this SubActivityModel model, int activityID,
    eInsightContext db)
{
    Type subActivityType = GetType(model.SubActivityType);
    // Check if the received type is a valid SubActivityType
    if (subActivityType != null &&
        subActivityType.IsSubclassOf(typeof(SubActivity)))
    {
        var dbActivity = db.Activity.FirstOrDefault(a => a.ActivityID ==
            activityID);
        dynamic dbSubActivity =
            dbActivity.GetSubActivities().FirstOrDefault(s => s.SubActivityID
            == model.SubActivityID);
        // Apparently the type changed (or the ID..). So we have to delete
            the old SubActivity and add a new one of the correct type
        if (dbSubActivity != null && dbSubActivity.GetType().BaseType.Name !=
            subActivityType.Name)
        {
            dbSubActivity.Deleted = true;
            dbSubActivity = null;
        }
        // Create a new subactivity of the correct type
        if (dbSubActivity == null)
        {
            // Cast to dynamic instance of this specific subactivity.
            dbSubActivity = Activator.CreateInstance(subActivityType);
            db.SubActivity.Add(dbSubActivity);
        }
```

```
            model.Validate(activity, db);

            // Use the interfaces to fill in properties that are simple and used
                by multiple types of subactivities
            FillDataModelBasedOnInterfaces(model, dbSubActivity, activity, db);

            // Since dbSubActivity is dynamic, the correct method will be called
                (most specific overload). If dbSubActivity was just a var, it
                would be one of type SubActivity, so all overloads for subclasses
                of SubActivity wouldn't be called.
            FillDataModel(model, dbSubActivity, activity, db);
            return dbSubActivity;
        }
        return null;
    }

    public static void FillDataModel(this SubActivityModel model,
        TransportLoadActivity subActivity, Activity activity, eInsightContext db)
    {
        model.FillTransportLoadActivityVehicles(subActivity, db);
        // After filling the model for this specific subtype, call the
            filldatamodel method for the superclass of this item. That way all
            fields that only contain in superclasses will be filled as well
        FillDataModel(model, (TransportSubActivity)subActivity, activity, db);
    }

    public static void FillDataModel(SubActivityModel model, SubActivity
        subActivity, Activity activity, eInsightContext db)
    {
        // This is the place where all info for subactivities will be filled in.
            This will be called when no more specific overflow exists for the
            given SubActivityType.
        subActivity.Altitude = model.Altitude;
        subActivity.Created = model.Created;
        // etc...
    }
```

- `ActivityModel`: Omdat voor activiteiten een stuk minder subklassen zijn dan voor subactiviteiten is er besloten hier wel gewoon dezelfde structuur aan te houden als de database. Dit bracht wel een probleem met zich mee: wanneer een activiteit wordt aangemaakt vanaf de website kan niet worden meegestuurd om welke subklasse van activiteit het gaat. Daarom wordt het type activiteit opgeslagen in het `ActivityModel`. Vervolgens wordt een extension method aangeroepen met de volgende definitie:

  ```
  public static ActivityModel DownCast(this ActivityModel activity,
      CarModel car, int? trailerID, int? rideNumber) { }
  ```

  Deze methode leest het activiteit type uit het gegeven model, en maakt vervolgens een nieuw model aan van het juiste type. Eventuele extra informatie zoals de auto, aanhangwagen of ritnummer worden ingevuld op basis van de de meegegeven argumenten hiervoor.

6.3.2 View

Views zijn ietwat complexe componenten binnen het systeem. Dit omdat er vaak een combinatie van programmeertalen en *markup-languages* als JavaScript, CSS (Cascading Syle Sheet), HTML (Hypertext Markup Language) en ASP.NET Razor (C#). In onderstaande tabel staat beschreven wat de functie van elk deze onderdeel is binnen het systeem:

Bij de implementatie van de schermen is er vaak een zogenaamd "on-the-fly"model aangemaakt dat data beschikbaar stelt binnen de ASP.NET omgeving in de view. Dit is wenselijk, omdat hierdoor minder business logic voorkomt in de view klassen. Zo is er bijvoorbeeld het ActivitiesViewModel dat gebruikt wordt in onder andere het scherm voor de tijdlijn. Dit model wordt tijdelijk voor de view aangemaakt bij het inladen van de pagina en bevat onder andere de

Programmeertaal / Markup Language	Functie
JavaScript	Kan webpagina's interactief maken door de DOM (Document Object Model) te manipuleren binnen de browser. Het wordt ook gebruikt voor validatie van invoervelden.
JQuery	JQuery is een framework dat het programmeren in JavaScript makkelijker maakt. Heeft handige features, zoals CSS selectors om snel en efficiënt DOM-manipulaties uit te voeren. JQuery kent ook vele plugins, waarvan de Management Portal er ook enkele gebruikt.
CSS	Cascading Style Sheets worden gebruikt om elementen binnen de DOM op te maken.
ASP.NET Razor	Razor is een ASP.NET *view engine* om dynamische webpagina's te creëen met behulp van de C# of Visual Basic programmeertalen.

Tabel 3: Deze tabel geeft aan welke talen en standaarden een rol spelen binnen de views van het systeem.

volgende informatie: de werknemer waarvoor de activiteiten geregistreerd zijn, het jaar, de maand en de dag waarvoor de data opgevraagd is, een lijst van activiteiten, sub-activiteiten en werkdagen die die dag hebben plaatsgevonden.

Voor de Management Portal wordt relatief veel gebruik gemaakt van JavaScript. Er worden vaak zogenaamde modal views, oftewel pop-ups, weergegeven die hun inhoud dynamisch verkrijgen doormiddel van een AJAX (Asynchrounous JavaScript and XML) verzoek aan de server. Om dit proces te vereenvoudigen en om de browser-compabiliteit ten goede te komen is er gebruik gemaakt van de JQuery bibliotheek. Ook wordt JQuery veelal gebruikt voor het selecteren van elementen in de DOM op basis van zogenaamde CSS Selectors om vervolgens manipulatie toe te passen binnen het scherm of validatie uit te voeren op elementen.

Al snel werd duidelijk dat er veel JavaScript code hergebruikt kon worden. Voordat er de code gerefactored werd, was er een soort "globale"JavaScript klasse die gebruikt werd in bijna alle schermen. Dit bestand bestond op een gegeven moment uit meer dan 600 regels code. Na het refactoren is alle code die van toepassing was op meerdere schermen uit dit bestand gehaald en in een zogenaamde helper klasse geïmplementeerd. Dit heeft geresulteerd in beter begrijpbare code en is de onderhoudbaarheid van de code ten goede gekomen. Ook is het aantal regels code van het oorspronkelijke bestand met ongeveer 300 regels teruggebracht naar 327 regels en zijn er veel stukken code die hetzelfde waren weggehaald. Onderstaand stuk code is onderdeel van de zojuist besproken helper klasse. Het combineert een aantal handige functionaliteiten die doorheen de website gebruikt worden.

```
$(function () {
    window.helper = (function () {
      ...
      // sends a request to the controller given an url, paramaters object and a
          callback
      // shows an alert whenever an error occurs
      sendControllerRequest: function (url, parameters, callback) {
        // show the loading screen
```

```
        $('body').modalmanager('loading');
        // construct the request
        $.postJSON(url, parameters, function (response) {
           // dismiss loading screen
           $('body').modalmanager('loading');
               // if an error occured went wrong, show an alert
               if (response.error) {
                   alert(response.error);
               } else {
               // all went okay, invoke the callback function
               if (callback != null && callback !== '') {
                   callback.call();
               }
            }
      });
    }
    ...
    // return the library
    return helper;
    } ());
});
```

Bovenstaande code maakt een POST verzoek naar de server met behulp van JQuery via AJAX. Dit verzoek wordt verstuurd naar de controller die in de url wordt gespecificeerd. In het verzoek kunnen ook parameters worden meegegeven, gewoonlijk een JSON object. Het laatste argument is optioneel en betreft een callback variabele. Hierin kan een functie worden meegegeven die aangeroepen wordt wanneer het verzoek succesvol is aangekomen bij de server en we een correct antwoord krijgen. Het gebruik van callbacks sluit goed aan bij de asynchrone structuur waar JavaScript bekend om staat. Ook wordt er in het geval dat er iets mis gegaan is met het verzoek een scherm weergegeven met daarin een beschrijving van de fout zodat de gebruiker feedback krijgt wanneer er deze is opgetreden. Tijdens het laden van het verzoek aan de server wordt er ook een laadbalk weergegeven, zodat de gebruiker weet dat de applicatie "aan het werk is". Bovenstaande code kan vanuit elk scherm worden aangeroepen. Een voorbeeld van hoe deze code gebruikt kan worden is onderstaand stuk code. Het toont aan hoe makkelijk het is om op een abstracte en consistente manier te communiceren met de server via JavaScript.

```
// send a request to the activity controller with the edited activity data
window.helper.sendControllerRequest('/acticity/editActivity',
  {id: 4, type: 'transportTankActivity'},
  function(){
    // when loading is done, reload the UI
    this.reloadTimeline();
    this.reloadActivitiesSubactivitiesTable();
});
```

Bovenstaand voorbeeld stuurt een verzoek naar de ActivityController met daarin de vraag of deze het Activity record met ID 4 kan updaten naar activiteit-type "TransportTankActivity", oftewel een tankactiviteit. Wanneer de server klaar is, worden er twee elementen in de user interface opnieuw ingeladen, zodat de gebruiker de pagina niet hoeft te verversen en direct de geüpdatete data kan zien.

Het dynamische formulier

In figuur 3 is te zien dat er veel verschillende sub-activiteiten geregistreerd kunnen worden door de werknemers. Via de Management Portal moeten deze ook ingevoerd en beheerd (en aangepast) kunnen worden. Er is besloten voor dit probleem een dynamische oplossing te implementeren. Een statische oplossing zou immers leiden tot heel veel dubbele code, slechte onderhoudbaarheid en minder flexibiliteit. Bovendien leent JavaScript zich goed voor dit soort problemen. In figuur 16 is te zien hoe het scherm eruit ziet. De applicatie toont op basis van wat er bij "Soort verrichting"geselecteerd is, invoervelden voor attributen die voor dit type sub-activiteit geregistreerd zijn.

De attributen die behoren bij elke sub-activiteit zijn gespecificeerd in de `SubActivityModelExtension` klasse binnen de ASP.NET omgeving. Ze worden in een zogenaamd *dictionary* opgeslagen en doorgegeven aan de controller. Onderstaand stuk code is een voorbeeld van de specificatie van een tank sub-activiteit tijdens een transportdienst. Voor elk attribuut wordt onder andere aangegeven of het een verplicht attribuut is, welk type het attribuut is (bijvoorbeeld een getal of een lijst van items) en bij welk "groep type"dit attribuut hoort. De groep types zijn gentroduceerd ter ondersteuning van de validatie van het dynamische formulier. De volgende types zijn gedefinieerd:

Naam groep	**Betekenis**
Alle	alle velden binnen deze groep zijn verplicht
Één of meer	minimaal een veld moet ingevuld zijn binnen de group
Alle of geen	ofwel alle velden, of geen velden moet ingevuld zijn
Geen of meer	een willekeurig aantal velden mag worden ingevuld

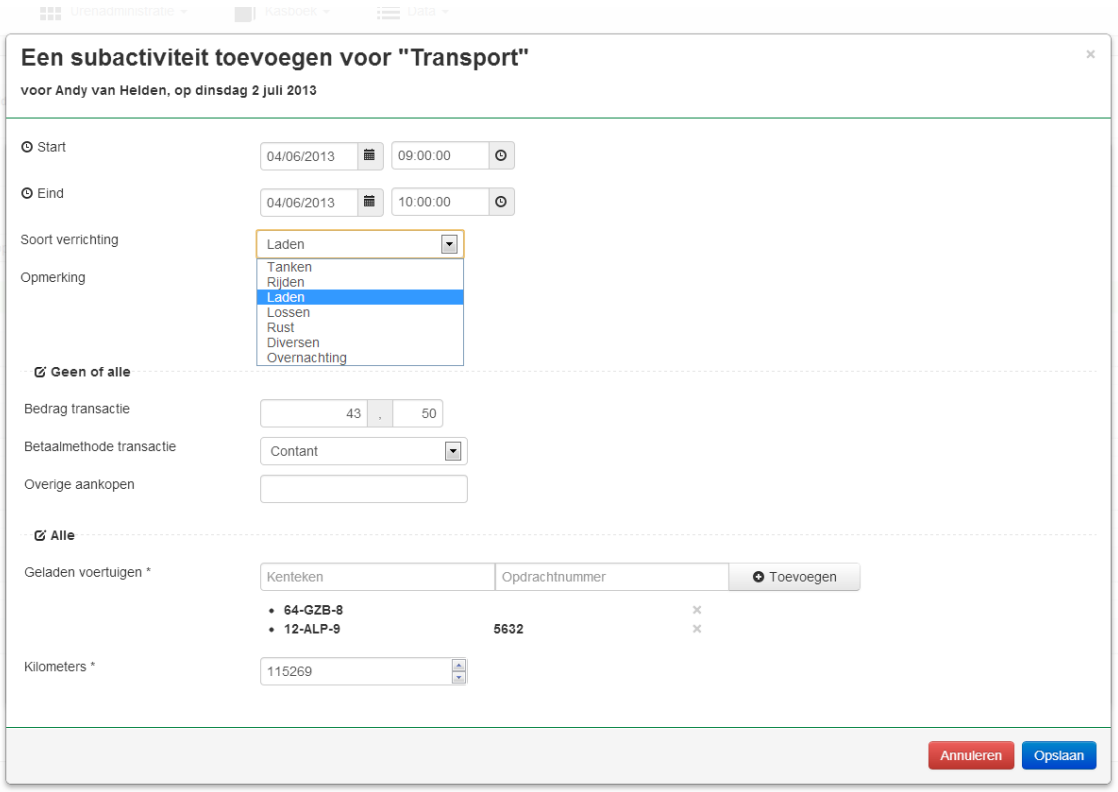
Tabel 4: Deze tabel geeft de groep types weer waar attributen van sub-activiteiten toe behoren.

```
switch (model.SubActivityType.ToLower())
{
    ...
case "transporttankactivity":
    AddToDictionary(values, v => v.KilometerCount, true, model, FieldType.Int );
    AddToDictionary(values, v => v.LitersFuel, false, model, FieldType.Decimal,
        string.Empty, "Tank",  GroupType.OneOrMore);
    AddToDictionary(values, v => v.LitersAdblue, false, model, FieldType.Decimal,
        string.Empty, "Tank", GroupType.OneOrMore);
    AddToDictionary(values, v => v.TransactionAmount, true, model,
        FieldType.Decimal, string.Empty, "Transaction", GroupType.All);
    AddToDictionary(values, v => v.TransactionPaymentMethod, true, model,
        FieldType.SelectList, string.Empty, "Transaction", GroupType.All);
    AddToDictionary(values, v => v.TransactionRemarks, false, model,
        FieldType.String, string.Empty, "Tank", GroupType.OneOrMore);
    break;
    ...
}
```

Wanneer de controller bovenstaand *dictionary* ontvangt, wordt deze doorgestuurd naar de view (ofwel, het scherm). Het scherm maakt dan op basis van wat er in het woordenboek staat, groepen aan, met daarin invoervelden voor de attributen van de sub-activiteit. Het resultaat hiervan is te zien in figuur 16.

Tenslotte wordt er JavaScript gebruikt voor de validatie van de invoervelden binnen het formulier. Hiervoor is een component genaamd *groupValidator* geïmplementeerd. Dit component is gedeeltelijk weergegeven in onderstaande code.

```
$(function () {
    window.groupValidator = (function () {
        var groupTypes = {
            all: function ($group) {
              ...
            },
            oneormore: function ($group) {
                ...
            },
            noneorall: function ($group) {
                ...
            },
```



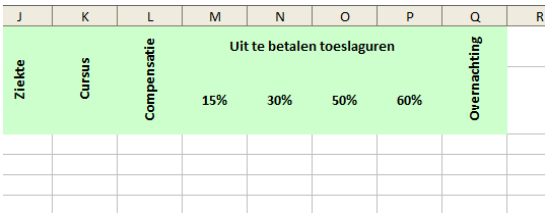
Figuur 16: Een screenshot van het scherm waarin de gebruiker een sub-activiteit kan aanmaken voor een reeds bestaande activiteit. De invoervelden in het formulier worden dynamisch gegenereerd en worden wel of niet weergegeven afhankelijk van wat er in het veld "'Soort verrichting"geselecteerd is.

```
            nonerequired: function ($group) {
                ...
            }
        };

        var validateListInput = function ($list) {
            ...
        };

        var groupValidator = {
            validate: function ($group) {
                if ($group && $group instanceof $) {
                    var type = $group.attr('data-group-type').toLowerCase();
                    return groupTypes[type]($group);
                } else {
                    throw "Input not compatible. Needs a JQuery object";
                }
            }
        };
        return groupValidator;
    } ());
});
```

Voor elke group binnen de DOM wordt er een attribuut "data-group-type" opgeslagen met daarin het type van de group. Op deze elementen kan er via JavaScript dan op de volgende manier worden nagegaan of aan de validatie is voldaan:

```
$group = $('#group') // select the group element from the DOM
```

```
if(window.groupValidator($group)) { // validate group input
  ...
} else {
  ...
}
```



Figuur 17: Screenshot van een Excel bestand dat door de Management Portal is gegenereerd en geopend in Microsoft Excel.

6.3.3 Controller

In eInsight bestaat elk item in het hoofdmenu een controller. Deze controllers voeren de database queries uit die nodig zijn om de informatie op te halen die de view toont. De gegevens die uit deze queries komen worden door de controllers omgezet tot models zoals die zijn beschreven in subparagraaf 6.3.1. Deze models worden vervolgens naar de view gestuurd.

Buiten de controllers per menu item zijn er nog twee controller klasse:

- `BaseController`: Deze controller voert functies uit die overal in de website gebruikt moeten worden. De andere controllers zijn subklasse van deze controller. Hij houdt onder andere bij welke gebruiker is ingelogd, en zorgt ervoor dat als een sessie verlopen is een gebruiker wordt teruggestuurd naar het login scherm.
- `LoginController`: Deze controller valideert de invoer op het login scherm en, wanneer de gebruikersnaam en het wachtwoord correct zijn, wordt de gebruikers doorgestuurd naar het laatste scherm dat hij open had of naar het home scherm als er nog geen schermen open zijn geweest.
- `ExcelExportController`: Deze controller is verantwoordelijk voor het verzamelen van de data die naar een Excel document moet worden geëxporeert. Voor het daadwerkelijk genereren van het Excel document wordt de `ExcelExportHelper` gebruikt (zie subparagraaf 6.3.4).

6.3.4 Helpers

Voor niet triviale functionaliteit gebruiken de controllers in eInsight helpers, bijvoorbeeld als de data uit de database nog verwerkt moet worden door een algoritme voordat deze kan worden weergegeven. Hieronder staat een beschrijving van de belangrijkste helpers in het eInsight project.

- `ExcelExportHelper`: Een belangrijke eis is de mogelijkheid om gegevens te exporteren naar een Excel document. De `ExcelExportController` is de controller die verantwoordelijk is voor het genereren van alle Excel documenten. Hiervoor maakt eInsight gebruikt van de EPPlus library [1]. Er zijn drie soorten spreadsheets die op deze manier gegenereerd kunnen worden: een overzicht met alle uitgevoerde activiteiten, het maandoverzicht van urenadministratie en het jaaroverzicht van de urenadministratie.

  EPPlus ziet een spreadsheet als een 2D array. Hiervan kunnen losse cellen of groepen van cellen (zoals een hele kolom) worden geselecteerd. De selectie wordt omgezet tot een object waarvan de eigenschappen, zoals de waarde, het formaat van het getal en de tekstkleur, kunnen worden bepaald. Het wijzigen van een van deze eigenschappen wordt direct gereflecteerd

de Excel spreadsheet. De library werkt hierdoor erg intuïtief waardoor het opbouwen van de gevraagde spreadsheets zonder veel problemen kon worden gerealiseerd.

In het activiteiten overzicht wordt voor elke activiteit en subactiviteit een rij toegevoegd aan de spreadsheet. Voor elk invoer veld bestaat een kolom. Om deze tabel te vullen wordt met een `for` loop door alle activiteiten en subactiviteiten gelopen. Alle velden die aanwezig zijn in de huidige activiteit of subactiviteit worden, met behulp van een dictionary, op de juiste plek in de spreadsheet ingevuld.

De jaar- en maandoverzichten uit de urenadministratie vertonen veel gelijkenissen. Ze hebben dezelfde stijl en zelfs ongeveer dezelfde kolommen. Daarom is er een methode geschreven die de structuur van deze spreadsheets dynamisch opbouwt. Er is een model gemaakt dat de headers van de kolommen beschrijft. Hierin is ook aangegeven hoeveel kolommen en rijen de header moet overlappen. Dit was nodig omdat de header voor toeslag-uren een wat ingewikkelde lay-out heeft (zie 6.3.2). Vervolgens wordt, net als bij de activiteiten overzicht met behulp van een dictionary, de informatie uit het model ingevuld in de juiste kolommen. De waarden in de laatste rij, waarin voor elke kolom het totaal wordt weergeven, worden bepaald met behulp van een Excel `SOM` functie en zijn dus dynamisch.

- `EncyptionHelper`: Deze helper klasse is in staat om de encryptie zoals die is beschreven in 6.2.3 toe te passen en zo wachtwoorden te controleren of te versleutelen.

- `ExceptionHelper`: Tijdens de ontwikkeling bleek het lastig om enkele bugs te reproduceren die tijdens tests van bijvoorbeeld de stagebegeleider naar voren kwamen. Om het debuggen eenvoudiger te maken is daarom de mogelijkheid toegevoegd om fouten te loggen naar de database. De `ExceptionHelper` is verantwoordelijk hiervoor. Er bestaat slechts één public methode in de `ExcelExportHelper`: `LogException(Exception exc, string username)`. Deze methode voegt de informatie over de gegeven exceptie toe aan de database. De excepties die op deze manier zijn opgeslagen zijn terug te vinden in het data menu bij het item 'Fouten log'.

- `HourCalculationHelper`: Om het aantal gewerkte- en verlof uren te berekenen voor het maand- en jaaroverzicht in de urenadministratie (zie subparagraaf 5.3.5) moet deze informatie uit de werkdagen en verlof activiteiten, zoals die in de database staan, worden gehaald. In de database staat echter alleen een begin en eindtijd. Bovendien kan het in theorie door fouten voorkomen dat werkdagen overlappen. De `HourCalculationHelper` is ervoor verantwoordelijk dat het aantal gewerkte uren correct wordt weergegeven.

  Het berekenen van het aantal gewerkte uren wordt gebaseerd op de werkdagen die binnen de gekozen periode vallen. De onderstaande code is verantwoordelijk voor het berekenen van de gewerkte uren. Soortgelijke methodes bestaan ook voor het berekenen van het aantal pauze uren en standby uren op een dag.

```
public static decimal GetWorkedHoursBetween(IEnumerable<WorkDayModel>
    workdays, DateTime start, DateTime end, bool countBreaks)
{
    if (!workdays.Any())
        return 0m;
    var workdaysToCount = workdays.Where(w => !w.Deleted && w.Start < end &&
        w.End > start);
    // Eerst wordt het totaal aantal uren berekent dat in de gekozen periode
        binnen werkdagen valt. Hierbij wordt geen rekening gehouden met
        overlappende werkdagen of pauzes
    decimal totalHoursToday = (decimal)workdaysToCount.Sum(w =>
        (DateTimeHelper.Earliest(end, w.End.Value) -
            DateTimeHelper.Latest(start, w.Start)).Value.TotalHours);
    // Vervolgens wordt berekend hoeveel uur van de totale tijd op
        overlappende werkdagen valt.
    var overlappingTime = GetOverlappingTime(workdays, countBreaks);
    // Als laatste wordt het aantal uren dat de werknemer op de huidige dag
        pauze had berekent.
```

```
    decimal breakTime = 0;
    if (!countBreaks)
    {
        breakTime = GetBreakTime(workdaysToCount, start, end) +
            GetStandbyTime(workdaysToCount, start, end) +
            GetSleepOverTime(workdaysToCount, start, end);
    }
    return totalHoursToday - overlappingTime - breakTime;
}

private static decimal GetOverlappingTime(IEnumerable<WorkDayModel> workdays,
    bool countBreaks)
{
    // Werkdagen omzetten tot "TimePeriod" objecten.
    var workdayArray = workdays.Select(w => new TimePeriod() {Start =
        w.Start, End = w.End ?? DateTime.Now}).OrderBy(w =>
        w.Start).ToArray();
    // Als de pauzes genegeerd moeten worden moeten ze ook genegeerd worden
        bij het berekenen van overlappende tijd. De werkdagen moeten dan dus
        opgeknipt worden als periodes tussen de pauzes.
    if (!countBreaks)
    {
        var splittedWorkedayArray = SplitWorkdaysOnBreaks(workdays);
        workdayArray = splittedWorkedayArray.OrderBy(w => w.Start).ToArray();
    }
    var overlappingTime = GetOverlappingTime(workdayArray);
    return overlappingTime;
}

private static decimal GetOverlappingTime(TimePeriod[] timePeriods)
{
    // Om de overlappende tijd te berekenen wordt per TimePeriod bijgehouden
        of die overlaps met een voorgaande TimePeriod, en zo ja welk deel van
        de periode dan overlappend is.
    for (var i = 0; i < timePeriods.Length; i++)
    {
        // Omdat overlaps niet dubbel geteld moeten worden hoeven in de
            tweede for loop alleen gekeken te worden naar de rest van het
            TimePeriod[]
        for (int j = i + 1; j < timePeriods.Length; j++)
        {
            if (timePeriods[i].End < timePeriods[j].Start)
                break;
            if (timePeriods[i].Start < timePeriods[j].End &&
                timePeriods[i].End > timePeriods[j].Start)
            {
                // Als deze werkdag een andere werkdag overlapt, dan moet de
                    overlap tijden worden aangepast
                var overlapStart =
                    DateTimeHelper.Latest(timePeriods[i].Start,
                    timePeriods[j].Start);
                var overlapEnd = DateTimeHelper.Earliest(timePeriods[i].End,
                    timePeriods[j].End);
                if (!timePeriods[j].OverlapStart.HasValue ||
                    timePeriods[j].OverlapStart > overlapStart)
                    timePeriods[j].OverlapStart = overlapStart;
                if (!timePeriods[j].OverlapEnd.HasValue ||
                    timePeriods[j].OverlapEnd.Value < overlapEnd)
                    timePeriods[j].OverlapEnd = overlapEnd;
            }
        }
    }
    // De totale tijd van alle overlaps wordt nu opgeteld
    var overlappingTime = 0m;
    foreach (var workdayPeriod in timePeriods)
    {
        if (workdayPeriod.OverlapStart.HasValue &&
```

```
                workdayPeriod.OverlapEnd.HasValue)
                overlappingTime += (decimal)(workdayPeriod.OverlapEnd -
                    workdayPeriod.Start).Value.TotalHours;
        }
        return overlappingTime;
    }
```

Voor de berekening van verlof- en ziekte uren wordt rekening gehouden met het aantal te registreren uren. Als de verlof- of ziekte activiteit niet volledig op de gekozen periode past, wordt het aantal te registreren uren hierop aangepast. Bijvoorbeeld: Stel er bestaat een verlof activiteit van 3 dagen (dus 72 uur) met 24 te registreren uren. Als nu hiervan het aantal verlofuren op een van de 3 dagen wordt opgevraagd, dan valt dus slechts $\frac{1}{3}$ van de verlofactiviteit binnen de gekozen periode. Hierdoor wordt ook slechts $\frac{1}{3}$ van de te registreren uren meegeteld, dus $\frac{1}{3} \times 24 = 8$ uur.

- `LocalizationHelper`: In eInsight worden standaard .NET resources gebruikt voor lokalisatie. .NET is in staat om, gebaseerd op de in het configuratie bestand gekozen cultuur, resources te selecteren die specifiek voor een taal zijn. Momenteel wordt alleen de Nederlandse taal ondersteund in eInsight en zijn niet alle zinnen vertaald. Door het gebruik van resources zou het in de toekomst echter wel goed mogelijk zijn om eInsight te vertalen. Voor gemakkelijke toegang tot de vertaalde resources is de `LocalizationHelper` geschreven. Deze helper vertaalt een string op basis van de resources en de huidige taal door in de resources de gegeven string op te zoeken.

- `OverviewHelper`: Deze helper is in staat om het model te vullen dat gebruikt wordt voor de jaar- en maandoverzichten in de urenadministratie. Hij maakt hierbij gebruik van de `HourCalculationHelper`.

6.4 Synchronisatie

De communicatie tussen de Android applicatie en de webserver met de database vindt plaats via een SOAP service. SOAP staat voor Simple Object Access Protocol en is ontworpen om op een gestructureerde manier informatie uit te wisselen tussen verschillende computers in een netwerk. SOAP gebruikt het HTTP protocol voor communicatie en XML om structuur aan te brengen in de te verzenden informatie. In eInsight wordt bij een aantal requests ook nog JSON gebruikt om de informatie op een meer compacte manier door te sturen en zo het aantal overgestuurde bytes te beperken. Bovendien waren er betere libraries beschikbaar om met JSON te werken voor zowel Android als .NET.

Er zijn verschillende informatie-eenheden waarvoor de synchronisatie moet plaatsvinden, zowel van server naar client als andersom. Zo moeten ingevoerde werkdagen, activiteiten en subactiviteiten met alle bijbehorende informatie worden overgezet naar de server. Hierbij is het van belang dat er geen informatie ontbreekt of dubbel in de database komt te staan. Ook moet de server alle bekende voertuigen, aanhangwagens, defect types en werknemers naar de app sturen, zodat de juiste lijstjes kunnen worden weergegeven.

Hieronder zal zowel de implementatie van de client (de Android app) als de server worden beschreven.

6.4.1 Android

De Android app maakt gebruikt van models voor communicatie met de locale database. Deze models hebben een superklasse: MODEL. Voor elke tabel in de database bestaan subklasse van MODEL Voor de synchronisatie moeten alle activiteiten, hoofdactiviteiten en werkdagen uit de database uitgelezen worden, en via SOAP verzonden worden naar de server. Ook moet de app in staat informatie te ontvangen en op te slaan, zoals bijvoorbeeld de lijst met auto's, aanhangwagens en defect typen. Deze informatie wordt niet in de database opgeslagen: de app gebruikt hiervoor java objecten. Deze objecten worden geserialiseerd en naar het bestandssysteem van de telefoon

geschreven om ze op te slaan. Bij een dergelijk object wordt ook een versienummer opgeslagen. Als de app de autolijst synchroniseert, controleert de server of het meegestuurde versienummer lager is dan het versienummer op de server. Als dat het geval is wordt de volledige autolijst naar de app verzonden. De oude autolijst wordt op de telefoon wordt dan vervangen. Als de versienummers gelijk zijn wordt er niets verzonden.

De synchronisatie zoals die is geïmplementeert in de Android app zal hieronder worden toegelicht met voorbeelden.

- **Activiteiten versturen**:
  Als tijdens het gebruik van de app een activiteit wordt toegevoegd wordt deze in de database gezet en gemarkeerd als zijnde "dirty". Wanneer er vervolgens de synchronisatie service wordt aangeroepen wordt aan de app database controller gevraagd een lijst te geven van alle dirty activiteiten models. Elk model dat via de synchronisatie verzonden kan worden implementeert de `toJson` methode. Hierin worden ook de primary keys van de objecten in de lokale database in de json opgeslagen, de zogenaamde `localID`. Deze methode wordt vervolgens aangeroepen voor alle models die de database controller terug geeft.

  De `json` die terug komt uit de `toJson` methode, wordt meegegeven aan de `ZeepController` klasse. De `ZeepController` maakt een SOAP pakket en vult deze vervolgens met de `json`, de gebruikersnaam van de huidige gebruiker en zijn wachtwoord. Dit pakket wordt verzonden naar de server. Daarna wacht de `ZeepController` op een antwoord van de server. Het verwachte antwoord is in dit geval een SOAP pakket met daarin een JSON array. Deze JSON array bevat per verzonden `localID` een indicatie die aangeeft of het toevoegen op de server succesvol is verlopen. Dit JSON array wordt uit het SOAP object gehaald en geïnterpreteerd door de `ZeepController` en de `ParseJson` klasse. Als laatste worden de activiteiten die volgens deze lijst succesvol zijn verzonden als "clean" gemarkeerd in de locale database.

- **Ontvangen van de lijst met auto's**:

6.4.2 Server

Op de web server draait een .NET SOAP service. Deze SOAP service is onder andere in staat om werkdagen, activiteiten en subactiviteiten, die vanuit de app komen, op te slaan. De methodes die hiervoor zorgen hebben ongeveer dezelfde structuur. Ten eerste verwachten ze allemaal 3 argumenten:

- `data`: JSON string met daarin de informatie die moet worden opgeslagen.
- `username`: De gebruikersnaam van de gebruiker voor wie de werkdagen/activiteiten/subactiviteiten moeten worden toegevoegd.
- `password`: Het wachtwoord van die gebruiker

Het enige verschil tussen de invoer van deze methodes is de structuur van het JSON object `data`. In elk van deze methoden wordt eerst gecontroleerd of de gebruikersnaam en het wachtwoord die zijn opgestuurd zijn correct zijn en er wordt opgehaald bij welke werknemer ze horen. Vervolgens worden helper methodes aangeroepen die de data daadwerkelijk toevoegen in de database.

Voor werkdagen is het opslaan het minst ingewikkeld. Er wordt naar een contract gezocht voor de gegeven werknemer dat geldig is tijdens de volledige werkdag. Vervolgens wordt gecontroleerd of er al werkdagen bestaan die op hetzelfde moment plaatsvinden. Als dat het geval is wordt de ontvangen werkdag samengevoegd met de reeds bestaande werkdagen. Alle activiteiten die bij de reeds bestaande werkdagen horen worden toegewezen aan de nieuwe werkdag.

Het opslaan van activiteiten is lastiger. Ten eerste zijn er natuurlijk meerdere soorten activiteiten welke ieder hun eigen (andere) velden hebben. Er moet dus worden bepaald om welk type activiteit het gaat. Ten tweede kan het voorkomen dat de werkdag die bij deze activiteit hoort nog niet is opgestuurd of dat tijdens het sturen de eerste keer iets mis is gegaan. In beide gevallen is

het wel wenselijk dat de activiteit wordt opgeslagen. Als er dus geen werkdag is gevonden die om deze activiteit past wordt er een werkdag toegevoegd, op de manier zoals hierboven beschreven, die precies om de activiteit past. Als laatste wordt de activiteit toegevoegd met alle bijbehorende velden.

Subactiviteiten zijn het meest ingewikkeld. De helper klasse die verantwoordelijk is voor het toevoegen van subactiviteiten is dan ook bijna 1200 regels lang. Hij werkt als volgt: Eerst wordt het type hoofdactiviteit bepaald dat bij de ontvangen subactiviteit hoort. Daarna wordt gekeken of er een hoofdactiviteit bestaat op het juiste moment voor de juiste persoon. Als dat niet zo is wordt er een aangemaakt. Als de bijbehorende werkdag ook nog niet bestaat wordt die ook aangemaakt zoals dat ook bij het toevoegen van activiteiten gebeurt. Daarna wordt gekeken om welk type subactiviteit het gaat en het verstuurde JSON object wordt geïnterpreteerd als zijnde het juiste type subactiviteit. Nu wordt er een methode aangeroepen die specifiek voor dat type subactiviteit alle waarden die daarbij horen invult. Voor sommige velden, zoals transacties en externe voertuigen, moet nog een object worden toegevoegd aan de database, namelijk de betreffende transactie of het betreffende voertuig. Bij laden en lossen kunnen zelfs meerdere voertuigen worden toegevoegd en moet ook de bijbehorende many-to-many relatie worden aangemaakt. Als laatste wordt de subactiviteit opgeslagen.

Bovenstaande methodes zijn ook op het gebied van uitvoer ongeveer gelijk. Voor elk toe te voegen object dient de client, in dit geval de Android applicatie, in het 'data' object de primary key, die het object heeft op de telefoon, door te sturen. Voor alle dergelijke primary keys geeft de SOAP service terug onder welk id het object is opgeslagen in de database op de server. Als het opslaan niet is gelukt, omdat er bijvoorbeeld informatie ontbreekt, wordt de foutmelding in de response mee terug gestuurd. De client weet dus of het opslaan is gelukt door te controleren of er een exceptie is meegestuurd in de response.

Buiten het toevoegen van data in de database via de SOAP service kan de applicatie ook data opvragen. Zo zijn er bijvoorbeeld een aantal selectie lijstjes die in de applicatie terug moeten komen, zoals de lijst met auto's, aanhangwagens en gebruikers. Voor het opvragen van elk van deze lijstjes bestaat een methode in de SOAP service. Deze methoden vereisen geen gebruikersnaam en wachtwoord zoals de toevoegen methodes. Deze informatie is immers niet persoonsafhankelijk. Wat wel een vereiste is voor elk van deze methode is een versie nummer. Op die manier kan de server controleren of de applicatie reeds de nieuwste versie heeft. Als dat het geval is wordt een lege string terug gestuurd. De client heeft dan namelijk al de laatste versie. Als er een nieuwere versie beschikbaar is wordt de hele lijst opnieuw verzonden. Dit is het geval omdat er momenteel niet wordt bijgehouden wat er gewijzigd is, alleen dat er wijzigingen zijn. De overhead hiervan is bovendien gering: alle lijstjes bestaan slechts uit een stuk of 20-30 items. Het versturen kost dus niet al te veel tijd. Al deze methodes geven een JSON array met hierin objecten die de opgevraagde data beschrijven.

6.5 SIG feedback

Zoals te lezen in de respons van SIG (Bijlage D, hoofdstuk 15) verdiende de code van eInsight 3 van de 5 sterren. De grootste kritiek die gegeven werd was dat methoden te lang en complex waren. Ook viel het op dat sommige code vaker terug kwam dan nodig. Om dit probleem tegen te gaan is er gerefactored.

Zo zijn in de C# code de `ToModel` en `Save` methode van SubActivity opgesplitst in verschillende methoden per property. Ook zijn een aantal controllers opgesplitst in verschillende methoden, zoals bijvoorbeeld de controller die verantwoordelijk is voor het activiteiten overzicht. Ook de `HourCalculationHelper` is herzien door de verschillende methoden hier in op te splitsen.

Het JavaScript dat gebruikt wordt in de views is volledig herschreven. Veel stukken code waren naar elke view waar ze nodig waren gekopiëerd, zoals bijvoorbeeld het javascript dat verantwoordelijk is voor het opbouwen van de popups. Deze code is nu in libraries gestopt die, wanneer nodig, aangeroepen worden. Dit komt de code duplicatie en method complexity ten goede. Ook is er zo geprobeerd zo min mogelijk code te "exposen" zodat er minder kans is op conflicten in de namespace. Bijvoorbeeld de methode `saveModel()` kan nu niet meer globaal aangeroepen wor-

den, maar alleen via de helper klasse: `window.helper.saveModel()`. Hierdoor kunnen eventuele andere ontwikkelaars gelijk zien dat de methode in deze helper klasse is gedefinieerd. Ten slotte zijn er voor dynamische componenten in de website, zoals het ListInput component, dat gebruikt wordt bij de invoer van bijvoorbeeld het laden en lossen, JQuery plugins gemaakt. Hierdoor zijn de componenten op dezelfde manier aan te spreken als plug-ins van derden dat de algemene cohesie ten goede komt.

De gps klassen zijn geheel herschreven van 2 klassen van in totaal 300 regels met 3 threads, naar 1 klasse van 100 regels en 1 thread. De Zeepcontroller klasse is gerefactored van 200 naar 100 regels door de configuratie van de tcp (soap)pakketen van, die de verschillende server requests bevatten, in een apparte methoden te stoppen. Er is voor gekozen de code die vijftig keer dubbel staat in alle grafische venstertjes te behouden. Deze code is zoals in subparagraaf 6.1.3 toegelicht wordt script gegenereerd. Door de code dubbel te laten staan zorgen we ervoor dat iemand die later iets aan een bepaald schermpje van de applicatie wil wijzigen, alleen de code van dat schermpje hoeft aan te passen. Er blijft de keuze om het script te gebruiken, dit is echter veel complexer in het gebruik.

7 Testen

Al tijdens de oriëntatiefase van het project is er nagedacht hoe de applicaties van eInsight getest zouden worden en wanneer we dit zouden moeten doen. Omdat geen van ons ervaring heeft met het schrijven van tests in Android, ASP.NET en JavaScript is er besloten unit tests te schrijven tijdens de laatste fase van het project (de ondersteuning-fase, zie bijlage 2 van het plan van aanpak). Dit omdat er voor de oplevering van eInsight een strikte deadline ingesteld was, namelijk zes weken na aanvang van het project. Met de gedachte dat we na de zesde week tijd zouden hebben voor onder andere het refactoren, testen en bug-fixen werden destijds het schrijven van unit tests uitgesteld. De werkelijkheid bleek anders: de deadline van oplevering werd niet gehaald en het refactoren en bug-fixen kreeg hogere prioriteit dan testen. Ook stonden er nog een aantal features open die noodzakelijk waren voor het correct opleveren van de applicatie. Na in week 9 een gesprek te hebben gehad met Prof. Hidders, kwamen we tot de conclusie dat er drie zaken waren waaraan er nog tijd besteed kon worden: het refactoren (of herschrijven) van de code, het schrijven van unit tests en het implementeren van de "would have"features van het MoSCoW model. Elk van deze keuzes had zijn voor- en nadelen:

Optie	**Pro**	**Con**
Refactoren	++ Onderhoudbaarheid	- Wanneer is de developer "klaar"met refactoren?
	+ Applicaties zijn makkelijker uit te breiden (bij gebruik van bekende design patterns is de code makkelijker te herkennen door andere developers)	
Unit Testen	++ Problemen ontdekken in de code	- - Een grote klus voor het project in dit stadium (JavaScript, C#, Java)
	+ Vroeg bugs ontdekken na refactoren	
Features Toevoegen	+ Klanttevredenheid	- - De code wordt er kwalitatief niet beter op, of gaat achteruit

Tabel 5: Deze tabel laat zien welke keuzes we hadden in week negen van het project en welke voor- en nadelen deze hadden.

Uitgaande van tabel 5 is er besloten de code te refactoren. Omdat de weken ervoor het product opgeleverd moest worden, was er veel code overhaast geïmplementeerd en naar ons gevoel had het prioriteit om hierin structuur aan te brengen om de onderhoudbaarheid op te krikken.

Toch is onze applicatie getest, in de vorm van gebruikerstests. Medewerkers van Adecs Airinfra hebben de applicaties tussen week 7 en week 9 getest. Verschillende onderdelen van de software zijn op verschillende manieren getest. De Management Portal en de Android App zijn (individueel) op de volgende manier getest:

1. Log in
2. Maak een werkdag aan voor een gebruiker
3. Maak voor die werkdag een hoofdactiviteit aan
4. Open het scherm waarin een nieuwe sub-activiteit kan worden aangemaakt
5. Vul géén waarden in en klik op **opslaan**

6. Controleer dat de juiste foutenberichten worden weergegeven wanneer dit nodig is, of dat het item correct wordt opgeslagen indien validatie goed ging
7. Vul "'realistische waarden" in voor de invoervelden en klik op **opslaan**
8. Verifieer het resultaat, kijk of de sub-activiteit correct is opgeslagen
9. Herhaal stap 4 voor alle mogelijke sub-activiteiten die onder de gekozen hoofdactiviteit vallen
10. Herhaal stap 3 voor alle activiteiten

Als integratietest is de volgende strategie toegepast:

1. Pas bovenstaande testmethode toe voor de Android App
2. Log in op de eInsight Management Portal
3. Verifieer de gesynchroniseerde data in het activiteitenoverzicht
4. Verifieer de attributen van de sub-activiteiten in het dagoverzicht
5. Verifieer eventuele transacties die hebben plaatsgevonden tijdens het registreren van activiteiten in het kasboek

Tijdens bovenstaande tests zijn formulieren gebruikt als hulpmiddel voor verificatie tijdens het testen. De formulieren zijn bijgevoegd in bijlage I.

8 Eisen evaluatie

Dit hoofdstuk biedt een reflectie op de gestelde eisen zoals die zijn te zien in hoofdstuk 4. Per gestelde eis zal worden bepaald tot in hoe verre deze verwezenlijkt is in het eindproduct. Eventuele eisen die er tijdens het ontwikkelingstraject worden schuingedrukt weergegeven. Om ruimte te besparen in deze herhaalde weergave van het in hoofdstuk 4 opgestelde MoSCoW requirements model, worden de afkortingen 'M', 'S', 'C' en 'W' gebruikt voor de termen "Must have", "Should have", "Could have" en "Would like".

8.1 App

M	✓	De mogelijkheid om alle uitgevoerde activiteiten te registreren.	De menu structuur maakt het mogelijk om op een intuïtieve manier alle mogelijke activiteiten en subactiviteiten in te voeren.
	✓	Cash transacties en het huidige verwachte saldo moeten inzichtelijk zijn	Via het kasboek overzicht in de app kan inzicht worden verkregen in alle cash transacties. Ook het huidige saldo wordt hier getoond.
	✓	Cash, creditcard, DKV en Schreurs pas transacties moeten kunnen worden toegevoegd	Bij het invoeren van activiteiten waarbij betalingen kunnen voorkomen kan een bedrag en een betalingswijze worden gekozen. Op deze manier geregistreerde betalingen komen terug in het kasboek op in de management portal.
	✓	Als er geen internetverbinding is moet het toch mogelijk zijn de app te gebruiken.	Ingevoerde informatie wordt opgeslagen in een lokale database op de telefoon. Hiervoor is geen internetverbinding vereist. Deze is pas nodig tijdens het synchroniseren met de server.
	✓	De app moet werken op de Motorola Defy+	De app is volledig ontwikkeld en getest op deze telefoon.
S	✓	Bijhouden van de locatie van de gebruiker wanneer deze activiteiten of werkdagen start (en/of eindigt)	Bij het toevoegen van een activiteit of werkdag wordt een nieuwe GPS locatie opgevraagd. Deze locatie wordt opgeslagen bij de activiteit.
C	✓	Automatische synchronisatie wanneer er een internetverbinding beschikbaar is	Wanneer de app in het hoofdmenu terecht komt en er is een Wi-Fi verbinding beschikbaar wordt de synchronisatie gestart
	✓/ ×	Mogelijkheid om de app te gebruiken op andere telefoons dan de Motorola Defy+	De app is op een aantal telefoons geïnstalleerd en oppervlakkig getest. Hierbij zijn geen bugs ontdekt die de app onbruikbaar maakt. Echter is er ook niet voldoende getest om dit met zekerheid te zeggen
W	×	Registratie van defecten aan auto's	Wegens tijdgebrek is deze functie niet geïmplementeerd.

8.2 Portal

M	✓	De database en de Management-Portal moeten kunnen draaien op een door de opdrachtgever aangeschafte Windows-server	Omdat gebruik is gemaakt van een Microsoft SQL Server database en een in .NET ontwikkelde back-end werkt eInsight optimaal op een Windows-server
	✓	De weergegeven activiteiten en uren- en ziektedata moeten kunnen worden geexporteerd naar een Microsoft Excel bestand	
	✓	Er moet een digitaal kasboek aanwezig zijn, waarin overschrijving met contant geld, DKV passen, creditcards en Schreurs tankpassen inzichtelijk zijn	Het kasboek menu in de portal en alles wat daarbij hoort vormt het digitale kasboek
	✓	De door het personeel geregistreerde uren, toe te kennen toeslagen en overuren moeten berekend kunnen worden	Het maand- en jaaroverzicht in het urenadministratie menu bevatten de gevraagde informatie
	✓	Registratie van ziekteverzuim en verlofuren van werknemers	Via administratie in het urenadministratie menu kunnen, door op de naam van een werknemer te klikken, ziekteverzuim en verlofuren worden geregistreerd
S	✓	De mogelijkheid om het wagenpark (auto's en aanhangwagens) te beheren	De CRUD functionaliteit die te bereiken is vanuit het data menu is hiertoe in staat
	✓	De mogelijkheid om personeelsleden toe te voegen	Vanuit het data menu kunnen werknemers beheerd worden. Hier kunnen onder andere personeelsleden worden toegevoegd
	✓	De mogelijkheid om DKV- en Schreurs passen toe te wijzen aan voertuigen	Bij het bewerken van dergelijke passen vanuit het data menu kan aangegeven worden bij welk voertuig de pas hoort
	✓	De mogelijkheid om creditcards toe te wijzen aan werknemers	Wanneer een gebruiker een creditcard bewerkt kan ook het personeelslid waarin die is toegewezen worden gewijzigd
	✓	De applicatie moet authenticatie gebruiken	Er wordt gebruik gemaakt van authenticatie met een gebruikersnaam en wachtwoord. Het wachtwoord is, versleuteld via het SHA1 algoritme, opgeslagen in de database

	✓/ ×	De applicatie kan omgaan met verschillende soorten gebruikers-accounts: administrator, kantoormedewerker en standaardgebruiker	Op dit moment hebben alleen administrators en kantoor medewerkers toegang tot de management portal. Gebruikers met andere rollen kunnen alleen gebruik maken van de app. In een ideale situatie zouden werknemers echter hun eigen geregistreerde uren kunnen inzien met het doel deze informatie te controleren.
C	×	Het registreren van defecten aan auto's en of deze al dan niet opgelost zijn	Door gebrek aan tijd is deze optie niet geïmplementeerd
W	×	Het plannen van activiteiten	Al voor de ontwerpfase is besloten dat er geen tijd beschikbaar was voor deze feature.

8.3 Technische eisen

M	✓	Activiteiten moeten verzonden kunnen worden naar de centrale database	De synchronisatie service die beschreven is in paragraaf 6.4 voegt deze functionaliteit toe.
	✓	Volledige functionaliteit op Android 2.3.7 (dit is de Android versie op de Motorola Defy+)	Omdat gebruik is gemaakt van de Motorola Defy+ met Android 2.3.7 om de app te ontwikkelen, is de app geoptimaliseerd voor deze Android versie.
	✓	Ondersteuning van de volgende browsers: Internet Explorer 9+, Google Chrome	De website is volledig getest in deze browsers en werkt naar behoren.
S	✓	De applicatie wordt ontwikkeld met behulp van het *Model, View, Controller* design-patroon	Dit design-patroon wordt volledig benut in de implementatie (zie ook paragraaf 6.3)
C	✓	Versleuteld opslaan van wachtwoorden	Wachtwoorden worden versleuteld met behulp van het SHA1 algoritme.
	×	Paginering voor pagina's waar veel data in kan komen (zoals het kasboek)	Door gebrek aan tijd is deze optie niet geïmplementeerd
W	✓	Fouten log voor administrators	Dit bleek tijdens de ontwikkeling een noodzakelijk onderdeel voor effectieve debug sessies. Deze feature is terug te bereiken via het Data menu.
	✓	Opslaan van logboek van communicatie tussen te server en de App	Ook dit bleek tijdens de ondersteuningsfase van het project een waardevolle feature om bugs op te sporen. Dankzij dit logboek is het namelijk mogelijk om dezelfde request nog eens naar de SOAP service te sturen en zo bugs te reproduceren

✓	Ajax oproepen voor schermen die modaal ingeladen worden	Het inladen van alle modaal te laden schermen in de originele pagina zou tot veel te veel nutteloze data leiden in de DOM. Daarom waren Ajax calls niks anders dan een logische stap, en worden ze gebruikt voor elke pop-up

9 Werkwijze

Tijdens het project is er nauw samengewerkt met de opdrachtgevers. Opdrachtgever**s**, omdat er twee zijn. De heer Broekmans is eigenaar van het vrachtwagenbedrijf waar de applicatie voor ontwikkeld is. De heer van Helden is manager binnen Airinfra het stagebedrijf. De projectgroep is tijdens het project drie keer naar Limburg afgereist voor overleg met beide opdrachtgevers. Voordat het project begon is de opdracht besproken tussen de twee opdrachtgevers en is er een story van de requirements gemaakt door de heer van Helden. De story(heeft iemand dit nog?) is door de projectgroep omgezet in een moscow model(Hoofdstuk 4) en een uml diagram van de te registreren diensten(Bijlage 17).

Bij het maken van de visuele interface is voor het programmeren een schets gemaakt. Deze schetsen zijn besproken tussen de interne opdrachtgever en de betreffende programmeur. Een aantal schetsen zijn door de heer van Helden uitgewerkt als mockups zoals te zien in bijlage 14 en bijlage 18.

Tijdens het porject is er gewerkt met taaklijstjes gesorteerd op prioriteit in overleg met de heer van Helden. Vanaf week 7 is er gebruik gemaak van een programma, Redmine, om bugs te rapporteren aan elkaar. Met Redmine is het mogelijk om bugs aan personen toe te wijzen en debug een prioriteit en urenschattingen te geven.

Het project is opgesplitst in drie delen: de website, de android applicatie en de database. Elk van deze delen is geschreven met behulp van andere programmeer talen. Door deze opsplitsing te maken hebben we een hoop tijd kunnen besparen. Alle code van één deel is door en voor één programmeur geschreven, hierdoor is het commentaar, het overleg en documentatie van de code beperkt gebleven. Ook is hierdoor de onderzoekstijd in de bibiliotheken van de verschillende programmeertalen minimaal gehalveerd. Integratie van deze verschillende onderdelen is gedaan door eerst met 2 man interface te ontwerpen. Dit ontwerp bevat zaken als de methodenamen, datacodering en manieren om de integratie zo effiecent en simpel mogelijk te maken.

Overleg is niet gedaan op vaste tijdstippen, maar op het moment wanneer dat nodig was. De projectgroep en de heer van Helden zaten in de zelfde ruimte tijdens het project. Kort overleg gebeurde in de ruimte voor ingewikkeld overleg werd er uitgeweken naar de vergaderzaal. De veelvuldig- en directheid van communicatie heeft veel bijgedragen aan het eindresultaat.

10 Reflectie

In dit hoofdstuk zullen wij, Jeroen Bareman, Bas Dado en Jordy van Kuijk, uitleggen hoe we het project ervaren hebben. We zullen in gaan op wat we geleerd hebben, wat we als moeilijk en wat we als makkelijk ervaren hebben. Ook zullen we uitleggen welke onderdelen van het project we als prettig ervaren hebben en wat minder prettige onderdelen waren.

10.1 Jeroen Bareman

User centered design, ontwerpen, testen, code verbeteren, documentatie, ... ,projectmanagement Ieder voorgaand punt in mijn bachelor verschillende opinies en methoden voor geleerd. Eigenlijk wil je alles gebruiken maar dat past niet in de 420 uur die je aan het project kan besteden. Vooral de prioriteit van het aantal feautures vond ik lastig te bepalen.

Wat heb ik eruit gehaald: Ik kan nu een Android app programmeren. Ik heb ervaring opgedaan op punten uit bovenstaande alinea. Je kan pas een stuk van de applicatie maken als je een ontwerp hebt, en je kan pas ontwerpen als je een stuk van de applicatie gemaakt hebt. Tijdens het project heb ik ervaren dat het loont om zoveel mogelijk eerst kladjes te maken en te overleggen.

Het was voor het eerst dat ik langer dan een week aan een project programmeren was en ja dan tel ik de vorige projecten uit mijn bachelor mee. De grote stappen snel thuis methode is achteraf gezien niet altijd handig geweest. Ik heb wel snel resultaat, maar als het ook maar een klein beetje anders moet was ik een hoop tijd kwijt. Ik ben gaandeweg in het project iets minder grotere stappen gaan nemen en heb me voorgenomen op deze manier te blijven "lopen".

Ik vond het heel leuk om met het project bezig te zijn, het project heeft mij een hoop inspiratie en uitdaging gegeven, om mezelf verder te ontwikkelen als programmeur.

10.2 Bas Dado

Ten eerste wil ik kwijt dat ik het een leuk project vond om aan te werken. Voor mij was dit het eerste software ontwikkelingstraject dat ik een in professionele sfeer van begin tot einde heb meegemaakt. Ik ben dan ook erg blij dat we een naar mijn mening goed eindproduct hebben kunnen afleveren. Ook vond ik dat de samenwerking goed is verlopen. Er zijn geen grote meningsverschillen ontstaan en ik had ook niet het idee dat er veel niveauverschil bestaat tussen de verschillende projectleden. Het feit dat elk van de projectleden een gebied van expertise heeft dat afwijkend is van de andere projectleden heeft bijgedragen aan de goede samenwerking omdat dit de taakverdeling een stuk eenvoudiger heeft gemaakt.

Ik vind het jammer dat door de beperkte hoeveelheid beschikbare tijd niet alle features die we graag zouden willen zijn geïmplementeerd. Ook vind ik het jammer dat we geen tijd hebben gehad om unit tests te schrijven en zo te verifiëren dat de implementatie correct is, maar ik heb er door de intensieve functionele tests wel vertrouwen in dat het product geen ernstige fouten meer bevat.

Wat ik met name geleerd en ervaren heb is dat het inschatten van de benodigde tijd voor het implementeren van features erg lastig is. Het is daarvoor echt nodig om een feature op te delen in kleine onderdelen die wel in te schatten zijn. Ook heb ik geleerd dat het om tijdsredenen weigeren van nieuwe features die de klant voorstelt soms echt nodig is. Technisch heb ik veel geleerd over ASP.NET en de bijbehorende MVC structuur en over het algemeen over het ontwikkelen van web applicaties. Ook het volledig opzetten van een toch redelijk complexe database was nieuw voor mij. Ten slotte heb ik nog geleerd dat het van tevoren goed nadenken over het ontwerp en over wat de klant nu echt wil ook écht tot een beter product leidt.

Over het algemeen ben ik tevreden over dit project en heb ik het gevoel veel nuttige kennis te hebben opgedaan die ik in de toekomst kan inzetten voor volgende projecten.

10.3 Jordy van Kuijk

Over het algemeen ben ik zeer tevreden over het verloop en het resultaat van het project. Ik ben erg trots op wat we met drie programmeurs en een begeleider hebben kunnen opleveren in een tijdspanne van ongeveer 11 weken. We hebben er hard voor moeten werken, maar omdat we een prettige stage-plek hadden bij Adecs Airinfra en we goed begeleid zijn, ook vanuit de TU, is het eindproduct toch iets geworden waar we naar mij mening trots op mogen zijn. Bij aanvang van het project had geen van ons ervaring met het bouwen van Android-applicaties, het opzetten van gigantische databases in Microsoft SQL-omgevingen of het bouwen van dynamische web-portals.

We wisten bij de start van het project dat het een enorme inspanning zou kosten om een applicatie van deze omvang in zes weken op te leveren. Naar mijn idee is het ons toch gelukt omdat we op een professionele manier te werk zijn gegaan. Zo hebben we bij de oriëntatie-fase allen de tijd gekregen om de programmeer-omgevingen van de verschillende applicaties (Android App en Web Portal) te leren kennen, iets dat ons allen ten goede is gekomen. In de ontwerp-fase was de communicatie van groot belang, iedereen moet het immers met elkaar eens zijn over het ontwerp van onderdelen van applicaties. Er ontstaan bijvoorbeeld vragen als: wat is het beste ontwerp en waarom, wat is het mooiste ontwerp, is dit van belang? Ik heb nog nooit zo vaak vergaderd in twee weken tijd als in de ontwerp-fase. Later besefte ik dat omdat we voor het eerst een applicatie voor een echte klant maakten, dit een essentieel onderdeel was van het software-pakket en dat de moeite die we in het ontwerp hebben gestoken het meer dan waard was, omdat dit ook reflecteert in het eindproduct. De implementatiefase heb ik ondervonden als meest stressvolle fase. Opeens was het tijd om ideëen en ontwerpen om te zetten in een werkend product. Terugkijkend op alle duizenden regels code die we hebben getypt in deze fase, vind ik het jammer dat we door de omvang van het project en de krappe deadline geen tijd hebben gevonden voor het schrijven van unit-tests. Ikzelf ben bijzonder geïnteresseerd in JavaScript en heb dan ook een poging gedaan om de plug-ins die ik geschreven heb, te testen. Helaas werd het door overvloed van ontdekte bugs en missing features in de beta versie onmogelijk om dit werk voort te zetten en is er besloten de applicatie te testen met behulp van gebruikerstests.

Wat betreft de projectgenoten ben ik zeer tevreden. Bij aanvang van het project kenden we elkaar niet goed maar al snel bleek dat we naar mijn mening goed en efficiënt konden samenwerken. Ook denk ik dat we qua niveau ongeveer op dezelfde lijn zaten. Zonder ons goede samenwerken was waarschijnlijk het nooit gelukt het product op te leveren. Ik wil hen bij deze dan ook bedanken voor de fijne samenwerking.

Adecs Airinfra was een erg fijn bedrijf om stage te lopen. Bij aankomst was mijn werkplek helemaal voorbereid en als snel voelde ik me vertrouwd met de werkomgeving. Wanneer we vragen hadden, stond er altijd iemand voor ons klaar om te helpen. Ook vond ik het erg leuk om collega's beter te leren kennen tijdens het bedrijfsuitje. Bij deze ook bedankt aan het team van Adecs Airinfra.

Ik beschouw dit project als een erg leerzame ervaring die ik zeker mee zal nemen in mijn verdere carrière.

11 Conclusie

Zoals besproken in hoofdstuk 3 ondervond Autohulpdienst Broekmans veel problemen bij hun administratie. Een softwarepakket dat voldoet aan de eisen die gesteld zijn in hoofdstuk 4 zouden deze problemen kunnen verhelpen. In de Eisen evaluatie is te zien dat eInsight aan alle belangrijke eisen voldoet: rittenstaten zijn niet langer nodig omdat alle activiteiten kunnen worden ingevoerd met behulp van een Android app. Ook kunnen de maandstaten automatisch worden opgesteld met behulp van het maand- en jaaroverzicht in de urenadministratie op de portal. Het corrigeren van eventuele fouten kan via de website op een gemakkelijke en intuïtieve manier. Het digitale kasboek is volledig functioneel. Wanneer een werknemer een activiteit invoert waarbij een transactie heeft plaatsgevonden wordt deze transactie automatisch in het kasboek geplaatst.

Alle problemen die Autohulpdienst Broekmans ondervond voordat zij eInsight gaan gebruiken zouden na het in gebruik nemen ervan opgelost zijn. Daarbovenop is eInsight een overzichtelijk geheel geworden en is het gemakkelijk in gebruik. Het project was dus een succes.

Wij hebben veel geleerd van dit project. Zowel over de ontwikkeling van een software pakket in het algemeen, als over meer specifieke zaken voor dit project zoals Android en Web development. We hopen dat Autohulpdienst Broekmans nog veel geluk zal beleven aan eInsight en wensen ze veel succes met het op orde brengen van hun administratie. Verder willen we Adecs Airinfra bedanken voor het bieden van deze stageplaats en voor de goede projectbegeleiding van Andy van Helden.

Jeroen Bareman

Bas Dado

Jordy van Kuijk

Referenties

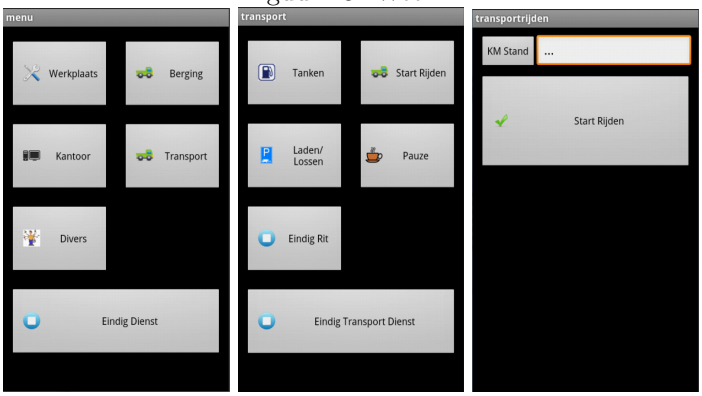
[1] Epplus - create advanced excel 2007 spreadsheets on the server. https://epplus.codeplex.com/. Bezocht in juni 2013.

[2] Twitter bootstrap framework. http://twitter.github.io/bootstrap/.

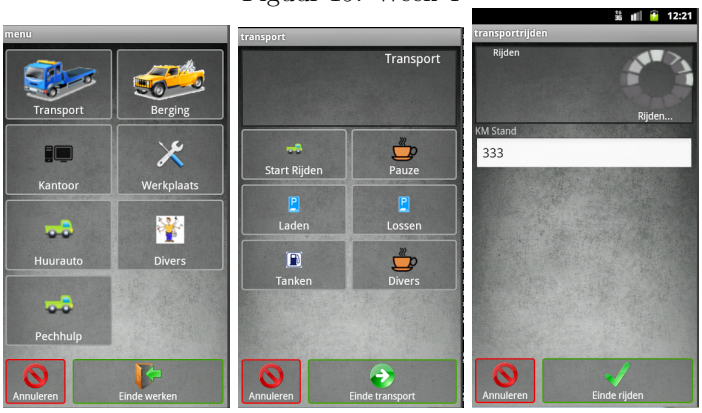
[3] Defuse Security. Salted password hashing - doing it right. http://crackstation.net/hashing-security.htm. Bezocht in mei 2013.

12 Bijlage A

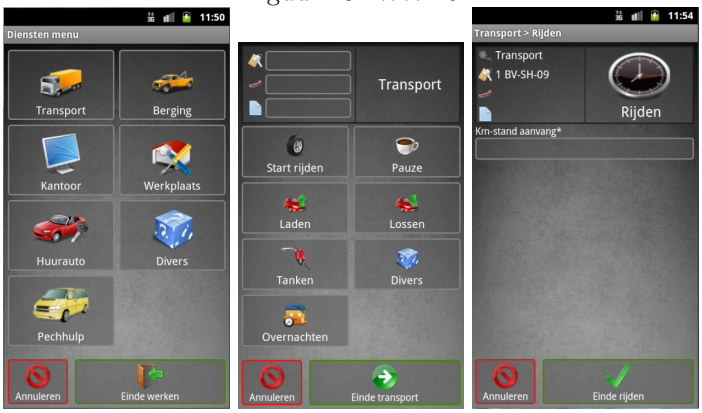
Figuur 18: Week 2



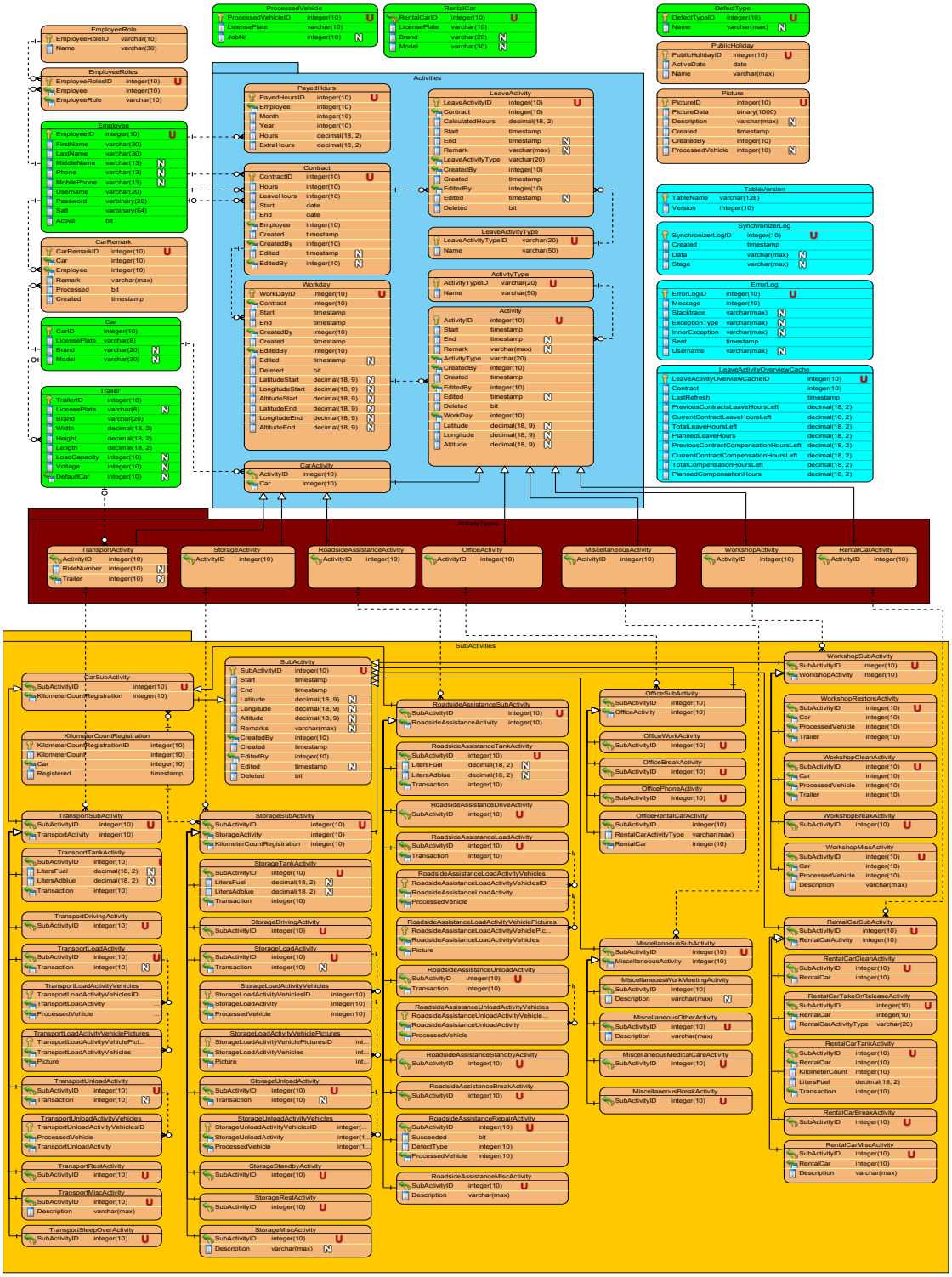
Figuur 19: Week 4

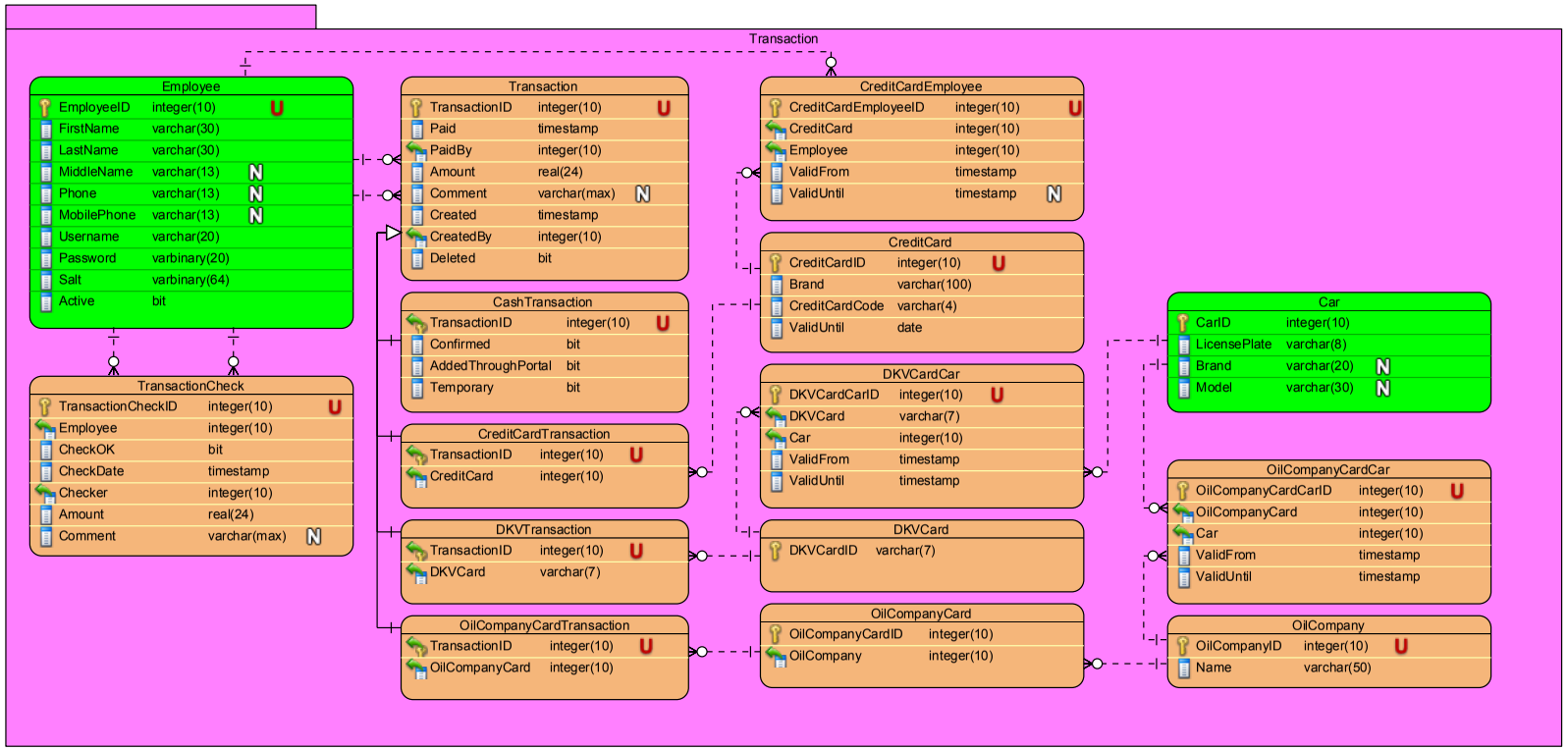


Figuur 20: Week 6

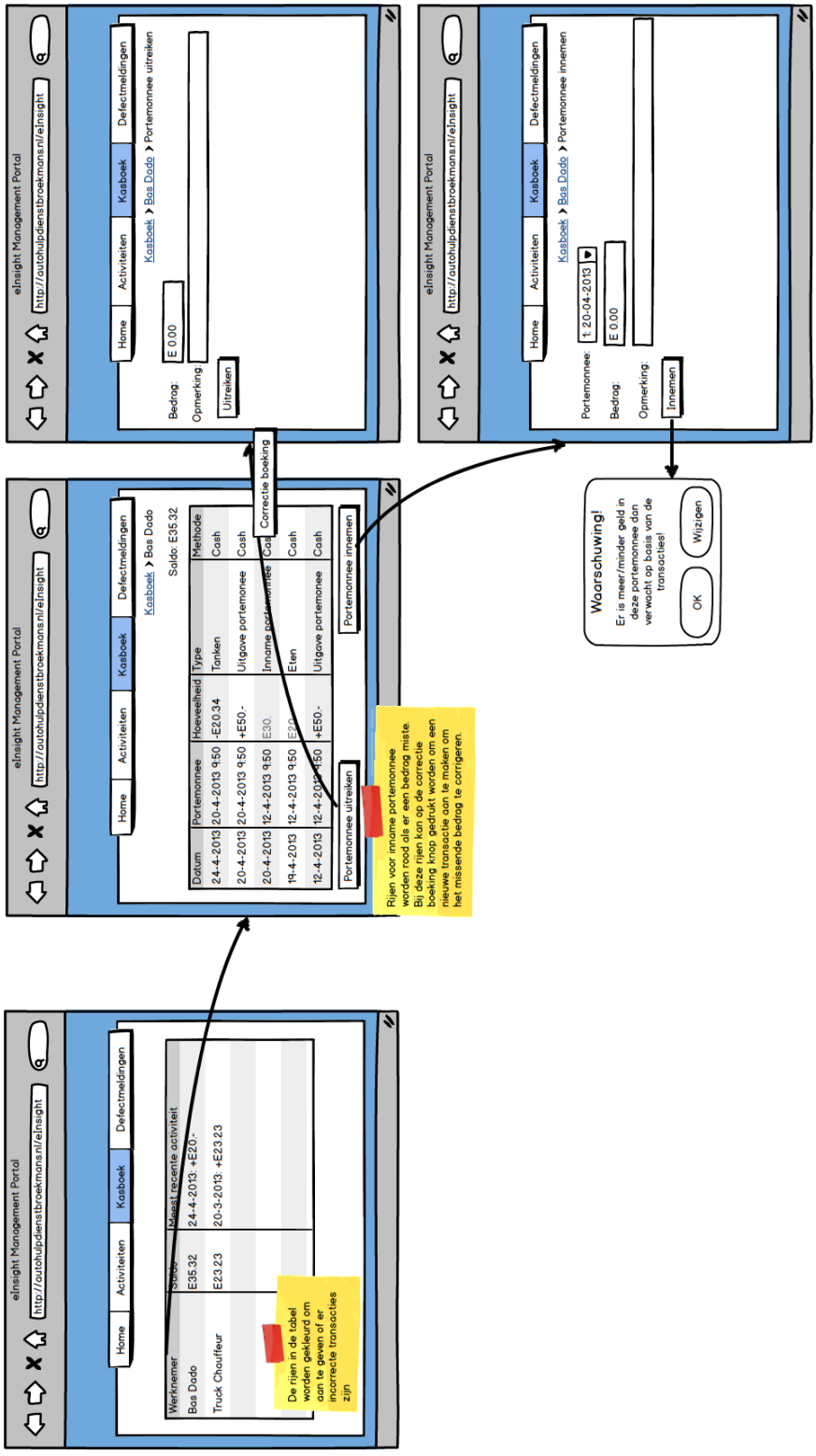


13 Bijlage B

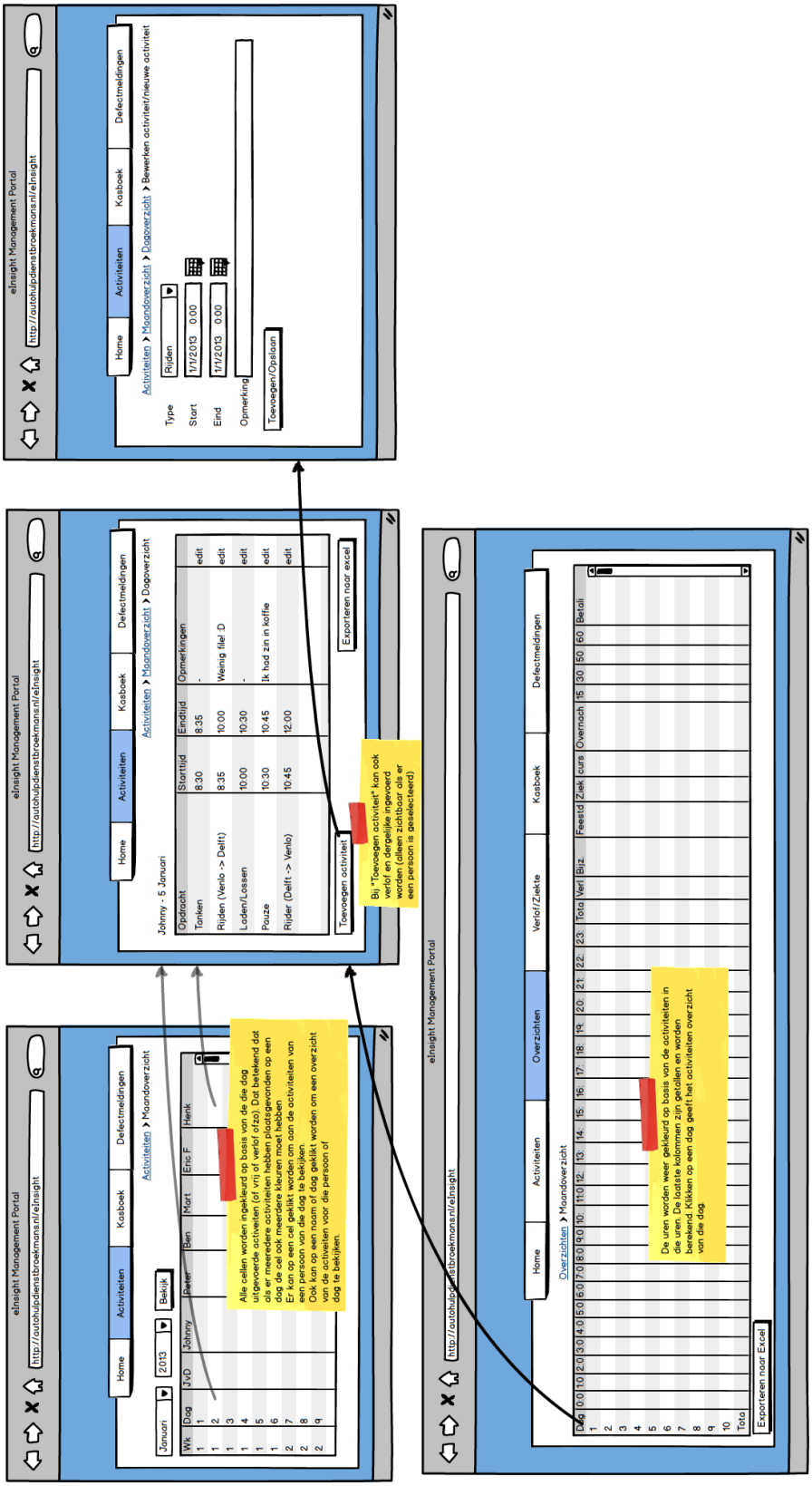

EmployeeRole
EmployeeRoles
Employee
CarRemark
Car
Trailer
ProcessedVehicle
RentalCar
DefectType
PublicHoliday
Picture
Activities
PayedHours
Contract
Workday
CarActivity
LeaveActivity
LeaveActivityType
ActivityType
Activity
TableVersion
SynchronizerLog
ErrorLog
LeaveActivityOverviewCache
ActivityTypes
TransportActivity
StorageActivity
RoadsideAssistanceActivity
OfficeActivity
MiscellaneousActivity
WorkshopActivity
RentalCarActivity
SubActivities
CarSubActivity
KilometerCountRegistration
SubActivity
RoadsideAssistanceSubActivity
RoadsideAssistanceTankActivity
RoadsideAssistanceDriveActivity
RoadsideAssistanceLoadActivity
RoadsideAssistanceLoadActivityVehicles
RoadsideAssistanceLoadActivityVehiclePictures
RoadsideAssistanceUnloadActivity
RoadsideAssistanceUnloadActivityVehicles
RoadsideAssistanceStandbyActivity
RoadsideAssistanceBreakActivity
RoadsideAssistanceRepairActivity
RoadsideAssistanceMiscActivity
OfficeSubActivity
OfficeWorkActivity
OfficeBreakActivity
OfficePhoneActivity
OfficeRentalCarActivity
MiscellaneousSubActivity
MiscellaneousWorkMeetingActivity
MiscellaneousOtherActivity
MiscellaneousMedicalCareActivity
MiscellaneousBreakActivity
WorkshopSubActivity
WorkshopRestoreActivity
WorkshopCleanActivity
WorkshopBreakActivity
WorkshopMiscActivity
RentalCarSubActivity
RentalCarCleanActivity
RentalCarTakeOrReleaseActivity
RentalCarTankActivity
RentalCarBreakActivity
RentalCarMiscActivity
TransportSubActivity
TransportTankActivity
TransportDrivingActivity
TransportLoadActivity
TransportLoadActivityVehicles
TransportLoadActivityVehiclePictures
TransportUnloadActivity
TransportUnloadActivityVehicles
TransportRestActivity
TransportMiscActivity
TransportSleepOverActivity
StorageSubActivity
StorageTankActivity
StorageDrivingActivity
StorageLoadActivity
StorageLoadActivityVehicles
StorageLoadActivityVehiclePictures
StorageUnloadActivity
StorageUnloadActivityVehicles
StorageStandbyActivity
StorageRestActivity
StorageMiscActivity


Transaction
Employee
EmployeeID integer(10) U
FirstName varchar(30)
LastName varchar(30)
MiddleName varchar(13) N
Phone varchar(13) N
MobilePhone varchar(13) N
Username varchar(20)
Password varbinary(20)
Salt varbinary(64)
Active bit
TransactionCheck
TransactionCheckID integer(10) U
Employee integer(10)
CheckOK bit
CheckDate timestamp
Checker integer(10)
Amount real(24)
Comment varchar(max) N
Transaction
TransactionID integer(10) U
Paid timestamp
PaidBy integer(10)
Amount real(24)
Comment varchar(max) N
Created timestamp
CreatedBy integer(10)
Deleted bit
CashTransaction
TransactionID integer(10) U
Confirmed bit
AddedThroughPortal bit
Temporary bit
CreditCardTransaction
TransactionID integer(10) U
CreditCard integer(10)
DKVTransaction
TransactionID integer(10) U
DKVCard varchar(7)
OilCompanyCardTransaction
TransactionID integer(10) U
OilCompanyCard integer(10)
CreditCardEmployee
CreditCardEmployeeID integer(10) U
CreditCard integer(10)
Employee integer(10)
ValidFrom timestamp
ValidUntil timestamp N
CreditCard
CreditCardID integer(10) U
Brand varchar(100)
CreditCardCode varchar(4)
ValidUntil date
DKVCardCar
DKVCardCarID integer(10) U
DKVCard varchar(7)
Car integer(10)
ValidFrom timestamp
ValidUntil timestamp
DKVCard
DKVCardID varchar(7)
OilCompanyCard
OilCompanyCardID integer(10)
OilCompany integer(10)
Car
CarID integer(10)
LicensePlate varchar(8)
Brand varchar(20) N
Model varchar(30) N
OilCompanyCardCar
OilCompanyCardCarID integer(10) U
OilCompanyCard integer(10)
Car integer(10)
ValidFrom timestamp
ValidUntil timestamp
OilCompany
OilCompanyID integer(10) U
Name varchar(50)

14 Bijlage C

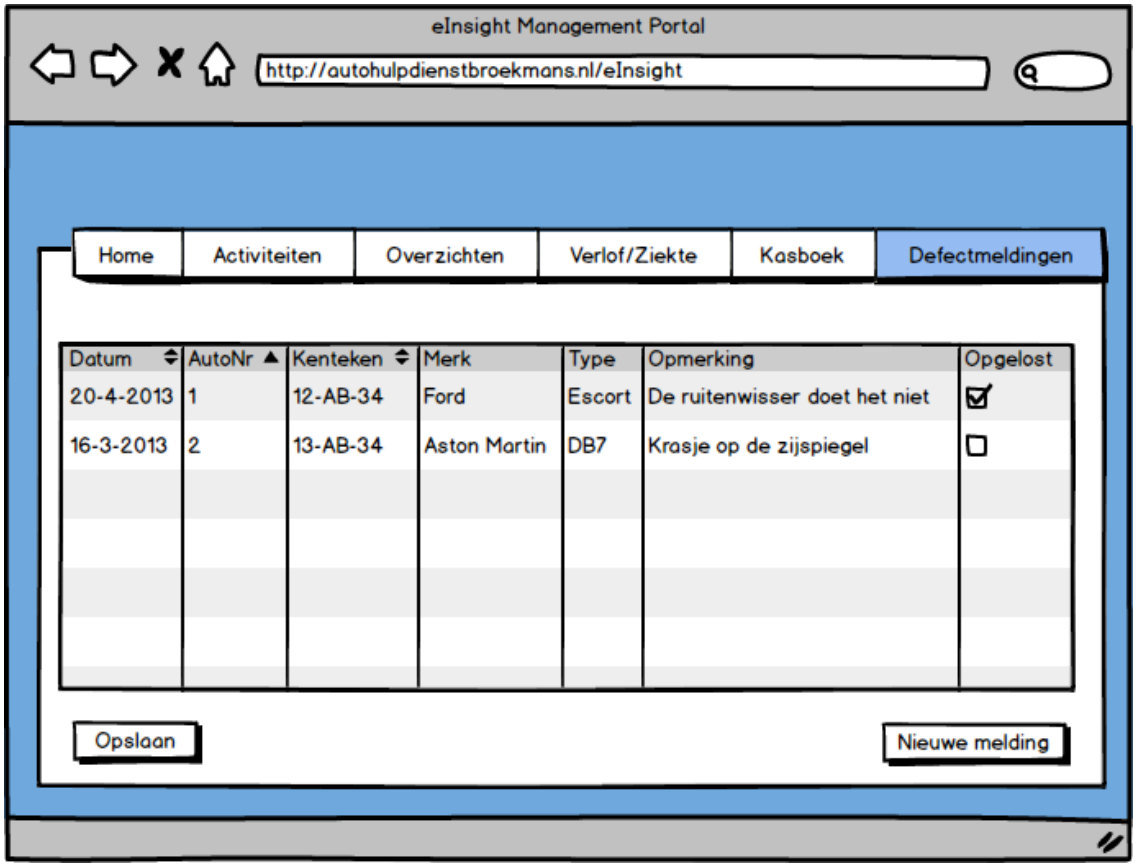


Figuur 21: Management portal mockups van het kasboek



Figuur 22: Management portal mockups van het activiteiten overzicht en detail overzicht

Figuur 23: Management portal mockup van het urenadministratie overzicht



Figuur 24: Management portal mockup van het defectmeldingen deel. Dit zit niet meer in de uiteindelijke website

Figuur 25: Management portal mockups van het verlof/ziekte registratie deel

15 Bijlage D

SIG Aanbevelingen

De code van het systeem scoort 3 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code gemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere score voor Unit Size, Unit Complexity en Duplication.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. Binnen de langere methodes in dit systeem, zoals bijvoorbeeld de '`ToModel`'-methode in de '`HtmlHelperExtensions`'-class, zijn aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methodes. Commentaarregels zoals bijvoorbeeld '`// Process transport sub activities`' en '`// Process storage sub activities`' zijn een goede indicatie dat er een autonoom stuk functionaliteit te ontdekken is. Het is aan te raden kritisch te kijken naar de langere methodes binnen dit systeem en deze waar mogelijk op te splitsen. Let erop dat er (te) lange methodes te vinden zijn in alle hoofd-technologien binnen dit systeem, deze aanbeveling is dus van toepassing op zowel Java, C# als de JavaScript.

Voor Unit Complexity wordt er gekeken naar het percentage code dat bovengemiddeld complex is. Ook hier geldt dat het opsplitsen van dit soort methodes in kleinere stukken ervoor zorgt dat elk onderdeel makkelijker te begrijpen, makkelijker te testen en daardoor eenvoudiger te onderhouden wordt. In dit geval komen de meest complexe methoden ook naar voren als de langste methoden, waardoor het oplossen van het eerste probleem ook dit probleem zal verhelpen.

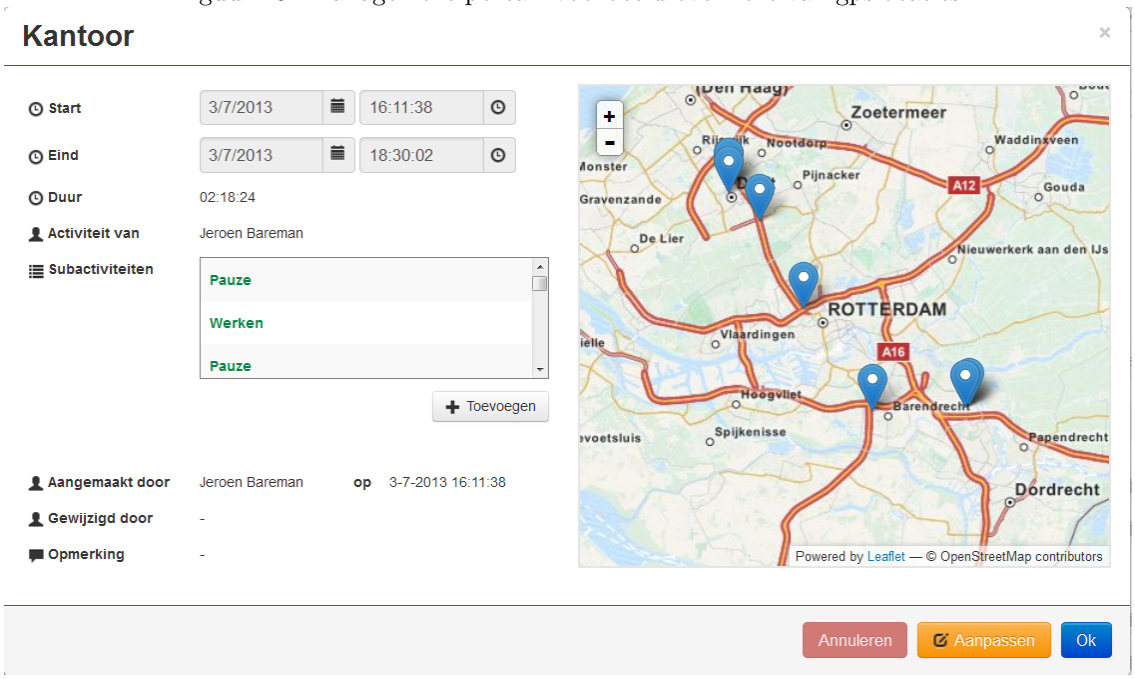
Voor Duplicatie wordt er gekeken naar het percentage van de code welke redundant is, oftewel de code die meerdere keren in het systeem voorkomt en in principe verwijderd zou kunnen worden. Vanuit het oogpunt van onderhoudbaarheid is het wenselijk om een laag percentage redundantie te hebben omdat aanpassingen aan deze stukken code doorgaans op meerdere plaatsen moet gebeuren. In dit systeem is er op verschillende plekken duplicatie te vinden. Binnen de Java-code is dezelfde code te vinden in verschillende controllers, terwijl er binnen de C# duplicatie is te vinden binnen de '`CashBookController`' (bijvoorbeeld onder het commentaar '`// add additional data to the model.`'). Het is aan te raden de duplicatie binnen dit systeem op te sporen en te reduceren.

Verder vallen er twee dingen op binnen dit project. Het eerste is dat verschillende onderdelen van het systeem door verschillende personen zijn gemaakt, wat ervoor kan zorgen dat niemand het overzicht heeft van het gehele systeem. Het is aan te raden dit overzicht te bewaren door bijvoorbeeld goede documentatie of code-reviews. Het tweede is dat er gegenereerde code naast handgeschreven code staat. Het risico is hier dat men aanpassingen gaat maken aan de initieel gegenereerde code, hierdoor is het dan niet meer mogelijk om de code te her-genereren. Om dit risico te beperken is het aan te raden om deze gegenereerde code apart te zetten en ervoor te zorgen dat de code-generatie bij elke compilatie plaatsvindt.

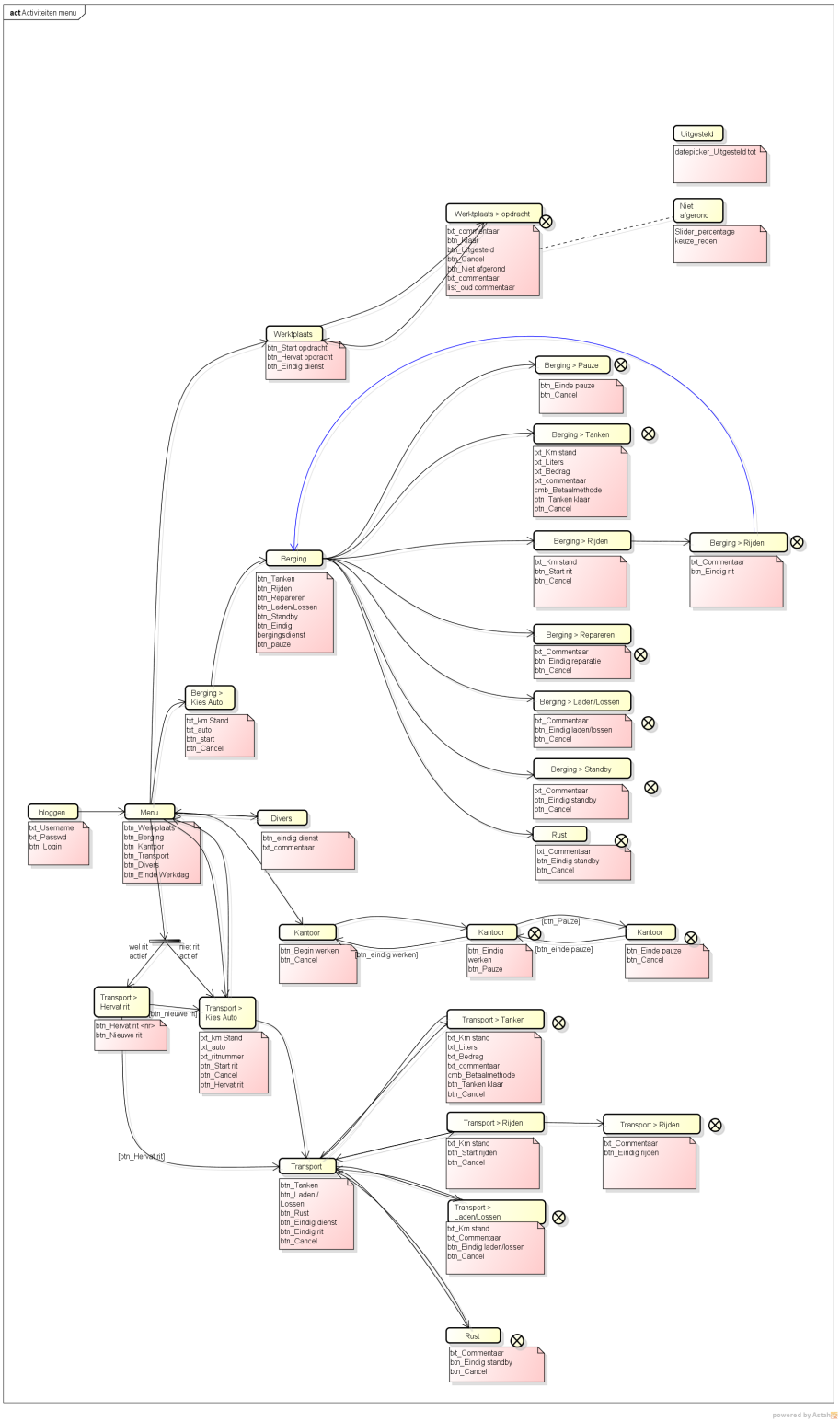
Over het algemeen scoort de code gemiddeld, hopelijk lukt het om dit niveau te behouden of te verbeteren tijdens de rest van de ontwikkelfase. Als laatste nog de opmerking dat er geen (unit)test-code is gevonden in de code-upload. Het is sterk aan te raden om in ieder geval voor de belangrijkste delen van de functionaliteit automatische tests gedefinieerd te hebben om ervoor te zorgen dat eventuele aanpassingen niet voor ongewenst gedrag zorgen.

16 Bijlage E

Figuur 26: Management portal: voorbeeld overzicht van gpslocaties

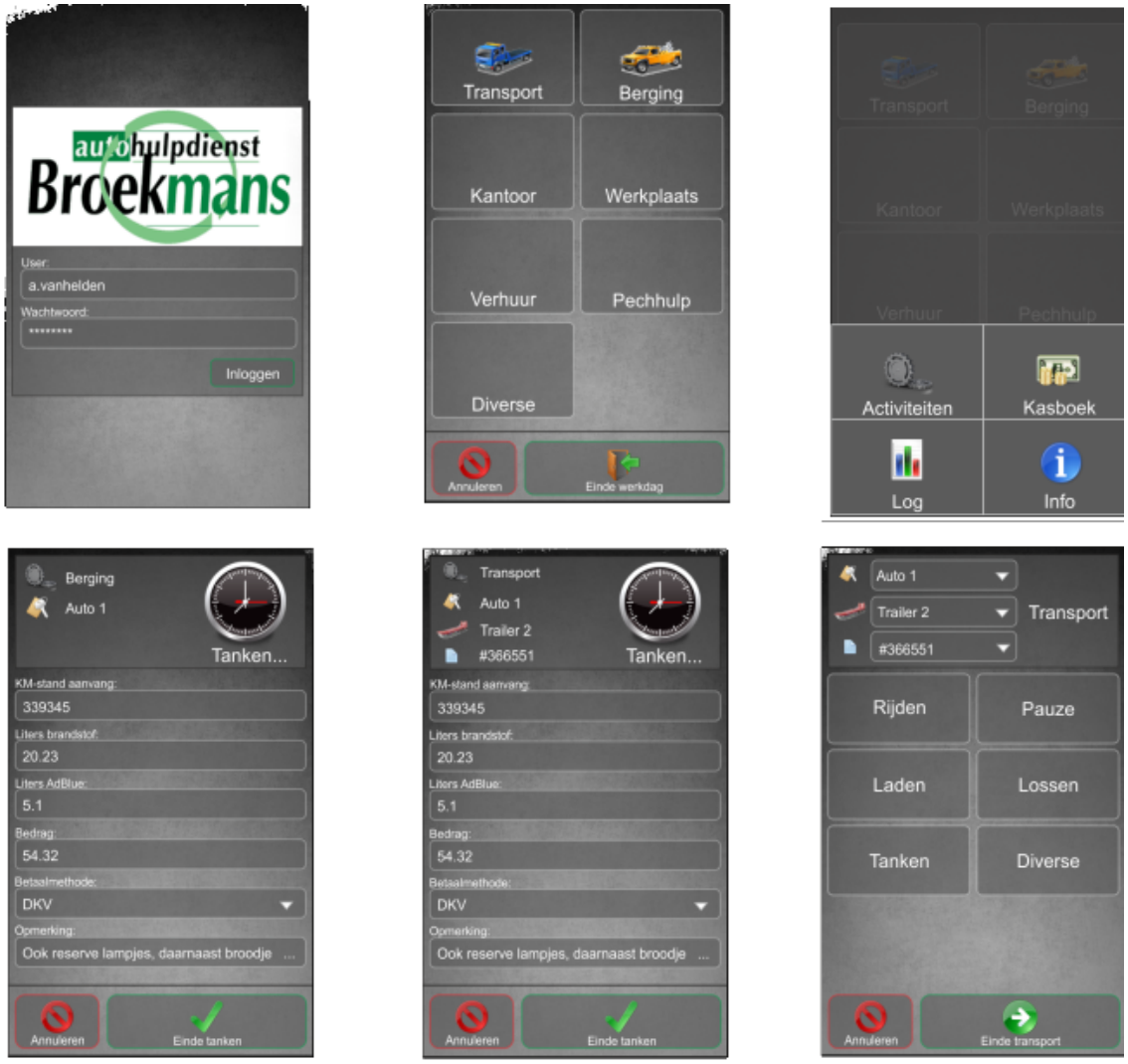


17 Bijlage F



Figuur 27: Eerste ontwerp diensten, de pijlen stellen schermovergangen voor

18 Bijlage G

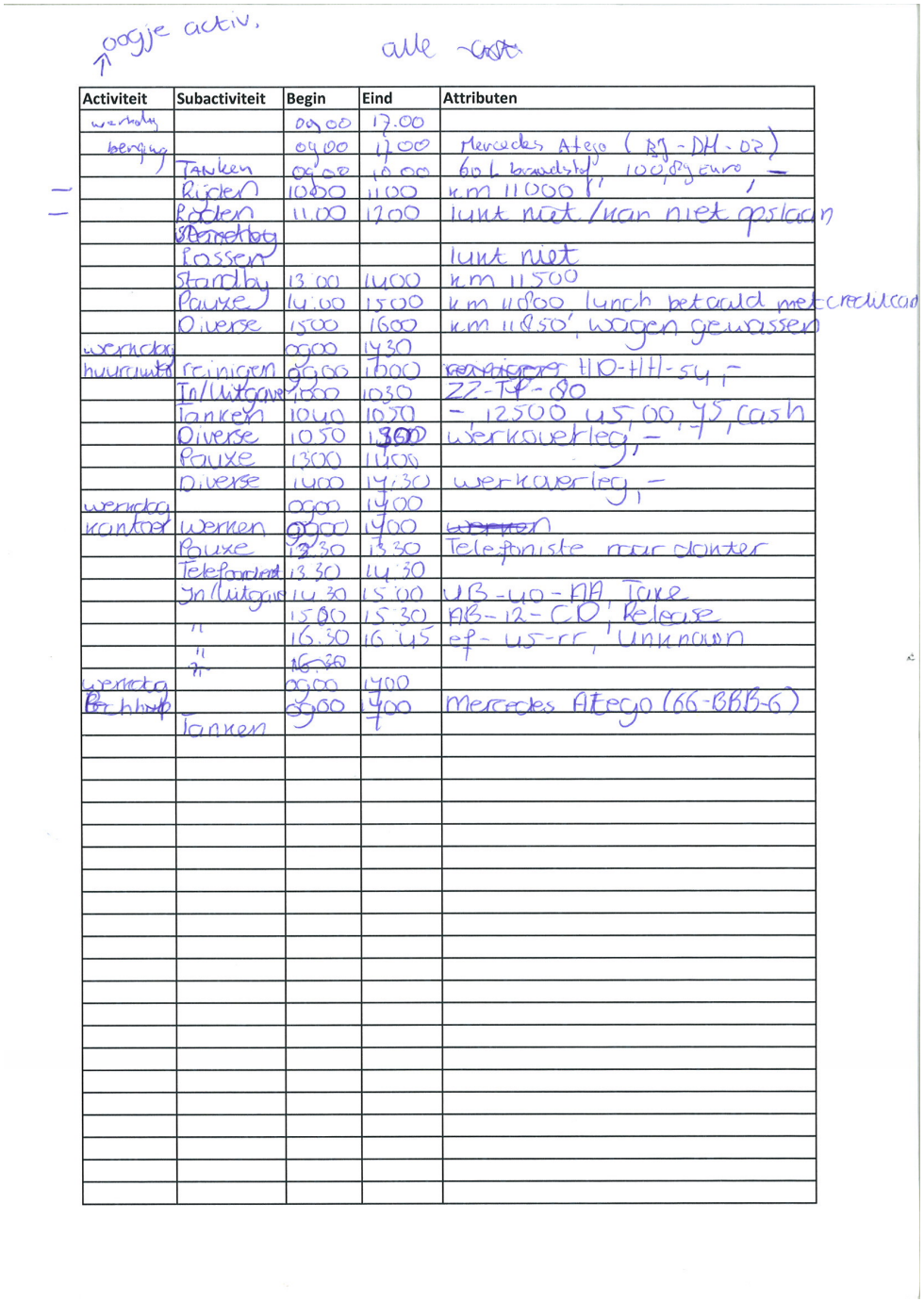


Figuur 28: Mockup app schermpjes

19 Bijlage H

Figuur 29: Huidige dienstregistratie Autohulpdient Broekmans

20 Bijlage I



oogje activ.

alle kosten

Activiteit	Subactiviteit	Begin	Eind	Attributen
werkdag		09.00	17.00	
bergung		09.00	17.00	Mercedes Atego (BJ-DH-02)
	Tanken	09.00	10.00	60 L brandstof, 100,89 euro, —
	Rijden	10.00	11.00	km 11000
	Rijden	11.00	12.00	kunt niet / kan niet opslaan
	~~Storneldag~~			
	Passen			kunt niet
	Standby	13.00	14.00	km 11500
	Pauze	14.00	15.00	km 11500, lunch betaald met creditcard
	Diverse	15.00	16.00	km 11850, wagen gewassen
werkdag		09.00	14.30	
huurauto	reinigen	09.00	16.00	~~reinigen~~ HD-HH-54,-
	In/Uitgave	16.00	10.30	ZZ-TP-80
	Tanken	10.40	10.50	-, 12500 45,00, J5, cash
	Diverse	10.50	13.00	werkoverleg,-
	Pauze	13.00	14.00	
	Diverse	14.00	14.30	werkoverleg,-
werkdag		09.00	14.00	
kantoor	werken	09.00	14.00	~~werken~~
	Pauze	13.30	13.30	Telefoniste naar donker
	Telefoonlijst	13.30	14.30	
	In/Uitgave	14.30	15.00	VB-40-AA, Jake
	"	15.00	15.30	AB-12-CD, Kelcise
	"	16.30	16.45	ef-45-rr, Unknown
	"	16.30		
werkdag		09.00	14.00	
Berkhout		09.00	14.00	Mercedes Atego (66-BBB-6)
	Tanken			

Figuur 30: Gebruikerstest van de Management Portal. Het formulier is als hulpmiddel gebruikt ter verificatie van de resultaten.

Activiteit	Subactiviteit	Begin	Eind	Attributen
begin werkdag		20.30	21:56	
Transp		20:32		Auto 1, Trailer 1, Rit 1
	start rijden	20:33	20:34	km 0000001
	Pauze	20:34	20:35	km 0000002
	Laden	20:36	20:41	km 0000003, opdr 1, opm: Test 1, xx-yy-22, €45,85
	Lossen	20:42	20:43	km 0000004, ~~opdr~~, TT-RR-44, Cash
	Tanken	20:43	20:44	km 0000005, 25 L, 38 L, €255,55, Test
	Divers	20:44	20:45	km 0000006, Test 5
	Overwicht	20:45	20:46	km 0000007
Bergung		20:58	21:11	Auto 2,
	Standby	20:58	21:00	0000100
	Rijden	21:00	21:00	0000200
	Pauze	21:00	21:01	—
	Laden	21:01	21:03	~~RR-44~~ 0000300, RR-44-22, €25,55, Creditcard
	Lossen	21:03	21:06	0000400, FF-MM-GH, €25, Cash, ~~Gold~~,
	Tanken	21:06	21:09	0000500, 25, 23, €200, Diesel
→	Divers	21:10	21:10	0000600, Test 4
Kraan		21:41	21:43	
	Pauze	21:41	21:41	
	Werken	21:41	21:41	—
	Werken	21:41	21:42	Test
		~~21:42~~		
Bergung				
Kraan		21:45	21:47	
	telefoon	21:45	21:45	
	In-/uitgaan	21:45	21:46	Test 7 FFGGHHMM66
	In/uit	21:47	21:47	GG~~EE~~66
~~Schaftpauze~~				
Werkplaats		21:52		
	Herstel	21:52	21:52	Test, T66666
→	Reinigen	21:52	21:55	3, 4
	Pauze	21:56	21:56	1 1
	Divers	21:5	21:56	6yg
begin uitval	Werkdag			
Kraan		22:02	22:00	
	Reiniging	22:02	22:03	FG66G, ...
	Inrukgave	21:03	22:07	GGGHH, —
	Tanken	22:03	22:04	0000657, 457, 45, FG6, Cash, FG4
	Divers	22:05	2207	577 12gggff
	Pauze	22:07	2207	

21 Bijlage J

Bagijnhof 80
2611 AR Delft
T 015 - 215 00 40
F 015 - 214 57 12
info@adecs-airinfra.nl
www.adecs-airinfra.nl

Autohulpdienst Broekmans
T.a.v de heer M. Broekmans
Colombusweg 11
5928 LA VENLO

Delft, 19 februari 2013

Uw kenmerk : -
Ons kenmerk : ab130205/aH/kd
Contactpersoon : ir. A. van Helden
Betreft : Offerte eInsight

Geachte heer Broekmans,

Zoals 15 februari 2013 besproken doen wij u een offerte toekomen voor een softwarepakket om u beter inzicht te geven in de uitgevoerde werkzaamheden binnen uw bedrijf en de door uw personeel gerealiseerde uren. In deze offerte zal naar dit systeem verwezen worden als eInsight.

In deze offerte is een overzicht gegeven van de functionaliteiten van het eInsight-systeem op basis van de bespreking op 15 februari. Aansluitend wordt de opbouw en werking van het systeem besproken. Vervolgens wordt er aan de hand van een viertal fases de ontwikkeling van het systeem beschreven en een aantal vereisten besproken. Afsluitend is een planning en overzicht van de kosten voor de ontwikkeling van eInsight opgenomen.

Functionaliteiten

Op basis van de bespreking van 15 februari jl. zijn de volgende functionaliteiten van eInsight gedefinieerd:

- Registratie van de activiteiten uitgevoerd door het personeel van Autohulpdienst Broekmans, welke in de huidige bedrijfsvoering worden verzameld door middel van rittenstaten.
- Registratie van werktijden die in de huidige bedrijfsvoering wordt verzameld door middel van de weekstaten.
- Bijhouden van een digitaal kasboek per personeelslid.
- Interactieve export van de verzamelde informatie naar Microsoft Excel voor analyse.
- Interactieve berekeningen van de door het personeel gerealiseerde uren, toegekende toeslagen en overuren.
- Interactieve registratie van ziekteverzuim en verlofuren van werknemers.

Systeem opbouw en werking

Het eInsight-systeem is opgebouwd uit twee applicaties: de eInsight App en eInsight Management Portal.

De eInsight App is een mobiele applicatie welke gebruikt zal worden door het personeel, om de door hen uitgevoerde werkzaamheden te registeren. Bij het invoeren van de werkzaamheden kan de gebruiker daarnaast andere, al dan niet verplichte, informatie toevoegen. Wanneer het personeel zich binnen het bereik van een toegankelijk draadloos netwerk begeeft dan zal de mobiele applicatie contact zoeken met dit netwerk om de verzamelde gegevens te synchroniseren met een centrale database. Indien het draadloze netwerk niet beschikbaar is zullen de gegevens bewaard blijven op het mobiele apparaat.

De eInsight Management Portal is een web-based applicatie welke toegang biedt tot de verzamelde gegevens. Via de Management Portal kunnen de gegevens ingezien worden en geëxporteerd naar Microsoft Excel voor verdere analyse. Verder kan een gebruiker hier verlofdagen en ziekteverzuim registreren en inzien. De Management Portal biedt daarnaast toegang tot de gerealiseerde uren, de toegekende toeslagen en gemaakte overuren.

Naast het inzien en exporteren van gegevens kan de eInsight Management Portal gebruikt worden om de kasboeken van werknemers te beheren. Indien aan een werknemer geld wordt uitgereikt kan het bedrag worden bijgeschreven in het digitale kasboek van de werknemer. De werknemer kan vervolgens het uitgereikte geld controleren, en dient via de eInsight App zijn akkoord te geven. Wanneer er uitgaven worden gedaan door de werknemer, dienen deze in de eInsight App geregistreerd te worden. Via zowel de Management Portal alsmede de App heeft men inzicht in de uitgaven van de werknemer, en de resterende kas.

Ontwikkelproces

De realisatie van het eInsight-systeem zal in nauwe samenwerking met de opdrachtgever gebeuren. Het proces zal bestaan uit vier fases:

1: Voorontwerp

In de eerste fase wordt aan de hand van de in deze offerte gespecificeerde functionaliteiten een voorontwerp gemaakt. Het voorontwerp bestaat uit een functioneel ontwerp, een schetsontwerp van de interface. In dit voorontwerp wordt ook vastgelegd welke werkzaamheden, activiteiten en gegevens bijgehouden dienen te worden door het systeem. Deze fase wordt afgesloten met een overleg waarin het voorontwerp wordt doorgesproken en getoetst aan de wensen van Autohulpdienst Broekmans. Op basis van dit overleg wordt het voorontwerp eventueel aangepast en zal in het verdere ontwikkelproces als leidraad dienen.

2: Ontwerp

In de tweede fase wordt het voorontwerp uitgewerkt tot een alfaversie waarin de belangrijkste functionaliteiten en de mens-machine interactie beschikbaar zijn. Deze applicatie zal vervolgens gedemonstreerd worden tijdens een faseafsluitend overleg. Dit overleg biedt de mogelijkheid om sturing te geven aan de verdere ontwikkeling binnen de lijnen uitgezet in het voorontwerp. Op basis van dit overleg wordt het ontwerp vastgelegd.

3: Bouw

Op basis van het ontwerp wordt in de bouwfase een bètaversie ontwikkeld. Gedurende deze periode krijgt Autohulpdienst Broekmans de mogelijkheid om de applicatie te testen en terugkoppeling te geven. Indien nodig wordt nog een release candidate versie beschikbaar gesteld om eventuele aanpassingen te laten testen.

4: Oplevering

In de laatste fase wordt overgaan tot de oplevering van het eInsight-systeem. De oplevering gebeurt tijdens een acceptatieoverleg om de ontwikkeling af te sluiten. Mocht op een later tijdstip blijken dat er nog bugs in het systeem aanwezig zijn zullen deze opgelost worden door Adecs Airinfra. Met bugs bedoelen we fouten in de in deze offerte gespecialiseerde functionaliteit.

Vereisten

Voor de installatie van de eInsight Management Portal is een systeem nodig met daarop Microsoft SQL Server 2012 R2 Express editie. Tevens moet op deze computer minimaal Microsoft.NET Framework 4.0 geïnstalleerd zijn. Verder dient deze computer aangesloten te zijn op een draadloos netwerk.

De eInsight App wordt ontwikkeld voor een smartphone welke is uitgerust met Android als besturingssysteem. De smartphone of andere mobile device dient verder uitgerust te zijn met GPS en Wifi connectiviteit bieden.

Om het maximale gebruikersgemak te realiseren wordt de applicatie geoptimaliseerd voor een nader overeen te komen type smartphone. Autohulpdienst Broekmans dient voor aanvang van de voorontwerpfase aan te geven voor welke type smartphone de applicatie geoptimaliseerd dient te worden. De specificaties van dit toestel zullen ook dienen als de minimale eis voor andere, door de eInsight App ondersteunde, smartphones. Indien gewenst kan Adecs Airinfra advies geven over een geschikte smartphone maar de verantwoordelijkheid voor de uiteindelijke keuze ligt bij Autohulpdienst Broekmans.

De benodigde hardware zal niet worden geleverd door Adecs Airinfra en er wordt van uitgegaan dat Autohulpdienst Broekmans hiervoor tijdig zelf zorg draagt. Er zal wel assistentie verleend worden door Adecs Airinfra bij het installeren van de eInsight software.

Voor een optimale ontwikkeling is het nodig om eInsight op een aantal momenten te kunnen testen op de beoogde hardware. In deze offerte wordt ervan uitgegaan dat de ontwikkelaars van Adecs Airinfra, in overleg met Autohulpdienst Broekmans, de mogelijkheid krijgen om op de vereiste hardware te testen.

Planning en kosten

Een gedeelte van de ontwikkeling van eInsight wordt uitgevoerd in het kader van een afstudeerproject door afstudeerders van de TU Delft onder begeleiding van Adecs Airinfra. In het kader hiervan is voor deze opdracht een ruimere planning aangehouden dan gebruikelijk en wordt

22 Bijlage O

Orientatieverslag Bachelorproject eInsight

Jeroen Bareman, Bas Dado, Jordy van Kuijk

5 juli 2013

Inhoudsopgave

1 Inleiding

De eerste fase van het project is de oriëntatiefase. We oriënteren ons in deze fase op de opdracht, de werkomgeving van Adecs Airinfra en er wordt naar bestaande *libraries*, *frameworks* en *API*'s gezocht die ons kunnen helpen bij het ontwikkelen van de applicaties. Zoals aangegeven in de tijdsplanning in het Plan van Aanpak, is er tijd ingeroosterd voor het opstellen van dit verslag. Omdat zowel de Management Portal als de Android Applicatie vanaf de grond worden opgebouwd, is het belangrijk dat we met behulp van onze ervaringen en met die van het bedrijf een keuze maken uit het brede spectrum aan applicaties, tools en frameworks die voor ons beschikbaar zijn, goede software af te leveren.

2 Projectopdracht en uitdagingen

2.1 Inleiding

De projectopdracht, zoals beschreven in hoofdstuk 2 van het Plan van Aanpak, introduceert een aantal uitdagingen, zowel op ontwerp-technisch, als op implementatieniveau. Dit hoofdstuk beschrijft eerst bondig hoe de projectopdracht tot stand is gekomen en daarna welke uitdagingen er liggen in zowel het ontwerp als de implementatie van het software-pakket eInsight.

2.2 Projectopdracht

2.2.1 Totstandkoming

Voorafgaand aan het project heeft Adecs Airinfra gesprekken gevoerd met opdrachtgever Autohulpdienst Broekmans. Op basis hiervan is een offerte opgesteld met daarin de eisen waaraan de op te leveren software moet voldoen. Ook heeft Adecs Airinfra op voorhand een recommandatie gedaan over welke smartphones de opdrachtgever moet aanschaffen ten behoeven van dit project.

2.2.2 Adecs Airinfra

Adecs Airinfra gebruikt intern voornamelijk applicaties van Microsoft voor de ontwikkeling van software, zoals Microsoft Visual Studio 2008 en de Microsoft SQL server-omgeving. Omdat één van de projectgenoten ervaring heeft met deze ontwikkeltools, is er besloten dat het team gebruik maakt van deze beschikbare technologiën voor de ontwikkeling van het eInsight software-pakket.

2.2.3 Uitdagingen

Op basis van de de eisen opgesteld in de offerte tijdens de totstandkoming van het project en de interne werkwijze van Adecs Airinfra voor de ontwikkeling van software, is er tijdens de oriëntatiefase gebrainstormd over het ontwerp van de applicatie.

Voor dit project moeten er twee applicaties worden opgeleverd: *eInsight App*: een Android-applicatie die draait op smartphones van medewerkers en de *eInsight Management Portal*, die data opgeslagen door de Android-applicatie inzichtelijk en bewerkbaar maakt voor administratoren. Deze applicaties moeten onderling kunnen communiceren via een service. Een van de grootste uitdagingen hierbij is dat het team niet één, maar twee applicaties vanaf de grond af aan op moet bouwen in een relatief korte tijd. Bijkomend heeft niemand binnen het team ervaring met het ontwikkelen van Android Applicaties.

De stagebegeleider heeft *mockups* van schermen van de applicatie gemaakt, die gebruikt zijn tijdens de eerste gesprekken met Autohulpdienst Broekmans. Deze zijn gebruikt tijdens brainstormsessies over het ontwerp van de eInsight App. Wat betreft het uiteindelijke ontwerp van zowel de eInsight-software zijn we behoorlijk vrij. Het is wel belangrijk de App goed te gebruiken is door werknemers van Autohulpdienst Broekmans. Dit kan lastig zijn omdat velen hiervan geen ervaring hebben met het gebruik van smartphones. De stagebegeleider spreekt liefst de ontwerpen van de schermen door met het team alvorens deze geïmplementeerd worden.

Voor de portal is de grootste uitdaging om alle data overzichtelijk op te slaan en weer te geven. Met name het weergeven van alle activiteiten van alle werknemers in een overzicht zonder data weg te laten is een van de grootste uitdagingen binnen de Management Portal.

3 Technische oriëntatie

Omdat veel libraries en APIs nog in ieder geval voor een groot deel onbekend waren voor de projectleden hebben we veel tijd besteed aan technische oriëtatie. Hieronder staat per onderdeel wat de projectleden hebben gedaan om zich te oriënteren en welke belangrijke bevindingen en/of beslissingen hieruit zijn gekomen. Ook is er een proof-of-concept van de eInsight App gemaakt.

3.1 eInsight App

Om zich te oriënteren heeft de projectgroep besloten om ieder de officiële android tutorial te volgen [3]. Hierna zijn we elk aan de slag gegaan met kleine testprojecten.

Bas is gaan werken aan een test project om de SQLite database die standaard in de Android API zit aan te spreken. Hiervoor is wederom gebruik gemaakt van een tutorial [9]. Om SQLite te testen is een kleine applicatie gemaakt die gebruikt maakt van een database die bestaat uit slechts een tabel met twee kolommen (een primary key en een string). Deze tabel kon dan vervolgens via een Android interface gevuld en uitgelezen worden. Het gebruik van de database is belangrijk voor de applicatie om informatie op te slaan als de gebruiker geen internettoegang heeft. Zo moeten bijvoorbeeld activiteiten en transacties altijd kunnen worden toegevoegd.

Jordy heeft gewerkt aan het schrijven van een proof-of-concept versie van het kasboek, genaamd "Cashier". Binnen de applicatie kan geld bijgestort worden voor het account van de gebruiker. Daarna kan deze transacties invoeren, aanpassen en verwijderen, ook wel *CRUD*(Create, Read, Update, Delete)-operaties genaamd. Bovenin wordt het saldo van het kasboek van de gebruiker weergegeven. De applicatie maakt gebruik van een lokale SQLite database om transacties in op te slaan en is evenals de eInsight Management Portal opgebouwd met behulp van het MVC design pattern. De bestandsstructuur en klassen van Cashier kunnen eventueel gebruikt worden in de definitieve versie van de eInsight App.

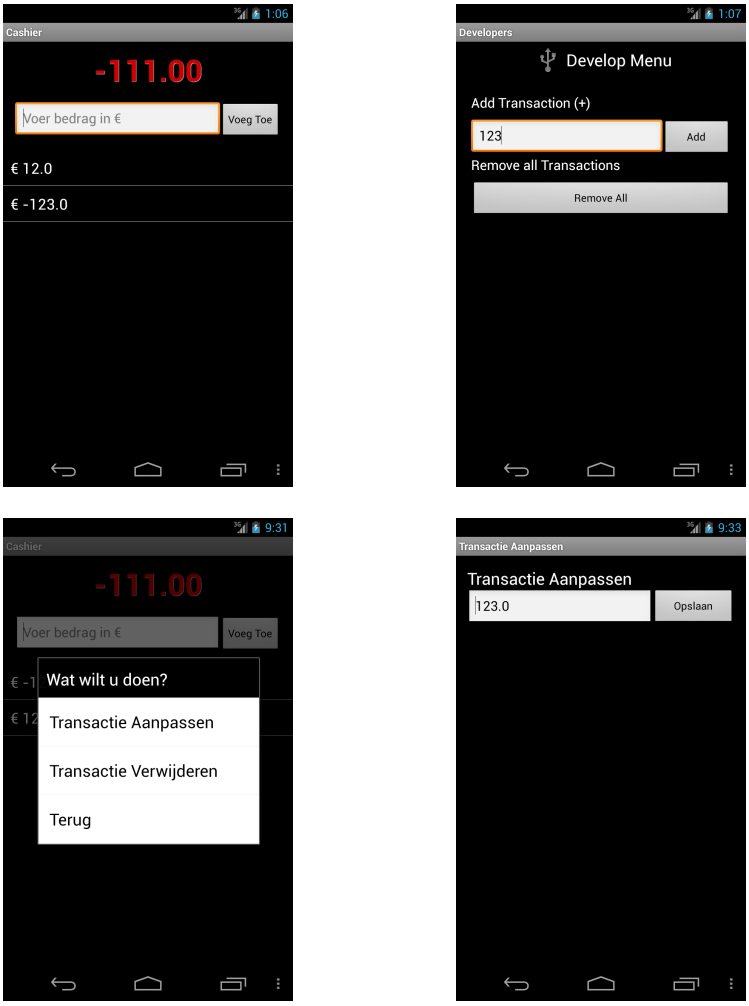
paragraaf 5.1 bevat een klassendiagram van Cashier. De applicatie is opgebouwd uit klassen die behoren tot een van de volgende vier onderdelen:

- Activity
- Controller
- Model
- DataModel

Views in Android bestaan uit XML bestanden en zijn om die reden niet opgenomen in het klassendiagram. De bovenstaande categoriën kunnen als volgt beschreven worden:

Naam	**Betekenis**
Activity	Google beschrijft Activity-klassen als volgt: "An activity represents a single screen with a user interface." Het zijn klassen die voor interactie zorgen met de gebruiker [**?**]. Deze klassen handelen bijvoorbeeld het tonen van het scherm voor ons af en geven veelal functionaliteit aan elementen binnen de UI. Ook kunnen events of gebeurtenissen die binnen de UI worden gegenereerd , afgevangen worden door activities. Denk hierbij bijvoorbeeld aan zogenaamde *actionHandlers*, die functionaliteit aansturen bij gebeurtenissen zoals het klikken, of in dit geval *tappen*, op een knop.
Controller	Ondanks het feit dat Activity-klassen ook controllers zijn, is er besloten een onderscheid te maken tussen de twee. We wilden namelijk liefst de interactie van het systeem met de views, de veelal in de Activity-klassen plaatsvindt, gescheiden houden met de logica die plaatsvindt in de controllers zoals we die kennen van het MVC model.

Jeroen is meteen aan de slag gegaan met het dynamisch genereren van Android interfaces. Het was al snel duidelijk dat er veel invoerschermen nodig gingen zijn voor de app. Om te voorkomen dat elk scherm los in elkaar gezet dient te worden is er besloten een script te maken dat op basis van een lijst simpele tekstbestanden de Android code schrijft om de schermpjes te genereren.



Figuur 1: Screenshots van "Cashier", het *proof-of-concept* van de implementatie van het kasboek in een Android applicatie. De screenshots tonen de CRUD-acties van de applicatie, van links-boven naar rechtsonder respectief: uitlezen (Read), aanmaken (Create), verwijderen (Delete) en aanpassen (Update)

Na de eerste kennismaking met Android is er gekeken naar meer specifieke functionaliteit. Zo is er gekeken hoe gebruik moet worden gemaakt van de GPS in het android toestel [8]. Op basis van een web tutorial is er een simpele klasse geschreven die de GPS coordinaten op kan halen. In een latere fase in het ontwikkelingsproces zou deze klasse meteen gebruikt kunnen worden om de GPS data op een simpele manier op te vragen.

3.2 eInsight Management Portal

De management portal heeft als taak alle data die geregistreerd wordt door de eInsight App, te tonen en bewerkbaar te maken. De applicatie is een web-gebaseerde toepassing en moet vanuit de browser toegankelijk zijn. Adecs Airinfra heeft ervaring met het bouwen van web-toepassingen en gebruikt daarvoor Visual Studio 2010. Er is besloten de applicatie in deze ontwikkelomgeving te bouwen.

3.2.1 MVC

De applicatie zal ontwikkeld worden met behulp van het Model-View-Controller (MVC) design pattern. Het maakt een scheiding tussen het domein, de presentatie en de actie gebaseerd op user-input in drie verschillende soorten klassen:

- **Model**
- **View**
- **Controller**

Het is een veelgebruikt *design pattern* dat een scheiding maakt tussen user-interface logica en business-logica. Microsoft's ASP.NET framework heeft een implementatie van het MVC patroon dat het mogelijk maakt om op een relatief simpele manier complexe web applicaties te ontwikkelen. Dit framework heet MVC3.

Met behulp van de beschikbare informatie in MSDN [7] is een test website opgezet. De test website heeft een simpele, met code first gebouwde database, waarin werknemers (met gebruikersnaam en wachtwoord), activiteiten en transacties kunnen worden opgeslagen. Voor elk van deze onderdelen bestaat één tabel in de database. Deze tabellen kunnen vervolgens op een CRUD achtige manier worden aangepast. Hierbij bestaat een tekstveld voor kolom in de database. Ook is het mogelijk om in te loggen op de website. Met behulp van een sessie wordt bijgehouden wie ingelogd is. Het menu zoals dat in vroege mockups van de web-portal staat is ook zichtbaar in deze website. Het klikken op onderdelen als het kasboek en de overzichten brengt een afbeelding van de bijbehorende mockup in beeld. Deze test website is zo getoond aan Autohulpdienst Broekmans om een idee te geven van hoe de uiteindelijke website er uit zal zien.

3.2.2 Excel libraries

Omdat het aanmaken van Excel spreadsheets een belangrijke eis is aan eInsight is er gezocht naar geschikte libraries om Excel documenten te generen op een server. De methodes die hiervoor gevonden zijn staan hieronder.

- **CSV bestanden**: CSV staat voor Comma Seperated Values. Dit type bestand bestaat simpelweg uit een lijst met waarden gescheiden door komma's. Excel is in staat deze bestanden te importeren en daarbij alle waarden die op een regel staat in een rij in het spreadsheet te stoppen. Elke komma geeft hierbij het begin van een nieuwe kolom aan. Dit leek echter een te primitieve manier.
- **ExcelLibrary** [2]: Een C# library die in staat is om xls bestanden te generen. Dit is het bestandstype dat Microsoft tot aan Excel 2003 handhaafde als standaard. Na wat experimenteren met bleek echter dat het exporteren van de output van deze library als

download op een MVC3 website niet gemakkelijk was. Bovendien was er weinig goede documentatie beschikbaar die kon helpen dit probleem op te lossen. Er is daarom verder gezocht.

- **EPPlus** [1]: Deze C# library is opensource en in staat om xlsx bestanden te exporteren, het meest moderne type Excel spreadsheets. Ook was er goede documentatie beschikbaar met voorbeelden voor het aanmaken van een download op een MVC3 website. De library is makkelijk in gebruik en biedt veel functionaliteit. Er is daarom besloten om deze library te gebruiken voor eInsight.

3.3 Database

Voor dat het ontwerp van de daadwerkelijke database kan beginnen moet eerst een onderzoek worden gedaan naar de te gebruiken tools. Ook moet er worden gekeken naar een goede manier om de database te ontwerpen. Zo zijn er verschillende activiteiten die elk hun eigen velden hebben. Deze dienen toch allemaal opgeslagen te worden.

3.3.1 Entity Framework

Om de Microsoft SQL Server database aan te spreken maken hebben we besloten gebruik te maken van het Entity Framework. Deze library, die ontwikkeld is door Microsoft, maakt het mogelijk om via C# modellen te communiceren met de database. Bas heeft ervaring met dit framework en er is besloten dat hij het grootste deel van de code die hiermee te maken heeft zal schrijven.

De belangrijkste klasse in het Entity Framework (EF) is de ObjectContext klasse. Deze klasse maakt op basis van een meegegeven "connection string"verbinding met de database en kan vervolgens op een eenvoudige manier queries uitvoeren op die database. Verder maakt Entity Framework gebruik van simpele modellen om database objecten te definiëren. Deze modellen zijn niks anders dan klassen die tabellen in de database voorstellen. Elke property in de klasse correspondeert met een tabel in de database. Om gemakkelijk objecten te bereiken waarnaar via een *Foreign Key* verwezen wordt kan nog een property met als modifier *virtual* worden toegevoegd. In figuur 2 staat een voorbeeld van een model voor de Employee en Contract tabel zoals deze voor EF gedefinieerd zouden moeten worden.

Een ERD van de database die bij de in figuur 2 gegeven modellen hoort is te vinden in figuur 3.

Queries aan de database opgesteld worden met behulp van LINQ. LINQ kan genoteerd worden als lambda expressie en dat is dan ook de syntax die wij gebruiken in het project. EF zet de LINQ syntax om in SQL en voert die uit op de database. de gegevens die hierbij uit de database komen worden omgezet tot hun corresponderende C# objecten en teruggegeven. Hieronder staat een voorbeeld van een query om alle employees op te halen die momenteel een contract hebben:

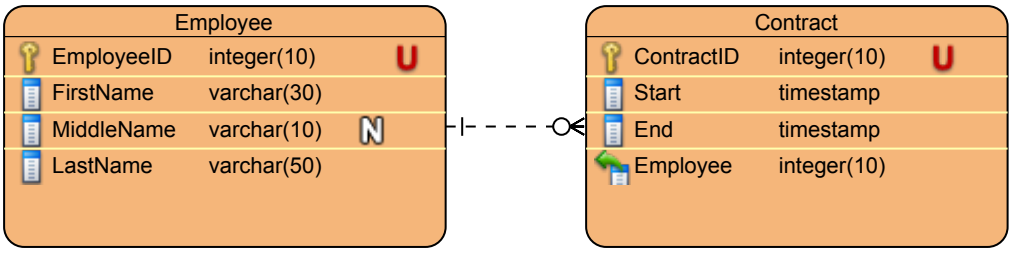
```
public static Employee[] GetEmployees()
{
    using (var db = new SimpleContext())
    {
        var employees = db.Employee
        .Where(e => e.Contracts.Any(c =>
            c.Contract_.Start < DateTime.Now &&
            c.Contract_.End > DateTime.Now)
        .ToArray()
    }
}
```

Het wijzigen of toevoegen van rijen in een tabel via het EF is ook erg simpel. Als een model object namelijk gewijzigd of toegevoegd wordt zorgt EF dit dat ook in de database wordt doorgevoerd met behulp van de methode ObjectContext.SaveChanges(). Het bewerken en toevoegen van een werknemer gaat bijvoorbeeld op de volgende manier:

```
public static SetFirstName(int employeeID, string firstName)
{
    using (var db = new SimpleContext())
```

Figuur 2: C# model voor een simpele database

```
public class Employee
{
    public int EmployeeID { get; set; }

    [Required, MaxLength(30)]
    public string FirstName { get; set; }
    [MaxLength(10)]
    public string MiddleName { get; set; }
    [Required, MaxLength(50)]
    public string LastName { get; set; }

    // Deze virtual property zorgt ervoor dat er gemakkelijk kan worden
        uitgevonden welke contracten allemaal verwijzen naar deze werknemer
    public virtual List<Contract> Contracts { get; set; }
}

public class Contract
{
    public int ContractID { get; set; }
    public DateTime Start { get; set; }
    public DateTime End { get; set; }

    // Deze virtual property geeft het object aan waarnaar de foreign key verwijst
    public virtual Employee Employee_ { get; set; }
    [ForeignKey("Employee_")]
    public int Employee { get; set; }
}

public class SimpleContext : ObjectContext
{
    // Op deze manier wordt aangegeven welke tabellen er bestaan in de database,
        zodat ze kunnen worden aangesproken
    public DbSet<Employee> Employee { get; set; }
    public DbSet<Contract> Contract { get; set; }
}
```



Figuur 3: ERD voor het simpele datamodel gegeven in figuur 2

```
    {
        var employee = db.Employee.First(e => e.EmployeeID == employeeID);
        employee.FirstName = firstName;
        db.SaveChanges()
    }
}

public static AddEmployee(string firstName, string middleName, string lastName)
{
    using (var db = new SimpleContext())
    {
        var employee = new Employee();
        employee.FirstName = firstName;
        employee.MiddleName = middleName;
        employee.LastName = lastName;
        db.Employee.Add(employee);
        db.SaveChanges();
    }
}
```

3.3.2 Code First

Een andere functie van Entity Framework die erg handig is en we zeker gaan gebruiken is het genereren van de database op basis van de modellen. Deze techniek heet *Code First*. Hierbij worden de modellen zoals te zien in figuur 2 gebruikt om een SQL script de genereren dat een database kan aanmaken die te bereiken is met de gegeven modellen.

3.3.3 Inheritance

In het project komen veel soort subactiviteiten voor die maar weinig van elkaar verschillen. Daarom lag het voor de hand om een inheritance structuur te gebruiken. Tijdens het onderzoek hierna zijn een aantal mogelijke manieren om inheritance in een databasestructuur weer te geven naar voren gekomen [6] [5] [4]:

- **Table Per Hierarchy** (TPH): De standaard methode in EF Code First. Alle typen worden in een tabel gestopt. Kolommen die alleen in bepaalde subtypen voorkomen komen als nullable kolom terug in deze tabel. Om te onderscheiden om welk type het gaat wordt een discriminator tabel toegevoegd. Een voordeel hiervan is dat het erg snel is: er zijn geen joins nodig om het juiste subtype te vinden. De nadelen zijn dat het veel loze ruimte kost op de schijf (kolommen die null als waarde hebben) en dat er geen manier is om expliciet in de database aan te geven dat een kolom verplicht is voor dat subtype.

- **Table Per Type** (TPT): Voor elk subtype wordt een nieuwe tabel toegevoegd met daarin de waarden die specifiek zijn voor dat subtype. De *primary key* van de tabel is tevens de foreign key die verwijst naar zijn superklasse. De voordelen hiervan zijn dat je expliciet kan aangeven in het datamodel welke kolommen verplicht zijn. Ook zijn er geen kolommen nodig die vrijwel altijd leeg zijn. Bovendien is het makkelijk om op deze manier een subklasse aan te passen: de tabel voor alle base klassen hoeft daarvoor niet gewijzigd te worden.

- **Table Per Concrete Class** (TPC): Bij deze methode wordt voor elk subtype een nieuwe tabel aangemaakt. Hierin zitten alle kolommen (dus ook die van superklassen van dit subtype). Dit geeft een redelijk goede performance en opslagruimte, maar is lastig te implementeren in het EF.

Na het afwegen van bovengenoemde voor en nadelen hebben we gekozen voor een Table Per Type hiërarchie. Met name omdat deze het mogelijk maakt om verplichte velden in de database te valideren. Bovendien voelt dit aan als de meest nette methode. Testen wees verder uit dat als er voor elk subtype een index werd aangemaakt de performance verschillen tussen TPH en TPT minimaal waren voor een niet al te grote inheritance structuur.

3.4 Tools

Visual Studio

Voor de ontwikkeling van de portal met behulp van ASP.NET en MVC3 en de ontwikkeling van de database is gebruik gemaakt van Visual Studio. Deze ontwikkel omgeving stond reeds op de computers die door Adecs zijn klaargezet voor de projectleden. Er wordt gebruik gemaakt van een project per onderdeel van de applicatie. Zo is er besloten om een project te maken voor de database, voor de functies die nodig zijn voor zowel het synchroniseren als de website, voor de website zelf en voor de synchronisatie service.

Mercurial

Vanuit het bedrijf is beslist dat voor het versie beheer Mercurial zal worden gebruikt. In het begin van het project is er een repository toegewezen met accounts voor alle groepsleden. Mercurial lijkt veel op git. Zo is het bijvoorbeeld ook gedecentraliseerd. Dat wil zeggen dat iedereen lokaal commits kan doen, zodanig dat de andere groepsleden deze nog niet zien. Na een willekeurig aantal commits (of als een volledig functioneel stuk klaar is) kan dan gepusht worden naar de repository zodat de andere projectleden gebruik kunnen maken van de wijziging.

Na een korte oriëntatie op het gebied van mercurial [**?**] is besloten om gebruik te maken van een aantal branches: In de 'default' en 'stable' branches hoort op elk moment in de geschiedenis een werkende versie te staan van wat er tot dan toe af is. Wijzigingen die plaats vinden tijdens de ontwikkeling van nieuwe kleine features wordt de 'dev' branch gebruikt. Voor de 'dev' branch bestaat dus geen enkele garantie dat een willekeurig moment in de historie een stabiele versie is. Voor de wat grotere wijziging kan eventueel nog een nieuwe tijdelijke branch worden aangemaakt, zodat de groepsleden die niet met die feature bezig zijn geen last hebben van half geïmplementeerde wijzigingen.

4 Conclusie

Deze onderzoeksfase heeft veel inzicht verleend in de mogelijkheden van de te gebruiken technieken en tools. In de verdere ontwikkeling zal gebruik worden gemaakt van deze kennis.

Alle project leden zijn nu bekend met Android en de mogelijkheden die dit operating system biedt. Ook weet iedereen nu hoe een Android applicatie is opgebouwd en hoe gebruik kan worden gemaakt van de in geïntegreerde SQLite database.

Voor de management portal is veel geleerd op het gebied van Entity Framework Code First. Er is zelfs al een kleine database opgezet die hier gebruik van maakt. Er is nagedacht over het ontwerp van de uiteindelijke database en welke structuur zal worden gebruikt voor inheritance in de database. Ook is er nu meer duidelijk over hoe ASP.NET en MVC3 gebruikt moeten worden voor het bouwen van een web applicatie. Ten slotte is er uitgedacht hoe een Excel document op op de server kan worden opgebouwd en verzonden aan een gebruiker.

Met deze kennis op zak kan de ontwikkeling van eInsight van start gaan.

Referenties

[1] Epplus. http://epplus.codeplex.com/. Bezocht in mei 2013.

[2] Excellibrary. http://code.google.com/p/excellibrary/. Bezocht in mei 2013.

[3] Getting started on android development. http://developer.android.com/training/index.html. Bezocht in mei 2013.

[4] Table relationships. http://msdn.microsoft.com/en-us/library/s780ea06%28v=vs.71%29.aspx. Bezocht in mei 2013.

[5] Alex James. Tip 12 - how to choose an inheritance strategy. http://blogs.msdn.com/b/alexj/archive/2009/04/15/tip-12-choosing-an-inheritance-strategy.aspx. Bezocht in mei 2013.

[6] kshyju. Entity framework code first inheritance : Table per hierarchy and table per type. http://www.codeproject.com/Articles/393228/Entity-Framework-Code-First-Inheritance-Table-Per. Bezocht in mei 2013.

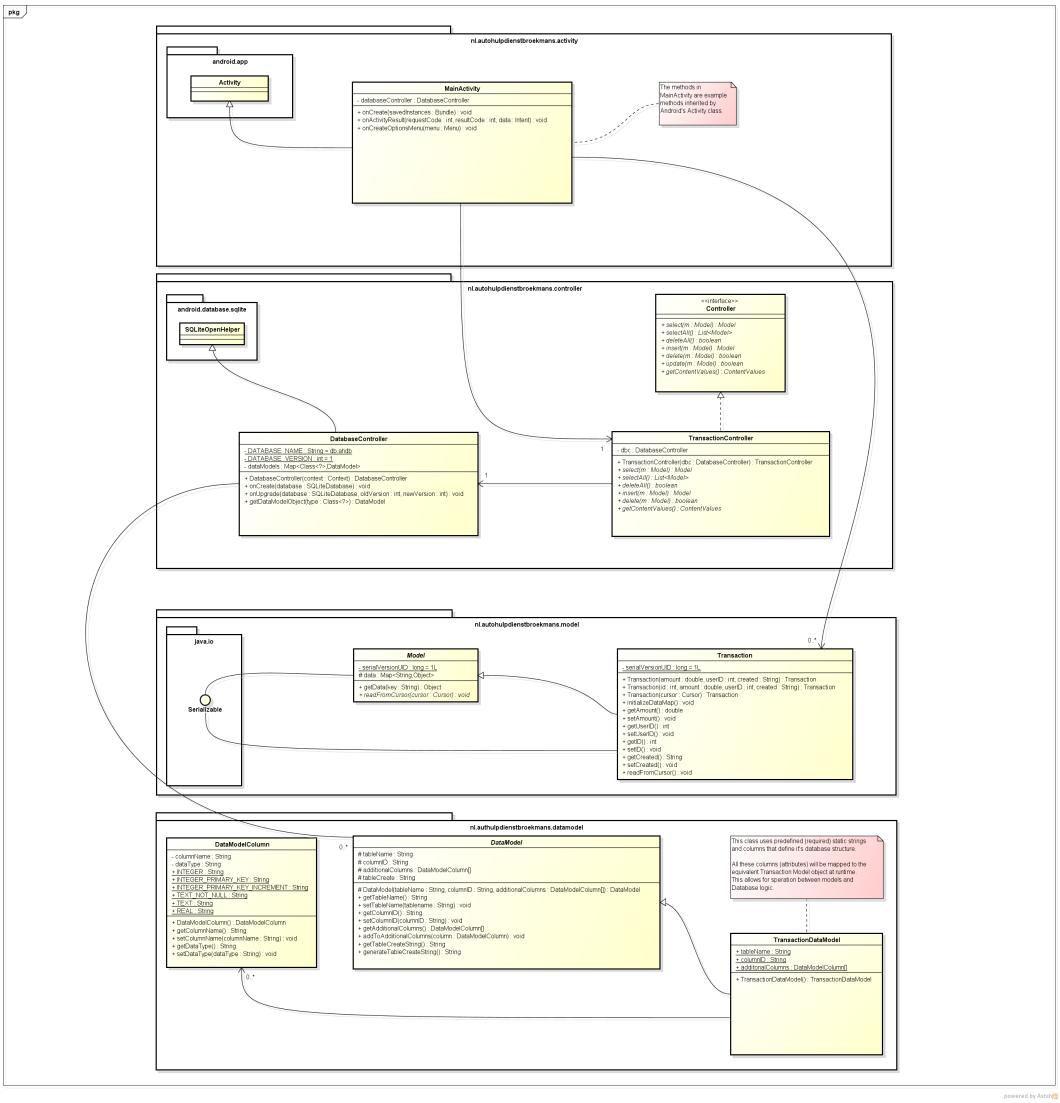
[7] Microsoft. Msdn. http://msdn.microsoft.com/. Bezocht in mei 2013.

[8] Ravi Tamada. Android gps, location manager tutorial. http://www.androidhive.info/2012/07/android-gps-location-manager-tutorial/. Bezocht in juni 2013.

[9] Lars Vogel. Android sqlite database and contentprovider tutorial. http://www.vogella.com/articles/AndroidSQLite/article.html. Bezocht in mei 2013.

5 Appendix

5.1 Klassendiagram Cashier



Figuur 4: Een klassendiagram van het proof-of-concept voor de Android Applicatie.

23 Bijlage P

Plan van Aanpak

Bas Dado, Jeroen Baremans, Jordy van Kuijk

July 4, 2013

Abstract

In dit verslag wordt waaraan de opdracht-gever en de projectmanager moeten voldoen. Er zal beschreven worden hoe de projectplanning ingedeeld zal worden en er worden afspraken vastgelegd wat betreft projectinrichting en kwaliteitsborging.

Contents

Voorwoord

Dit plan van aanpak is geschreven door studenten aan de tudelft voor het bachelor eind project TI3800. Door de projectgroep zal een android applicatie ontwikkeld worden met een web based backend.

1 Introductie

Het plan van aanpak omschrijft een aantal aspecten van de projectopdracht.

1.1 Aanleiding

De stage-begeleider, ir. A. van Helden, werkzaam bij Adecs Airinfra, is via een contact bij de opdrachtgever, Autohulpdienst Broekmans, op de hoogte gekomen dat het bedrijf problemen had met het plannen en administratief beheer van de uit te voeren werkzaamheden. Hierop volgde een gesprek tussen Adecs Airinfra en Autohulpdienst Broekmans, waarin onderzocht werd op welke manier de onderneming verbeteringen kon doorvoeren op deze gebieden. Omdat het huidige software pakket waarmee gewerkt wordt om de werkzaamheden in te plannen een update zou ontvangen die veel problemen zou verhelpen, is hiervoor geen verdere ontwikkeling nodig. De administratieve problemen die het bedrijf heeft met de activiteitenregistratie is de aanleiding voor de ontwikkeling van het software-pakket *eInsight* voor de opdrachtgever.

1.2 Accordering en bijstelling

Nadat het plan van aanpak opgesteld is, zal dit worden voorgelegd aan de begeleider van de TU Delft, dr. ir. A. J. H. Hidders, en de stage-begeleider ir. A. van Helzen. In week 2 en 4 van het project zal overleg plaatsvinden tussen de projectgroep, ir.A. van Helzen en Autohulpdienst Broekmans. In dit overleg zal de projectgroep de huidige productversie toelichten en vragen naar feedback. Deze informatie zal verwerkt worden in de iteraties tijdens de ontwerp- en implementatiefase. Na 2 bijeenkomsten, met indien nodig nog een derde bijeenkomst, zal het ontwerp van de software definief worden en zal deze zonder verdere wijziginen uitgewerkt worden tot de applicatie.

1.3 Toelichting op de bouw van het plan

De indeling van het document is gebaseerd op het plan van aanpak dat beschikbaar is gesteld binnen de *backboard*-omgeving van de TU Delft. Sectie 2 zal ingaan op de projectopdracht en hierbij een analyse maken van onder andere de projectomgeving, de doelstelling van het project en de opdrachtformulering. Het hoofdstuk wordt daarna afgerond met een vaststelling van de op te leveren producten en diensten, de eisen en beperkingen en de voorwaarden waaraan de opdrachtgever en de projectleden moeten voldoen om de gestelde eisen waar te maken. Hoofdstuk 3 zal bespreken welke aanpak gehanteerd wordt en welke tijdsplanning aan wordt gehouden. Ten slotte Wordt in Hoofdstuk 4 de projectinrichting uitgelegd.

2 Projectopdracht

2.1 Projectomgeving

Autohulpdienst Broekmans, gevestigd op de Colombusweg 11 te Venlo, is een klein transport- en bergingsbedrijf waar ongeveer dertig mensen werkzaam zijn. Zij hebben veertien vrachtwagens in dienst die 24/7 actief zijn door Europa. Het bedrijf maakt momenteel gebruik van zogenaamde ritstaten (zie bijlage 1), waarin bijgehouden wordt welke activiteiten er plaatsvinden en door wie. In de huidige situatie worden hiervoor papieren formulieren gebruikt. Het komt vaak voor dat er fouten in deze papieren documenten sluipen of dat deze bijvoorbeeld achteraf pas ingevuld worden. Ook is het verwerken van de activiteiten, dat momenteel gebeurt door de werkbriefjes over te typen in Microsoft Excel, erg omslachtig en tijdrovend.

2.2 Doelstelling project

De opdrachtgever wil boventaande problemen aanpakken, waardoor het bedrijf efficiënter kan functioneren en dus kosten bespaart kunnen worden. Deze verbeterde efficiëntie moet worden bereikt dankzij een door ons te ontwikkelen systeem welke een aantal van de bovenstaande stappen zal elimineren. Zo hoeven kantoormedewerkers bijvoorbeeld de data op de papieren formulieren met behulp van ons systeem niet langer over te zetten naar Microsoft Excel. De activiteiten registratie door vrachtwagen-chauffeurs wordt vergemakkelijkt omdat pen en papier worden vervangen door

een paar aanrakingen vanp het scherm een activiteit te registreren, de app zal de chauffeurs assisteren door bekende velden zoals tijd, datum en nummerbord automatisch in te vullen. Door toepassing van locatie-technologie in ons product, zoals GPS-positionering, kan de klant meer controle uitoefenen op haar werknemers door wanneer gewenst de locatie van de chauffeurs te controleren. In de applicatie zal functionaliteid ingebouwd worden om de bedrijfsportommonnee van de chauffeurs bij te houden.

2.3 Opdrachtformulering

De opdracht is het ontwerpen, ontwikkelen en afleveren van zowel een Android-applicatie als een management-portal die bovenstaande doelstellingen waarmaakt (samen *eInsight* genaamd). De Android-applicatie zal gebruikers toelaten snel, efficient en op een gebruiksvriendelijke manier activiteiten te registreren en het digitaal kasboek te benaderen. De opgeslagen data zal worden overgezet naar een centrale database/server, die vervolgens op verscheidene manieren kan worden gepresenteerd binnen de management-portal. Ook kan binnen de management-portal het digitaal kasboek per werknemer beheerd worden en kunnen ziekteverzuim en verlofuren van werknemers bijgehouden worden. De management-portal zal in de vorm van een web-toepassing beschikbaar zijn voor medewerkers van de opdrachtgever en zal een aantal bestaande, handmatig ingevulde, Microsoft Excel spreadsheets kunnen vervangen. De gebruikers van de portal zullen ook de mogelijkheid hebben data te exporteren naar een Microsoft Excel-compatibel bestand. De opdrachtgever verwacht dat wij tijdens het ontwerp- en ontwikkelproces een beta- en twee opvolgende testversies (van zowel de Android-applicatie als het management-portal) afleveren alvorens de definitieve aflevering van het product. Dit houdt onder andere in dat wij de opdrachtgever een of meerdere bezoeken zullen brengen ter demonstratie van het product.

2.4 Op te leveren producten en diensten

Het resultaat van het project bestaat uit een softwarepakket, genaamd *eInsight*. Dit softwarepakket bestaat uit twee applicaties, namelijk een Android-applicatie en een web-gebaseerde management-portal. De Android applicatie zal geïnstalleerd worden op *Motorola Defy Plus* toestellen (zie bijlage 2), die door de opdrachtgever voor dit project zijn aangeschaft. De management-portal zal in de browser bekeken kunnen worden, bereikbaar zijn via internet en gehost worden op een Windows Server. De android applicatie moet in staat zijn om de activeiten van werknemers te registreren. Ook moet hierop kunnen worden ingevuld welke kosten er zijn gemaakt names het bedrijf om het kasboek te kunnen bijhouden. De management-portal moet in staat zijn om de door de app verzamelde gegevens te kunnen inzien. Ook moet het mogelijk zijn om ziekteverzuim en verlofuren van werknemers te registreren.

2.5 Eisen en beperkingen

Tijdens de ontwikkeling van de applicatie moet de opdrachtgever de mogelijkheid hebben om eventuele verbetering door te voeren. Hiervoor is het vereist dat er minimaal twee maal een test versie van de applicatie aan de opdrachtgever moet worden getoond waarna hierop feedback kan worden gegeven. De applicaties moeten verder aan de volgende eisen voldoen:

Mobiele Android-applicatie:

- Android 2.3 support
- Registratie van (verschillende) activiteiten uitgevoerd door het personeel van de opdrachtgever, die in de huidige bedrijfsvoering worden verzameld door middel van rittenstaten
- Registratie van werktijden die in de huidige bedrijfsvoering worden verzameld door middel van weekstaten
- Transactiebeheer digitaal kasboek (bijstortingen, uitgaves)

- Bufferen van verzamelde gegevens tot de gebruiker weer op het kantoor is

Management-portal:

- De database en portal moeten kunnen draaien op een door de opdrachtgever aangeschafte windows-server
- Export van de verzamelde informatie naar Microsoft Excel voor analyse
- Bijhouden van digitaal kasboek
- Berekeningen van de door het personeel gerealiseerde uren, toegekende toeslagen, overuren
- Registratie vam ziekteverzuim en verlofuren van werknemers

2.6 Voorwaarden

3 Aanpak en tijdsplanning

Het project zal worden onderverdeeld in de volgende fases, waaraan in chronologische volgorde zal worden gewerkt:

- Voorontwerp Tijdens het voorontwerp wordt er vooral gekeken naar de in de voorgaande secties besproken onderwerpen. Het doel is het identificeren van het probleem, brainstormen over een mogelijke oplossing voor het probleem en een technologisch onderzoek. Hierbij hoort ook het opstellen van het plan van aanpak en het oriëntatie onderzoek. Aan het einde van deze fase zal er bezoek worden gebracht aan de klant. Tijdens dit bezoek zullen mock-ups/prototypes van de applicatie worden getoond zodat duidelijk is hoe wij het probleem en de oplossing begrijpen. De klant kan hierop feedback geven die wij in de volgende fase zullen verwerken.
- Ontwerp Tijdens de ontwerp fase zullen er op basis van de feedback uit de vorige fase, indien nodig, veranderingen aangebracht worden aan onze eisen of functionaliteiten. Hierna worden alle basiscomponenten van de Android-applicatie en de web-gebaseerde management-portal geïmplementeerd. Uiteindelijk moet er aan het einde van de fase een BETA-versie van het software pakket klaar zijn, die door de klant kan worden getest. Ook hierop kan weer commentaar geleverd worden dat we in de bouw fase kunnen gebruiken.
- Bouw In de bouw fase zullen alle eisen worden geïmplementeerd en zal feedback uit de vorige fase worden bewerkt zodat aan het eind van deze fase versie 1.0 van het product opgeleverd kan worden. De mobiele applicatie, de database en de web-gebaseerde management tool moeten aan elkaar gekoppeld worden. Ook moet er voldaan worden aan verscheidene test-criteria. De fase neemt drie weken in beslag. Aan het eind van de eerste twee weken al er een testversie beschikbaar zijn voor de werknemers van Autohulpdienst Broekmans. Zij zullen ons dan voorzien van test-data door de toestellen dan al in gebruik te nemen. Er wordt in deze fase nog een bezoek gepland aan de klant, niet alleen om feedback te krijgen, maar ook om ons voor te bereiden op de oplevering van het product.
- Ondersteuning Deze laatste fase zal vooral dienen om onze klant te ondersteunen bij het gebruik van onze producten. Ook worden er in deze fase nog bug-fixes doorgevoerd en is er tijd ingepland voor het schrijven van de stage-rapporten, het eindverslag en het maken van de presentatie van het project. De fase neemt vier weken in beslag.

Een schema met daarin de onderverdeling van de verschillende fasen kan gevonden worden in bijlage 2.

Table 1: Schatting benodigde resources en tijd

#	Geschatte Tijd	Fase	Omschrijving
1	36	Voorontwerp	Plan van Aanpak
2	45	Voorontwerp	Oriëntatie-verslag
3	12	Voorontwerp	Opstellen decision-tree mobiele applicatie
4	54	Voorontwerp	Kennismaking met Android
5	54	Voorontwerp	Kennismaking met C# en ASP.NET
6	36	Voorontwerp	Functional-requirements plan opstellen
7	??	Ontwerp	Database ontwerpen
8	??	Ontwerp	Klassendiagram opstellen
9	??	Ontwerp	Use Cases & Sequence diagrams opstellen
10	??	Ontwerp	Test and implementation plan schrijven
11	??	Ontwerp	Testen BETA-versie
12	??	Bouw	Implementatie Android App
13	??	Bouw	Implementatie Management-portal
14	??	Bouw	Implementatie Android App
15	??	Bouw	Implementatie Management-portal v0.1
16	??	Bouw	Implementatie Management-portal v0.2
17	??	Bouw	Implementatie Management-portal
18	??	Bouw	Implementatie Android App v0.1
19	??	Bouw	Implementatie Android App v0.2
20	??	Ondersteuning	Bug-fixing
21	??	Ondersteuning	Eindverslag
22	??	Ondersteuning	Presentatie

4 Projectinrichting

4.1 Organisatie

De projectgroep bestaat uit drie leden, die gezamenlijk de verantwoordelijkheid hebben een product af te leveren dat voldoet aan de eerder gestelde eisen en wensen van de klant. Binnen de groep draagt elk persoon dezelfde verantwoordelijkheid. Ieder zal zijn bedrage moeten leveren aan elk onderdeel. Het is niet zo dat er specifiek een persoon alleen aan bijvoorbeeld de mobiele applicatie zal werken. Om te voorkomen dat er door deze organisatie dubbel werk wordt gedaan wordt er van tevoren bepaald welke taken er precies gedaan moeten worden. Wanneer een van de projectleden begint met werken aan zo'n taak wordt er verwacht dat hij dit doorgeeft aan de andere leden, zodat iedereen in d eprojectgroep ten alle tijden op de hoogte is van waaraan iedereen op dat moment werkt.

4.2 Personeel

Binnen het personeel is naast onze stage-begeleider altijd een IT-specialist bereikbaar, mochten we vragen hebben of tegen problemen aanlopen. De stage-begeleider is van werkplek verplaatst, zodat hij als het ware "een oogje in het zeil" kan houden. Verder wordt er van de klant verwacht dat hij is staat is om feedback te geven op de daarvoor vastgestelde momenten.

4.3 Administratieve procedures

Er zal elke dag aan het begin van de dag een korte vergadering worden gehouden tussen de teamleden en de stage-begeleider. Hierin wordt kort doorgenomen hoe ver we de dag ervoor zijn opgeschoten en wat er voor de rest van de dag op de planning staat. Verder worden er bij aanvang

van een nieuwe fase een aantal mile-stones afgesproken.

4.4 Rapportering

De groep zal minimaal twee en maximaal drie keer een bezoek brengen aan de klant, te Venlo, ter demonstratie van het product. Ook kan er tijdens deze bezoeken feedback verkregen worden van niet alleen de opdrachtgever, maar ook zijn werknemers, die van de software gebruik zullen gaan maken. Verdere communicatie zal verlopen via e-mail en telefoon.

4.5 Resources

De werkplekken binnen Adecs Airinfra bestaan uit gezamenlijk bureaus. De drie projectgenoten en de stage-mentor zitten allemaal aan hetzelfde bureau en hebben elk een pc ter beschikking. Op de computers wordt er gebruik gemaakt van de volgende resources:

- Microsoft Windows XP, Windows Vista of Windows 7
- Microsoft Exchange server voor interne- en externe e-mail, contactenbeheer en kalenderfuncties
- Eclipse voor ontwikkeling van de Android-applicatie
- Microsoft Visual Studio voor het ontwikkelen van de web-based management-portal
- Microsoft SQL Server voor het beheren van de database waarin de data afkomstig uit de mobiele applicatie en de manager-portal wordt opgeslagen

5 Kwaliteitsborging

Zoals eerder genoemd zijn er gaat de projectgroep minimaal twee keer, de opdrachtgever bezoeken om het product te tonen. De klant heeft dan de mogelijkheid feedback te geven op de software, die de projectgenoten vervolgens kunnen gebruiken om de kwaliteit te verhogen. Na de release van de BETA-versie van de applicatie gaan de werknemers van de opdrachtgever met de applicatie aan de slag en kunnen de projectgenoten de data die opgeslagen wordt in de database gebruiken en inspecteren om het product te verbeteren.

Bijlagen

Bijlage 1: internationale rittenstaat

Bijlage 2: Fase-indelingsschema

Voorontwerp	22-4-2013	Maandag	
	23-4-2013	Dinsdag	
	24-4-2013	Woensdag	
	25-4-2013	Donderdag	
	26-4-2013	Vrijdag	
	29-4-2013	Maandag	
	30-4-2013	Dinsdag	
	1-5-2013	Woensdag	
	2-5-2013	Donderdag	
	3-5-2013	Vrijdag	Bespreken voorontwerp, in ontvangst nemen telefoon
Ontwerp	6-5-2013	Maandag	
	7-5-2013	Dinsdag	
	8-5-2013	Woensdag	
	9-5-2013	Donderdag	
	10-5-2013	Vrijdag	
	13-5-2013	Maandag	
	14-5-2013	Dinsdag	
	15-5-2013	Woensdag	
	16-5-2013	Donderdag	Opleveren BETA-versie
	17-5-2013	Vrijdag	
Bouw	20-5-2013	Maandag	
	21-5-2013	Dinsdag	
	22-5-2013	Woensdag	
	23-5-2013	Donderdag	Testversie 1
	24-5-2013	Vrijdag	
	27-5-2013	Maandag	
	28-5-2013	Dinsdag	
	29-5-2013	Woensdag	
	30-5-2013	Donderdag	Testversie 2
	31-5-2013	Vrijdag	
	3-6-2013	Maandag	
	4-6-2013	Dinsdag	
	5-6-2013	Woensdag	
	6-6-2013	Donderdag	
	7-6-2013	Vrijdag	Opleveren E-insight
Ondersteuning	10-6-2013	Maandag	
	11-6-2013	Dinsdag	
	12-6-2013	Woensdag	
	13-6-2013	Donderdag	
	14-6-2013	Vrijdag	
	17-6-2013	Maandag	
	18-6-2013	Dinsdag	
	19-6-2013	Woensdag	
	20-6-2013	Donderdag	
	21-6-2013	Vrijdag	
	24-6-2013	Maandag	
	25-6-2013	Dinsdag	
	26-6-2013	Woensdag	
	27-6-2013	Donderdag	
	28-6-2013	Vrijdag	
	1-7-2013	Maandag	
	2-7-2013	Dinsdag	
	3-7-2013	Woensdag	
	4-7-2013	Donderdag	
	5-7-2013	Vrijdag	Afsluiting project