AFSTUDEERVERSLAG

Mobiele applicatie voor evenementensysteem

Bram Slob
11050365
Informatica deeltijd

Revisiehistorie

Versie	Revisiedatum	Wijzigingen
0.1	27-03-2014	Initiële versie
1.0	29-04-2014	Eerste oplevering
2.0	14-05-2014	Tweede oplevering
3.0	05-06-2014	Definitieve oplevering

Distribution

Dit document is bedoeld voor:

Naam	Functie	Datum van uitgave	Versie
Gerard Mijnarends	Begeleidend examinator	29-04-2014	1.0
Gerard Mijnarends	Begeleidend examinator	14-05-2014	2.0
Ben Kuiper	Tweede examinator	14-05-2014	2.0
Gerard Mijnarends	Begeleidend examinator	05-06-2014	3.0
Ben Kuiper	Tweede examinator	05-06-2014	3.0

Inhoudsopgave

1. Inleiding

Het document dat u voor u hebt liggen beschrijft het afstudeertraject van Bram Slob op de mobiele Iventer applicatie. Tijdens de afstudeerperiode heb ik mij bezig gehouden met het bouwen van deze applicatie, welke als voornaamste doel heeft om tijdens een evenement de presentie van vooraf bekende reserveringen op te nemen en om leads te werven tijdens dat evenement.

In het tweede hoofdstuk vindt u een uiteenzetting van de opdracht, waar deze vandaan komt en welke significantie deze heeft in het bedrijf. Ook zal er gekeken worden naar enkele eisen waaraan de applicatie moet voldoen.

In het derde hoofdstuk kunt u vinden hoe dit project organisatorisch opgezet is. Er worden verschillende afspraken besproken als ook de systeemontwikkelmethode en hoe deze in het project is toegepast.

In het derde hoofdstuk kunt u een stuk vinden over welke requirements er aan dit project zijn gesteld. Wat hoort er wel bij en wat niet. Daarnaast zal er ingegaan worden op verschillende aannames en concepten die voor en tijdens het doorlopen van het project tot stand zijn gekomen.

In het vijfde hoofdstuk vindt u een beschrijving van het proces waarin de applicatie tot stand is gekomen. Het hoofdstuk biedt inzage in het tot stand komen van de applicatie en is opgedeeld in de verschillende sprints.

In het zesde hoofdstuk komen de gebruikte technieken, zoals JavaScript libraries, naar voren. Deze technieken zullen kort worden toegelicht en de toegevoegde waarde komt hier ook naar voren. Ook wordt in dit hoofdstuk gesproken over de testtechnieken.

In het zevende hoofdstuk zullen verschillende uitdagingen van de opdracht worden om- en beschreven. De oplossingen voor deze uitdagingen worden later in dat hoofdstuk beschreven.

In het achtste hoofdstuk vindt u verscheidene bijlagen. Hier is onder andere een lijst te vinden met gebruikte termen en een verklarend woordenlijst. Daarnaast zijn hier afbeeldingen en andere bijlagen te vinden die het verslag verder ondersteunen.

2. De opdracht

De opdracht is gestart vanwege de vraag vanuit de werkgever voor een mobiele applicatie voor het evenementen systeem dat in de maak is. In dit hoofdstuk wordt de opdracht en de context en geschiedenis ervan beschreven.

2.1 Het bedrijf

Faceworks is een bedrijf dat zich richt op het leveren van (online) communicatiemiddelen in de vorm van websites, webshops, webapplicaties en (mobiele) apps. De klanten variëren in grootte, met als grootste klanten BMW, MINI en de Rabobank en aan de andere kant kleinere klanten zoals verschillende golfclubs verspreid door het land en enkele ZZP'ers. Het bedrijf bestaat uit 10 man, een commerciële man, een ontwerper en acht developers. De taken zijn verdeeld onder de werknemers op basis van kennis en ervaring. Er wordt hoofdzakelijk in PHP geprogrammeerd, waarbij technieken als HTML, CSS, Javascript en SQL natuurlijk ook essentieel zijn. Momenteel is het bedrijf bezig om op verschillende vlakken te innoveren, zowel door het ontwikkelen van het Iventer evenementen systeem als door een intuïtief CMS systeem door middel van drag-and-drop technieken.

Vanuit het bedrijf is mij de mogelijkheid gegeven om drie werkdagen in de week volledig bezig te zijn met dit project. Dit zorgt voor de ruimte en mogelijkheid om de opdracht goede richting en snelheid te geven. Er zijn weken dat er zelfs vijf werkdagen beschikbaar zijn, andere weken komen er verschillende projecten tussendoor waardoor de beschikbare tijd iets lager uitkomt. Maar over het algemeen krijg ik voldoende tijd om de taken te vervullen.

2.2 Probleem

Met enige regelmaat organiseren (voornamelijk grote) klanten evenementen of doen zij daar aan mee. Hierbij moet gedacht worden aan een beurs, ledendag, golfdag, etc. Tijdens deze evenementen is het wenselijk om te weten hoeveel van de uitgenodigde ofwel aangemelde reserveringen feitelijk gekomen zijn. Daarnaast kan een klant tijdens het evenement leads willen werven. Onder leads kunnen aanmeldingen voor een nieuwsbrief of brochure worden verstaan, ofwel nieuwe zakelijke mogelijkheden.

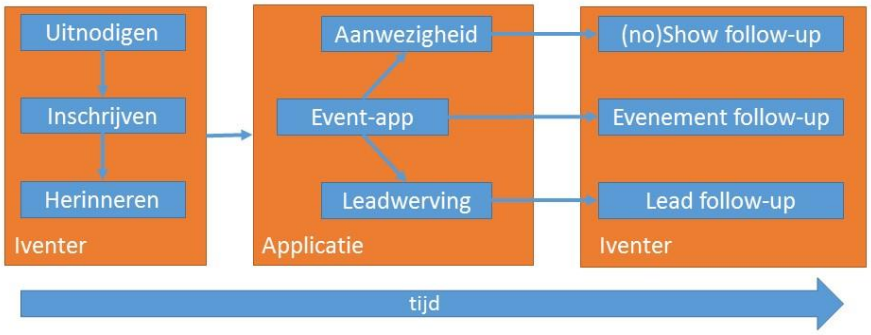
Tot nu heeft Faceworks per evenement een applicatie gemaakt die toegespitst was op het evenement. Dit is een tijdintensief proces dat veel specifieke code vergt. Aangezien alle applicaties uniek zijn, moeten ze allen afzonderlijk onderhouden worden en kunnen er fouten in de code sluipen. Hierdoor is de wens ontstaan om een applicatie te ontwikkelen dat als basis gaat dienen voor de toekomstige applicaties.

2.3 Oplossing

De oplossing is het ontwikkelen van een applicatie, die als schil moet dienen. Binnen deze applicatie moeten de verschillende evenementen ingeladen en gedraaid kunnen worden. Dit heeft een aantal duidelijke voordelen. Zo wordt de wordt het uitrollen van een nieuwe event-app minder foutgevoelig, omdat gebruik gemaakt wordt van dezelfde code. Het zorgt ook voor een snellere opleversnelheid, omdat delen van de applicatie, zoals de admin en het vinden/bekijken van reserveringen, voor de meeste al dan niet alle evenementen hetzelfde zijn. Wanneer gebruik gemaakt wordt van een standaard applicatie is het een stuk eenvoudiger om statistieken bij te houden, omdat er uitgegaan kan worden van gemeenschappelijke metrieken, zoals het aantal verzamelde leads en het aangemelde aantal reserveringen.

2.4 Iventer

Tijdens het organiseren van de verschillende evenementen heeft Faceworks gemerkt dat er veel verschillende partijen gemoeid zijn bij het organiseren van een evenement. Zo kan er een partij zijn die de registratie voor een evenement regelt, een partij die de mailings regelt, een partij die relatiegegevens beheert en/of distribueert en een partij die devices levert die op het evenement gebruikt worden. Dit zorgt voor veel verschillende communicatielagen voor de klant wat voor vertragingen kan zorgen in het organiseren van het digitale deel van een evenement, zijnde het inschrijven en registreren van presentie en het werven van leads. Om hier een oplossing voor te bieden is Faceworks begonnen aan het ontwikkelen van Iventer. Hieronder een schematische weergave van de stappen die Iventer beheert.



2.1 Processflow evenement binnen Iventer

Iventer moet een systeem worden dat de hele flow van een evenement kan beheren. Het beheert het uitnodigen van de (geïmporteerde) relaties en het inschrijven voor het evenement en zo een reservering te plaatsen ervoor. Het is eventueel ook mogelijk om verschillende kaarten te reserveren, voor de relatie (als hoofdboeker) en verschillende introducés. Na een bepaalde tijd kan er een herinnerings-e-mail en/of -SMS worden gestuurd. Vervolgens wordt de applicatie op het evenement gebruikt om de aanwezigheid van de reserveringen bij te houden en om leads te werven. Na het evenement kan er op basis van de verworven gegevens follow-up worden gedaan.

Zo heeft de klant invloed op alle vlakken van een evenement wat de klant meer inzage geeft in het proces en de tools om zelf wijzigingen aan te brengen. Dit bespaart zowel Faceworks als de klant tijd en geld.

Een goed voorbeeld van een situatie waarin Iventer uitkomst biedt is het organiseren van een evenement voor BMW. Op dit moment zitten er verschillende partijen tussen BMW en een evenement in. Zo is er Faceworks, maar ook Events voor het leveren van de apparatuur voor een evenement, iProspect voor het leveren van de CRM gegevens om relaties uit te nodigen en binnen BMW is er ook nog rekening te houden met CRM systeem Carmen. Al deze partijen en software zorgen bij het plannen en uitvoeren van een evenement een lang en prijzig proces omdat zoveel verschillende partijen hun input en output moeten leveren.

Iventer is bedoeld om alle digitale communicatie uit handen te nemen en waar nodig te vervangen. Zo heeft de klant minder leveranciers om rekening mee te houden. Iventer is een middleware systeem dat wordt aangesproken door middel van REST API's. Waardoor het mogelijk is om verschillende front- en backenden te schrijven voor het systeem, zolang ze maar dezelfde API's aanspreken.

Het uiteindelijke product van dit project is het applicatie gedeelte in figuur 2.1.

2.5 Beschrijving

De opdracht die is uitgevoerd had als doel om een mobiele applicatie te maken, waarmee op een evenement presentie kan worden bijgehouden en leads kunnen worden verworven om deze vervolgens naar het Iventer systeem te sturen. Er zijn vanuit Faceworks verschillende eisen gesteld aan de werking en opbouw van de applicatie.

Een van de belangrijkste zal zijn dat de applicatie standaard offline werkt. Er mag niet uitgegaan worden van een connectie met het internet en dus met de API of externe database. Daarnaast moet de applicatie eenvoudig te hergebruiken zijn voor een andere klant, doormiddel van andere stijling en formuliervragen. Wat daar mee samen hangt is dat de opzet van de applicatie erg generiek moet zijn, hoe meer er dynamisch te regelen is, hoe beter.

2.6 De oude situatie

Gezien de oude code voor het maken van een applicatie zijn er geldende verwachtingen betreffende de functionaliteit en werking van de applicatie. Het ontwerp van de oude code is meegenomen in de ontwikkeling van de applicatie. Hierna volgt een beschrijving van de functionaliteiten die verwacht worden en hoe deze zich tot elkaar verhouden.

Om de applicatie binnen te komen moet de gebruiker een inlogcode geven. Na het ingeven van deze code komt de gebruiker op een startscherm waar er genavigeerd kan worden naar het scannen van een reservering of het werven van een lead. Deze navigatie is op elke pagina beschikbaar, zodat er eenvoudig tussen de verschillende functionaliteiten gewisseld kan worden.

Een gebruiker moet de gegevens die zijn verzameld via de applicatie ook kunnen versturen naar Iventer. Dit moet gebeuren via de admin. Er moet een overzicht zijn van de verschillende leads en reserveringen die verstuurd kunnen worden. Daarnaast moet een lead ook verwijderd kunnen worden.

3. Projectorganisatie

In het vorige hoofdstuk is beschreven wát er gemaakt moet worden. In dit hoofdstuk wordt beschreven hoe er naar dit doel toe is gestreefd. Er komt vooral aan bod hoe het blok georganiseerd is in termen van afspraken, wijze van organiseren, hoe er ontworpen is en hoe er gedocumenteerd is.

3.1 Plan van aanpak

Voordat er aan het project is begonnen is er een plan van aanpak opgesteld. In dit plan van aanpak zijn verschillende afspraken gemaakt betreffende het aanpakken van het project en het opleveren van mijlpaalproducten. Hier volgt een evaluatie van de uitvoer van de afspraken en het maken van de mijlpaalproducten.

Afbakening opdracht

Het was de bedoeling dat het project een mix zou zijn tussen front- en backend werk, wat inhoudt dat er zowel werk verzet zou moeten worden aan de code voor de applicatie als aan de API aan de kant van Iventer. Dit is tijdens de ontwikkeling accuraat gebleken. De meeste onderdelen van de API waren al beschikbaar, maar bijvoorbeeld het door kunnen sturen van instellingen en assets zijn tijdens het ontwikkelen van de front-end naar voren gekomen.

Daarnaast is er tijdens het ontwikkelen alleen aandacht geschonken aan het werkend krijgen van de applicatie op een PC en tablet. Het visuele en functionele ontwerp van de applicatie is hier ook op toegespitst. Er is bijvoorbeeld niet ontwikkeld voor en getest op een mobiele telefoon, omdat deze situatie niet van toepassing is.

Randvoorwaarden

Als een van de randvoorwaarden van het project, is gesteld dat er genoeg testmogelijkheden beschikbaar moesten. Hier is tijdens het doorlopen van het project aan voldaan, aangezien er een iPad beschikbaar was om op te testen. Daarnaast is er tijdens het project ook een Android tablet aangeschaft door het bedrijf, waardoor testen op het Android platform ook een mogelijkheid werd.

Daarnaast is er ook voldaan aan de tweede randvoorwaarde, duidelijke en tijdige feedback. Omdat de opdrachtgever in hetzelfde pand aanwezig is, is het erg eenvoudig om vragen aan hem te stellen en zo feedback op de genomen richting te krijgen.

Risicofactoren

De verschillende risicofactoren die beschreven zijn, zijn voor het grootste gedeelte ontweken. Zo is er niet uitgelopen. Daarnaast is er voldaan aan de verschillende requirements die er gesteld zijn, zoals te zien is aan de resultaten van de acceptatietest.

Er is echter wel werk overnieuw gedaan. Vooral de keuzes rondom de opbouw van het framework hebben voor veel werk gezorgd dat opnieuw gedaan moest worden. Zo werd eerst uitgegaan van de oude code voor het maken van een applicatie. Vervolgens is de keuze gemaakt om dit opnieuw te schrijven, maar wel vast te houden aan de oude structuur. En uiteindelijk is er voor gekozen om de applicatie opnieuw op te zetten in AngularJS.

Wijze van rapporteren

Het rapporteren van een sprint is uiteindelijk niet formeel gebeurd. Het evalueren van een sprint is opgenomen in de sprintplanning van de volgende sprint. De reden hiervoor is dat er geen dubbele evaluatie geschreven hoeft te worden. Daarnaast is een van belangrijke punten om te rapporteren, de gedraaide tests. Echter er zijn in de eerste paar sprints geen tests gedraaid, wat het rapporteren minder nuttig maakt.

Planning

De globale planning van een sprint is voor een groot gedeelte niet gevolgd. Hier zijn een aantal redenen voor, de grootste is nog wel dat er vrijwel geen enkele sprint 12 dagen beschikbaar waren om te werken aan het project. Daarnaast is er slechts 1 dagdeel per sprint besteed aan planning.

Er is tijdens de eerste paar sprints geen aandacht besteed aan formeel testen. Er is in die sprints niet op een gestructureerde wijze getest. Natuurlijk is er wel getest of de geproduceerde code werkte, maar dit is slechts gedaan door error guessing om het probleem te achterhalen.

Zoals al bij 'wijze van rapportage' is genoemd, is er tussendoor geen rapportage opgemaakt, omdat er buiten het testen om geen rapportage nodig leek.

Mijlpaalproducten

Er is grotendeels gehouden aan de opzet van de mijlpaalproducten. Zo is er een plan van aanpak gemaakt en een user stories document.

Daarnaast is er per sprint een sprintplanning gemaakt en is de code in die sprint gewijzigd. Er is echter geen testrapport per sprint gemaakt.

Er is echter wel een testrapport gemaakt in de laatste sprint, voor de acceptatietest.

3.2 Afspraken

Voordat het project is gestart, zijn er met het bedrijf verschillende afspraken gemaakt met betrekking tot het uitvoeren van de opdracht, die hier opgenomen staan.

Er komt een redelijke hoeveelheid aan tijd vrij tijdens werktijd, voor deze opdracht

Dit was een van de grootste punten die ik goed afgebakend wilde hebben. Om goed aan de slag te kunnen met het project, zou ik genoeg tijd op het werk moeten hebben om aan het project te werken. Natuurlijk ben ik bereid om thuis ook aan het project te werken, maar het is goed om aan het project te werken op momenten dat de opdrachtgever in het pand is. De reden daarvoor komt in een komende afspraak. Uiteindelijk is in principe afgesproken dat er gemiddeld 3 werkdagen per week beschikbaar zijn voor het werken aan de opdracht. De overige tijd is voor overig werk, indien van toepassing. Als er tijd over is, mag deze besteed worden aan het project.

De code komt te staan binnen een versiebeheersysteem

Het is normaal binnen het bedrijf om code waaraan gewerkt wordt, op te nemen in een versiebeheersysteem. Dat is in het geval van dit project niet anders. Het komt te staan op de SVN server van het bedrijf. Er dient op z'n minst één keer per dag een commit te worden gedaan naar het systeem, op dagen dat er aan gewerkt wordt. Zo moet de integriteit van het project gewaarborgd worden.

Als er een belangrijk project tussendoor komt, moet daar prioriteit aan gegeven worden

Als er een belangrijk project, opdracht of bug binnen komt, moet hier prioriteit aan gegeven worden. Dit betekend in de praktijk dat het werk van het afstudeerproject neergelegd kan worden en dat er zo snel mogelijk aan het andere werk gewerkt moet worden. Dit bekend echter niet dat er geen tijd meer beschikbaar is voor het afstudeerproject, slechts dat het andere werk prioriteit erover heeft.

De opdrachtgever is beschikbaar om vragen te beantwoorden en mee te denken

Het project is in eerste instantie een intern project. Dit houdt in dat de opdrachtgever een collega is, wat zorgt een volledig andere dynamiek dan gevonden kan worden in een 'standaard' project. Het maakt de communicatielijnen stukken korter en er kan eenvoudig over de schouder mee worden gekeken. Daarnaast is de opdrachtgever thuis in de situatie en de oude code voor het maken van een

applicatie, wat hem een expert maakt over de gewenste werking de gemaakte applicatie. De afspraak is dat er altijd naar de opdrachtgever gelopen kan worden voor vragen en opmerkingen betreffende het project. Deze hoeft hij niet direct te beantwoorden maar hij komt er wel op terug.

3.3 Systeemontwikkelmethode

Er is geen vastgelegde systeemontwikkelmethode gebruikt bij deze opdracht. Er wel veel geleend van Extreme Programming[1]. Niet alle onderdelen hiervan waren even toepasselijk, maar er zaten een aantal onderdelen in die interessant zijn en die het project goed hebben gedaan. Hieronder staat een toelichting over de onderdelen die gebruikt zijn en in welke mate.

Iteratief werken

Extreme Programming, als Agile ontwikkelmethode, werkt met sprints. Een sprint is een periode van 1 á 2 weken waarin een van te voren bepaalde set aan taken wordt uitgevoerd. Over het algemeen wordt uitgegaan van een relevante user story, waarnaar toe wordt gewerkt. Er worden duidelijke taken omschreven en er wordt een tijdinschatting aan gegeven. Vervolgens worden de verschillende taken in de planning voor de sprint gezet.

Het biedt een houvast om te zien waaraan deze twee weken wordt gewerkt. Na een sprint wordt een reflectie gehouden op de uitgevoerde sprint en indien er werk over is wordt dit opgenomen in de stapel aan nog uit te voeren werkzaamheden. En het proces begint opnieuw, er wordt een set aan taken gekozen om uit te voeren, enzovoorts.

Door deze wijze van ontwikkelen wordt de voortgang nauwgezet gemonitord. Als er een lange planning wordt gemaakt, is het eenvoudig om het overzicht kwijt te raken en verzandt te raken in details terwijl er nog core-features gemaakt moeten worden.

Klant is onderdeel van het ontwikkelteam

De klant van dit systeem is mijn huidige werkgever, dat maakt dit onderdeel van Extreme Programming eenvoudig te implementeren. De wensen van de opdrachtgever, mochten deze wijzigen tijdens het ontwikkelproces, zijn eenvoudig te bespreken aangezien de opdrachtgever in hetzelfde pand zit. Dit is tijdens de ontwikkeling ook meerdere malen gedaan.

Eens in de 2 á 3 weken zijn er overlegmomenten geweest met de opdrachtgever en een andere collega om de status van het project te bespreken en om te zien of er in de applicatie op dat moment delen ontbraken of om nieuwe features te introduceren. Zo is het opslaan van de configuratie van een event-app in een algemene lokale database, ten einde op een later tijdstip van event-app te kunnen wisselen, uit een van deze meetings voortgekomen.

De meetings met de opdrachtgever hebben extra focus aan het project gegeven en zorgt uiteindelijk voor een beter en robuuster project.

User Stories

Voor het ontwikkelen moet er uitgegaan worden van realistische situaties waarin het product gebruikt gaat worden. Er worden user stories gemaakt die beschrijven het verwachtte gebruik is van het product, aan de hand waarvan de software geschreven moet worden. Daarnaast vormen ze de basis voor sprints. Uiteindelijk hebben er sprints plaatsgevonden waar geen user stories meer centraal stonden omdat ze in basis allemaal al geïmplementeerd waren en er op dat moment vooral debuggen en verbeteren op het programma stond.

[1] http://www.extremeprogramming.org/

Een zo simpel mogelijk ontwerp

Door een simpel ontwerp te hanteren is het systeem beter te onderhouden en te refactoren. Er is op verschillende manieren getracht hier op te letten. Een van de manieren waarop dit gebeurd is, is het gebruik van AngularJS. Een functionaliteiten als Views is belangrijk vanwege de modulaire structuur die de HTML pagina's en JavaScript schoner houdt. En Services die herbruikbare code introduceren die eenvoudig toe te passen en uit te lezen is wat de controllers een stuk leesbaarder en onderhoudbaar maakt.

Het gebruik van JSON om de configuratie te regelen, zorgt er daarnaast voor dat het inregelen van een applicatie een stuk makkelijker wordt.

Continue refactoring

De wensen van de opdrachtgever kunnen wijzigen, wat nieuwe of gewijzigde features tot gevolg kan hebben. Dit vereist vervolgens dat de code wordt aangepast of aangevuld. Dit is in de loop van de ontwikkeling veel gebeurd. Vooral op het gebied van de AngularJS services is veel gebeurd.

In het begin van de werkzaamheden met AngularJS zijn er veel verschillende services gemaakt, voor elke mogelijke wens wel een. Zo was er een getRelation en updateRelation service. Beide spraken dezelfde tabel aan, maar hadden beide de volledige code om de database uit te vragen in zich. Een van de belangrijkste en ingrijpendste was de gang van bovenstaande situatie naar specifieke services voor het communiceren met de database: dbQuery. Daarnaast zijn in de loop der tijd verschillende services samengevoegd. Een voorbeeld hiervan waren de services die verschillende onderdelen van een reservering beheren, zoals een om een reservering te updaten en een om de reservering op te halen. Deze zijn samengevoegd in één reserveringsservice. Uiteindelijk is er voor vrijwel elke data een specifieke service die de communicatie met de view en of database regelt.

Daarnaast zijn er ook veel wijzigingen geweest in de manier waarop het project is gestructureerd en hoe de verschillende controllers in elkaar steken.

Continue integratie

Software is een som der delen. Door de verschillende delen van de code bij elkaar te zetten en een build te maken, kan gezien worden of het systeem nog doet wat gewenst is. Een build is in dit geval een werkende versie binnen Chrome. Tijdens het ontwikkelen is gebleken dat het builden van een applicatie in Cordova een dermate groot beslag legt op de tijd van de ontwikkelaar, dat het niet rendabel is om een native app build te maken.

Regelmatige releases

Door regelmatig een oplevermoment te hebben, kan er door de opdrachtgever goed gezien worden hoe het project vordert. Daarnaast bied het voor de programmeur een goed inzicht in de huidige status en kan er beter gepland worden naar de toekomst.

Zoals eerder besproken tijdens het punt 'Klant is onderdeel van het ontwikkelteam', is er regelmatig geschakeld met de opdrachtgever en is er hierdoor een duidelijk beeld van de applicatie ontstaan. De event-app is in de eerste sprints niet klaar geweest om een feitelijke release / ingebruikname te faciliteren. Er ontbraken daarvoor te veel essentiële onderdelen. Richting het einde van de opdracht is het wel mogelijk geworden om verschillen releases te doen.

4. Requirements en aannames

Voordat aan een opdracht gestart kan worden moet er goed gekeken worden naar de requirements van de applicatie. Daarnaast zijn er bij aanvang van het project een aantal aannames. In dit hoofdstuk komen zowel de requirements als de aannames aan bod.

4.1 Requirements

De opdracht heeft verschillende requirements meegekregen. Deze zijn nodig om goed inzichtelijk te houden wat de verantwoordelijkheden van de applicatie zijn. Daarnaast zijn ze ook nodig om een duidelijk eindproduct te kunnen identificeren. Want als je niet weet wat er wel en niet bij de applicatie hoort, hoe kun je dan zeggen dat het klaar is. Hieronder staan enkele van de duidelijk gedefinieerde requirements van de opdracht.

De applicatie moet offline werken

Uit eerdere ervaringen met applicaties die voor evenementen zijn gemaakt, is naar voren gekomen dat er op de evenementlocatie slecht of helemaal geen internet aanwezig kan zijn. Dit kan de werking van een applicatie ernstig verstoren. Vandaar dat de keuze is gemaakt dat de applicatie offline moet werken. Het verzamelen van leads en het aan- en afmelden van een reservering moet te allen tijde kunnen. Er mag wel internet gebruikt worden, om bijvoorbeeld data te downloaden naar de applicatie of om data via de API naar Iventer te sturen, maar dit zijn vooral handelingen die voor een na een evenement uitgevoerd (kunnen) worden. Mocht er toch internet aanwezig zijn op de locatie, dan werkt de applicatie ook.

De applicatie moet op verschillende platformen kunnen draaien

Het is niet rendabel om de applicatie voor slechts één platform, zoals Android of iOS. Het moet mogelijk zijn om voor een klant de applicatie op elk gewenst platform te leveren. Als de klant bijvoorbeeld liever een iOS device gebruikt dan een Android, moet het mogelijk zijn om de applicatie daar op te laten draaien, zonder een lange ontwikkeltijd. Als referentie voor de nodige platformen kan uitgegaan worden van de afbeelding, aangezien dit de platform zijn met de grootste aantallen gebruikers. Echter de focus ligt voornamelijk op Android en iOS. Wanneer de applicatie eenvoudig opgezet kan worden in secundaire platformen zoals Windows Phone en BlackBerry, is dit mooi meegenomen. Het is echter geen requirement om op deze secundaire platformen te kunnen draaien.

4.1 De grootste mobiele platformen van het moment

Gegevens voor de applicatie moeten uit de API gehaald worden

Er zijn verschillende gegevens die gebruikt kunnen en moeten worden in de applicatie. Voorbeelden hiervan zijn de reserveringen die geplaatst zijn voor het evenement en de configuratie voor een evenement. Deze moeten vanuit de Iventer API worden gehaald en in de applicatie worden opgeslagen. Vervolgens wordt deze informatie gebruikt om te verifiëren of de persoon aan de balie zich heeft aangemeld voor het evenement en om te zien welke kaarten aan de persoon zijn gekoppeld.

Alle communicatie met Iventer moet veilig gebeuren

Er gaat veel vertrouwelijke informatie van en naar de applicatie, tijden het installeren van een event-app of het versturen van de gegevens vanaf de applicatie naar Iventer. Deze communicatie moet veilig gebeuren, om er voor te zorgen dat de gegevens niet onderschept kunnen worden. Vandaar dat de communicatie met Iventer moet plaatsvinden via een beveiligd kanaal. Een toekomstige

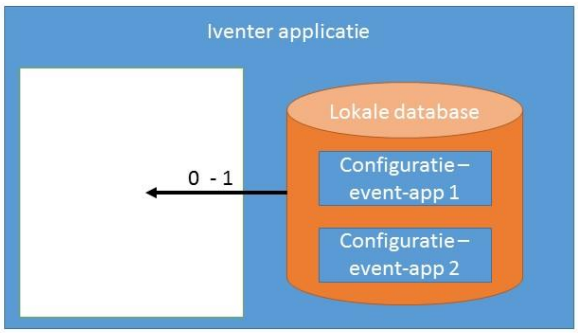
verbetering voor de beveiliging, zou het gebruik van tokens zijn voor de requests en responses. Op deze manier is het slechts mogelijk om vanuit een applicatie te communiceren met de server.

Scannen van een barcode moet ondersteund zijn

De reserveringen hebben allen een unieke scancode. Deze scancode hebben de relaties in hun e-mail ontvangen in de vorm van een barcode. Deze barcode tonen zij aan de ingang van het evenement waar een medewerker deze scant. Op basis hiervan moet gekeken worden of de reservering bekend is. Het biedt een snelle en persoonlijke manier om een reservering te vinden en vervolgens present te kunnen melden.

Meerdere event-apps binnen één applicatie

In de oude situatie werd er voor elk evenement een aparte applicatie ontwikkeld. Echter om tijd te besparen is het belangrijk dat er meerdere event-apps kunnen draaien binnen dezelfde applicatie. Na een evenement moet er eenvoudig een nieuwe event-app op gezet kunnen worden. Dit houdt in dat de formulieren, gegevens en delen van de lay-out gewijzigd moeten (kunnen) worden. Deze functionaliteit zorgt voor het sneller ontwikkelen en uitrollen van een event-app, omdat veel delen van de event-app al zijn opgenomen in de applicatie die vervolgens niet meer geprogrammeerd hoeven te worden. Dit houdt wel in dat de applicatie in grote mate generiek in opzet moet zijn.



4.2 De configuratie voor een event-app komt uit de lokale database

In figuur 4.2 is te zien dat de configuratie van een event-app in de lokale database opgeslagen is, naar de andere beschikbare event-apps. Deze configuratie kan ingeladen worden om zo de applicatie in te zetten voor een ander evenement.

De applicatie moet eenvoudig te debuggen zijn

Om een applicatie te maken die erg generiek in opzet is, moet er rekening gehouden worden met een grote hoeveelheid aan mogelijkheden en situaties. Om de testbaarheid en onderhoudbaarheid van de applicatie te vergroten is het van groot belang dat de applicatie eenvoudig te debuggen is. Er moeten meldingen over de verschillende functionaliteiten van de applicatie worden opgevangen en eventueel worden geretourneerd.

De applicatie is bedoeld voor tablets

Het doel van de opdracht is een mobiele applicatie, maar er zijn verschillende soorten mobiele applicaties, gericht op verschillende apparaten en schermgroottes. Wegens het tonen van de informatie en het invullen van de formulieren is in eerste instantie gekozen voor tablets met een scherm doorsnee van 9 tot 10 inch (formaat iPad). Dit heeft effect op het formaat van formulieren en de hoeveelheid van informatie die getoond kan/mag worden.

4.2 Aannames / concepten

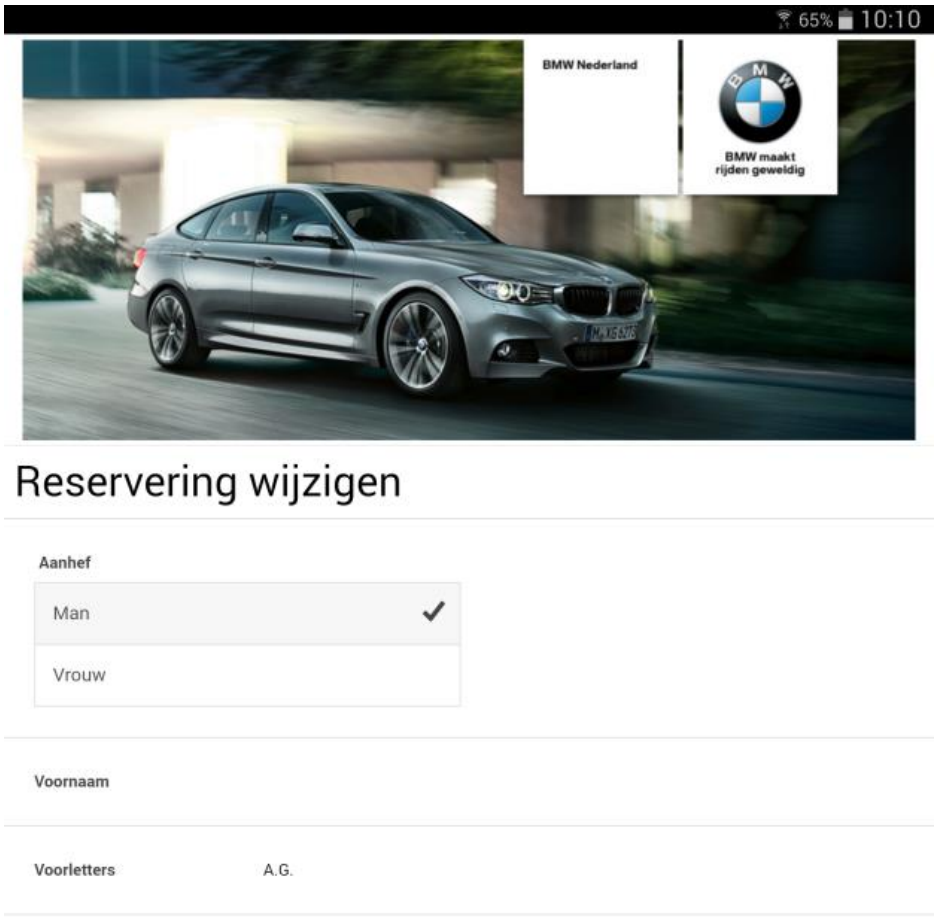
Tijdens het formuleren van de opdracht zijn verschillende aannames opgesteld. Een ander woord hiervoor zou ook ‘concepten’ kunnen zijn. Hiermee bedoel ik een idee voor hoe iets zou moeten werken of wat het juist niet zou moeten doen. Hieronder zijn enkele van deze aannames / concepten opgenomen. Het beschrijven hiervan is belangrijk omdat zij een basis vormen voor veel keuzes die zijn gemaakt tijdens de ontwikkeling.

De applicatie moet zoveel mogelijk instelbaar zijn via JSON

Dit is wellicht een van de meest belangrijke concepten en komt terug in vrijwel elk onderdeel van de applicatie. De applicatie bestaat namelijk uit een aantal losse onderdelen, zoals het present melden van reserveringen, het werven van leads en als laatste het synchroniseren van de lokale database met de online database van Iventer.

Het kan echter zijn dat een latere instantie van de applicatie, voor bijvoorbeeld BMW, geen reserveringen heeft om present te melden en slechts de applicatie wil gebruiken voor het werven van leads. In dat geval moet het menu icoon en alle daaraan gekoppelde acties niet beschikbaar zijn in de applicatie, aangezien hier geen gebruik van gemaakt gaat worden.

Er is voor JSON gekozen omdat het een gestructureerde manier biedt om data door te geven, die in de applicatie vervolgens omgezet kan worden naar HTML. Er kan eenvoudig een object extra worden opgenomen of een worden weggehaald. Daarnaast laat je zo het opbouwen van de HTML over aan de applicatie ingesteld kan worden om een bepaald onderdeel (zoals een men item) niet te tonen, omdat het in de JSON zo beschreven staat. Daarnaast kan de JSON, naast de applicatie, eenvoudig gebruikt worden in andere situaties zoals een website of een andere mobiele applicatie.



4.3 Een simpel formulier om de reservering te wijzigen, opgezet via JSON, getoond op een tablet

Daarnaast is het mogelijk om een leadformulier of reserveringsformulier volledig via JSON in te stellen. De enige standaard code voor het leadformulier is de functionaliteit om van de ene naar de andere pagina te gaan en om het leadformulier uiteindelijk op te slaan. Daarnaast is er een JavaScript object dat de JSON omzet in een HTML formulier en indien nodig deze over verschillende, vooraf in de JSON gedefinieerde, pagina's verdeeld. Op deze manier kan een formulier dus volledig op maat worden gemaakt. Dit heeft als groot voordeel dat er op deze manier geen applicatie ontwikkeld hoeft te worden per klant, maar dat de huidige applicatie hergebruikt kan worden. Dit bespaart veel ontwikkeltijd en communicatie met de klant.

Het is ook mogelijk een reservering verschillende gegevens per event-app mee te geven. In het reservering stuk van de code zit een uitgebreid zoeken pagina. Hier kan op een aantal zoekvelden gezocht worden. Welke velden dat zijn (voornaam, achternaam, e-mailadres, telefoonnummer, postcode, etc.) is ook via de JSON in te stellen. Iets dat een paar uur zou kosten om per event-app in te stellen en te testen, kan op deze manier vrij eenvoudig worden ingesteld.

In bijlage 9.5 is een stuk JSON te vinden dat het voorbeeld van figuur 4.4 zou kunnen genereren. Het maakt een pagina aan waar twee rijen zijn opgenomen, eentje voor de checkboxes en eentje voor het e-mailadres. Er wordt gebruik gemaakt van een behavior om het e-mailadres te tonen of niet. Over behaviors is meer te vinden in de volgende paragraaf.

Gebruik van behaviors in formulieren

Zoals eerder gezegd wordt het leadformulier volledig opgebouwd vanuit JSON. Echter het zal gewenst JavaScript gedrag niet aan kunnen maken. Dit is echter wel een vereiste. Het is de bedoeling dat er dynamische formulieren gemaakt worden. Dit houdt in dat een behoefte is aan een manier om extra gedrag te geven aan een formulier, iets om te extra invloed op de flow in een formulier te geven. Hier komen behaviors, 'gedrag' in het Engels, om de hoek kijken.

Behaviors geven conditionele dynamiek aan het formulier. Een voorbeeld van behaviors is de situatie dat als je radio button 'ik wil contact opnemen' aan klikt, een tekstveld voor een e-mailadres tevoorschijn komt. Behaviors werken met 2 sets aan elementen, triggers en targets. Beide zijn referenties naar bestaande elementen, veelal een HTML element. In de voorbeeld JSON in bijlage 9.5 is bij de behavior als trigger `dom::antwoord1` gebruikt en als target `dom::email` gebruikt. Hierbij betekend het `dom::` gedeelte dat wat er achteraan komt, een referentie is naar een dom element, op basis van een id. In jQuery kan deze vervolgens eenvoudig worden aangesproken. Wanneer een vooropgezette verandering plaatsvindt op een of meerdere triggers zal gekeken worden of de huidige situatie het toe laat dat er een vooropgezette actie wordt uitgevoerd op de targets.

Een voorbeeld in pseudocode van een dergelijke check zou zijn:

```
wanneer(klik op 'ik wil contact opnemen') {
    state = 'toon e-mailadres invoerveld'
}
```

Behaviors maken gebruik van een state, waarin is opgenomen wat de huidige staat van de targets is. Dit houdt in dat bijgehouden wat er aan de targets gewijzigd en/of toegevoegd moet worden zodra de state wordt uitgevoerd, zoals in het bovenstaande voorbeeld het tonen van een e-mailadres invoerveld. Voordat een pagina van een formulier volledig wordt opgebouwd, wordt gekeken of er behaviors zijn op deze of eerdere pagina's die een state hebben die uitgevoerd moet worden. Op

deze manier kan een checkbox die je op pagina 1 hebt aangevinkt, zorgen dat je op pagina 4 een andere set aan vragen te zien krijgt dan je anders zou zien.

Voor het geval er meerdere behaviors zijn dezelfde targets beheren, wordt er bij het uitvoeren gekeken naar de prioriteit die elke afzonderlijke behavior heeft. Zo wordt bij het uitvoeren van de behaviors gestart bij de behaviors met de laagste prioriteit en wordt er naar steeds 'belangrijkere' behaviors toegewerkt. Elke nieuwe behavior heeft de mogelijkheid om de wijziging van de vorige behavior te overschrijven door een wijziging te doen op hetzelfde target. Op deze manier worden alle behaviors uitgevoerd, maar wordt wel het gewenste resultaat bereikt.

Het kan voorkomen dat er verschillende behaviors tegenstrijdige handelingen uitvoeren op een target, zoals deze eerst tonen en vervolgens verbergen. Het afvangen hiervan is niet de verantwoordelijkheid van de applicatie. Dit is een punt dat ofwel bij de programmeur ligt, of afgevangen moet worden aan de kant van Iventer. Op basis van de prioriteit worden de verschillende behaviors uitgevoerd, of ze eerdere wijzigingen ongedaan maken of niet.

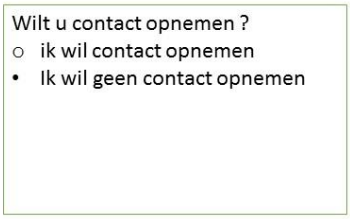
Behaviors zijn een hoeksteen van het concept dynamische formulieren, omdat het een eenvoudige manier bied om gedrag toe te voegen aan een formulier. Er zijn een aantal, veel gebruikte, behaviors standaard beschikbaar, de rest kan ook in string formaat worden meegegeven via de JSON en deze wordt dan ingeladen in het formulier.

Gebruik van rules in formulieren

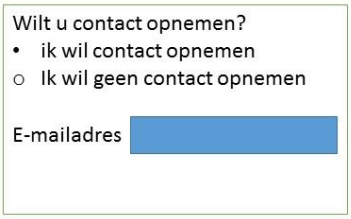
Naast behaviors, die gedrag toevoegen aan een formulier, zijn er rules die validatie toevoegen aan een formulier. Op zich zou dit opgelost kunnen worden door middel van HTML(5) attributen als `type`, `required`, `pattern`, `minlength`, `maxlength`, `min` en `max`. Maar dit is slechts een gedeeltelijke oplossing, voor meerdere redenen:

- Er moet namelijk ook de mogelijkheid zijn om een specifieke foutmelding aan te geven als het daadwerkelijk fout is gegaan.
- Er zijn moeilijke rules die ofwel meerdere elementen beslaan of ingewikkelde logica bevatten. Deze zijn niet in HTML alleen op te vangen en er moet daarom een manier zijn om deze af te dwingen.

Wat ook van belang is, is dat een rule genegeerd moet kunnen worden. Laten we de eerdere gebruikte situatie als uitgangspunt nemen. Als de checkbox ‘ik wil contact opnemen’ wordt aangevinkt, moet er een e-mailadres invoerveld komen. Als ik het niet aanvink, is het e-mailadres niet zichtbaar. Dit e-mailadres is verplicht en er ligt een regel op dat het verplicht maakt wat inhoudt dat als het niet is ingevuld er een validatie melding zal worden getoond. Echter wanneer ‘ik wil contact opnemen’ niet is aangevinkt en het e-mailadres invoerveld niet zichtbaar is, mag het veld niet verplicht en zichtbaar zijn. Want anders is een veld incorrect dat de gebruikers überhaupt niet zien of kunnen invullen. Hieronder staat het voorbeeld uitgebeeld, de gevulde bol geeft aan welk antwoord is geselecteerd.



4.4 Behavior verbergt het veld



Behavior toont het veld

Op deze manier kan een behavior invloed uitoefenen op een rule. Als namelijk het element wordt verborgen is de rule namelijk niet meer van toepassing. Daarnaast kan een rule ook feitelijk uitgezet worden. Een van de variabelen van een rule is of deze actief is of niet. Door deze variabele aan te passen vanuit een behavior, wordt de rule niet meegenomen bij het controleren van de invoer op de pagina. Dit is een voorbeeld van de manier waarop behaviors en rules samen moeten (kunnen) werken.

Gegevens beschermen via authenticatie

De applicatie beschikt over het algemeen over gevoelige informatie waaronder naam, e-mailadres en eventueel telefoonnummer en adres. Deze informatie dient slechts gebruikt te worden voor leads en eventueel om te verifiëren dat het om de juiste persoon gaat, wanneer een reservering opgezocht wordt.

Omdat deze informatie niet zomaar beschikbaar mag zijn en omdat het niet wenselijk is dat een ieder alles kan doen met de applicatie, is er een authenticatie methode in de applicatie gebouwd. De event-app kan alleen maar geopend worden met een gebruikersnaam. Zodra de applicatie wordt gesloten of het device op inactief wordt gezet moet er daarna weer opnieuw de gebruikersnaam ingevoerd worden.

Om toegang te krijgen tot de admin, moet het wachtwoord worden ingegeven. Zo wordt een cruciaal onderdeel van de applicatie extra beschermd. Het wachtwoord wordt ook gevraagd zodra een gebruiker een lead verwijderd uit de applicatie. Dit om er zeker van te zijn dat de gebruiker deze actie daadwerkelijk bedoeld heeft.

In het ontwerp van de beveiliging is aandacht besteedt over de vraag hoeveel authenticatie vragen er worden gesteld. Moet de applicatie zowel een gebruikersnaam en wachtwoord vragen of moet er alleen gebruikersnaam of wachtwoord worden gevraagd. Uiteindelijk is gekozen om beide te vragen, maar afzonderlijk van elkaar. Om in de event-app te komen is een gebruikersnaam nodig. Dit is een handeling die vaak gedaan gaat worden (elke keer als de event-app gesloten wordt moet er opnieuw worden ingelogd). Maar voor verdere handelingen in de applicatie, zoals het versturen van de data of wijzigingen aan de leads moet vervolgens een wachtwoord worden opgegeven. De reden dat een wachtwoord wordt gevraagd in plaats van een gebruikersnaam, is omdat een wachtwoord minder eenvoudig te raden is als een gebruikersnaam en daarom meer veiligheid kan betekenen. Over het algemeen zal een gebruikersnaam bestaan uit onderdelen van de naam van de medewerker en/of het evenement. Aangezien je met meerdere mensen daar rond loopt kan geraden worden, op basis van de gebruikersnaam van de een wat de gebruikersnaam van de ander is. Als de gebruikersnaam van Henk ‘HenkIventer2014’ is dan zal de gebruikersnaam van Sjaak wel ‘SjaakIventer2014’ zijn.

Een onderdeel dat is opgenomen bij de gebruikers en wordt gebruikt is het concept van een `accesLevel`. Dit houdt in dat per gebruiker ingegeven kan worden tot op wel niveau van de applicatie deze mag komen. Het bekijken/ present melden van reserveringen ligt op niveau 0. De toegang tot het versturen van de data naar de API kan liggen op niveau 1. Het verwijderen van een lead en het resetten van de leads (dat ze opnieuw verstuurd kunnen worden) grijpen erg in op de effectiviteit van de applicatie en het is cruciaal dat deze onderdelen niet door een ieder gedaan kunnen worden. Vandaar dat deze handelingen alleen gedaan kunnen worden op niveau 2.
Deze opzet geeft de event-app meer structuur en zorgt er voor dat de gegevens in de applicatie wellicht veiliger zijn opgeslagen, wat de kwaliteit van de applicatie ten goede komt.

5. Uitvoering sprints

In dit hoofdstuk wordt het proces beschreven, hoe de applicatie tot stand is gekomen. Hier komt een chronologische beschrijving van de genomen stappen en beslissingen. Dit hoofdstuk is opgedeeld in de verschillende sprints die genomen zijn. De verschillende sprintplanningen kunt u vinden in de bijlage.

5.1 Pre-sprints

Zoals opgenomen in het plan van aanpak, is de eerste week van het project besteedt aan het uitdenken en uitschrijven van het plan van aanpak zelf. Dit om genoeg tijd te nemen voor het uitdenken van de opdracht en de benodigdheden hiervoor. Na deze periode is een tijd van een week genomen om user stories te definiëren en intern te bespreken. Deze user stories zouden centraal staan voor de gehele opdracht en moeten vandaar goed uitgedacht en gedefinieerd zijn. Aangezien er gebruik gemaakt is van een vorm van Extreme Programming, zijn de user stories in de loop der tijd enige mate aangepast, wat de kwaliteit van het product uiteindelijk ten goede is gekomen.

Na het stuk design is er begonnen met het codeerwerk. Het eerste punt dat op stapel stond was het doorgronden en kundig worden in de huidige code. Deze code zou gebruikt gaan worden als basis van de te bouwen Iventer applicatie en was tot dan toe de standaard om een applicatie voor een evenement te maken. De manier om de code eigen te maken was het ombouwen en verbeteren van de code op punten die van te voren waren aangemerkt als knel-/verbeterpunten.

Één daarvan was het dynamisch inladen van de behaviors. Tot dan toe werden alle benodigde JavaScript bestanden ingeladen bij het opstarten van de applicatie. Maar deze scripts zijn niet direct nodig bij het opstarten van de applicatie, bijvoorbeeld omdat het code betreft die pas op een veel later tijdstip aangeroepen wordt. Vandaar dat deze code ook op een later moment ingeladen kan worden.

Na hier een tijd mee te hebben doorgebracht en een tijd te hebben gekeken naar de mogelijkheden, ben ik tot de conclusie gekomen dat het niet werkbaar zou zijn om de oude code voor het maken van een applicatie om te bouwen. Het framework moest opnieuw opgebouwd worden om werkbaar te zijn, iets wat het project erg ten goede is gekomen op een later tijdstip.

In december is geen noemenswaardig werk verricht wegens werkdruk en slechte planning. Begin januari werd het duidelijk dat er te veel vertraging opgelopen was om de eis van 17 werkweken nog te halen. Vandaar dat er uitstel is aangevraagd, die is toegewezen op 21 januari.

Vanaf 14 januari is overgestapt op een gestructureerde wijze van uren bijhouden en plannen wat er gebeuren moet. Binnen het bedrijf heeft er met betrekking tot de opdracht ook een verandering plaatsgevonden, er is subsidie toegewezen aan het project. Dit houdt in dat er uren bijgehouden moeten worden en een duidelijk overzicht van werkzaamheden moet worden bijgehouden. Dit is gedaan door van vrijwel elke dag die besteed is aan het project, de voortgang vast te leggen. Alleen bij insignificante wijzigingen, wordt dit stuk overgeslagen. Dit verplicht schrijven heeft het uitdenken van het project gestimuleerd en het uiteindelijk schrijven van dit verslag erg veel geholpen.

Vanaf 14 januari tot het begin van de eerste sprint is enkele dagen gewerkt aan het concept van een dynamisch in te laden event-app. De gedachte is dat er een AJAX call gedaan kan worden naar de Iventer server, waarbij een set aan configuratie opgehaald kan worden die zowel het uiterlijk als de werking van de applicatie wijzigen. Hiervoor zijn enkele database tabellen aangemaakt in de Iventer database, om zowel de configuratie als de bestanden op te slaan.

5.2 Sprint 1

Deze sprint heeft gelopen van 20-01-2014 tot 31-01-2014.

Event-app configuratie

De sprint is begonnen met een voortzetting van de 'pre-sprint' periode, door het opvragen van de Iventer database voor event-app configuratie. De gegevens die uit de Iventer database komen betreffen:

```
- Revision (welke versie van een event-app gebruikt wordt)
- RevisionDate (van welke datum de versie is)
- Tables (een verzameling van tabellen, nodig in de event-app)
    - Name (naam van de tabel)
    - Query (een SQL script om de tabel aan te maken)
- Queries (de verzameling van SQL query's, nodig in de event-app)
    - Name (naam/omschrijving van de query)
    - Query (het SQL script voor de query)
```

Door deze instellingen moeten zaken als het aanmaken van gebruikersaccounts voor de medewerkers op het evenement en het inladen van de reserveringen geregeld kunnen worden. Deze instellingen moeten binnen komen via de API die vanuit Iventer geleverd wordt. Door de Iventer API url aan te spreken met als URL pad `/apps/[appTag]/settings`, worden bovenstaande gegevens in JSON formaat gepresenteerd.

Naast de settings is het ook belangrijk dat er zogenaamde 'assets' doorgegeven kunnen worden. Onder assets worden alle bestanden verstaan die van nut zijn voor een event-app, waarbij voornamelijk gedacht kan worden aan afbeeldingen, CSS en Javascript bestanden. Dit biedt een zeer interessante uitdaging. Hoe wordt er namelijk omgegaan met bestanden in JavaScript en kunnen er überhaupt bestanden worden ingeladen via een API om vervolgens herhaaldelijk gebruikt te worden?

FileSystem API

Na enig[2] onderzoek[3] ben ik uitgekomen bij de FileSystem API die onderdeel is van de HTML5 specificatie. Het biedt de mogelijkheid om gegevens op te slaan en op te vragen op de harde schijf van het apparaat, binnen een vooraf gedefinieerde ruimte. Deze ruimte kan permanent (tot de gebruiker het verwijderd) of tijdelijk (tot de browser besluit dat het weg moet) gereserveerd en gebruikt worden. Wat vooral belangrijk is aan het gebruik van deze techniek, is het feit dat het volledig offline werkt. Er hoeft dus na de initiële opslag van de bestanden, geen connectie meer gemaakt te worden met het internet. Dit zorgt er voor dat de requirement dat de applicatie offline moet werken een stap dichterbij komt.

De uitdaging bij de FileSystem API ligt vooral bij de complexiteit van de geboden methodes en diens asynchrone karakter. Om een bestand te kunnen lezen of schrijven moet er eerst een map worden aangesproken, vervolgens een bestand en uiteindelijk moet aangegeven worden wat er mee gedaan moet worden (lezen, schrijven, etc.). Dit zorgt voor veel complexiteit in het aanmaken van een bestand. Daarnaast zorgt het asynchrone karakter voor problemen bij het creëren van bestanden. Om een bestand te schrijven moet deze in een map zitten, als deze map niet bestaat kan het bestand niet geschreven worden. Dus moet er een methode geschreven worden die eerst de map aanmaakt

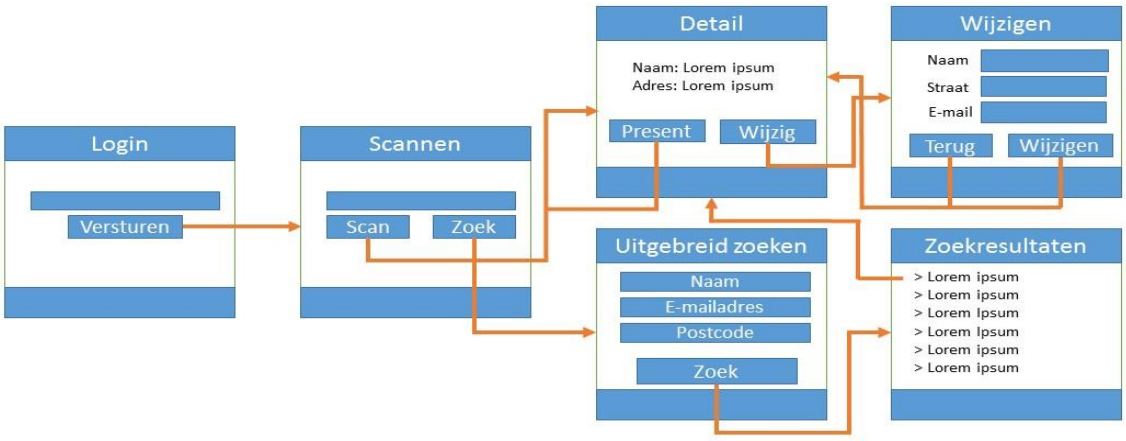
[2] http://www.html5rocks.com/en/tutorials/file/filesystem/
[3] http://net.tutsplus.com/tutorials/html-css-techniques/toying-with-the-html5-filesystem-api/

en dan pas het bestand. Dit zorgt voor problemen met betrekking tot timing en volgorde van uitvoeren.

Na een week werken aan de FileSystem API bleek dat JavaScript bestanden die opgeslagen waren, niet uitvoerbaar waren in Chrome, vanwege veiligheidsredenen. De JavaScript bestanden zijn wel uitvoerbaar binnen de Cordova applicatie, dus testen met de FileSystem API zal gebeuren moeten in een applicatie of een simulator. En uiteindelijk is er voor gekozen om het door ontwikkelen van de functionaliteit stop te zetten, aangezien het op dat moment geen basisfunctionaliteit betrof. Het was bijvoorbeeld op dat moment nog niet mogelijk om een reservering present te melden, iets dat veel belangrijker is voor de applicatie. In sprint 8 is de ontwikkeling echter weer opgepakt.

Pagina's voor reservering

In deze sprint is gekeken naar het present melden van een reservering. Hiervoor is ook een ontwerp gemaakt voor de flow die doorlopen kan worden door de applicatie, om reserveringen te scannen, zoeken, bekijken, present melden en wijzigen. Hieronder is zijn deze acties schematisch opgenomen, met daarbij welke knop naar welke pagina navigeert.



Overzicht mogelijke acties rond reserveringen

Nadat er door een gebruiker is ingelogd kan er gescand worden. Als de reservering gevonden wordt, worden de details zichtbaar waar de reservering present gemeld kan worden. De gebruiker komt dan weer terug bij het detailscherm. Via dit scherm kan er ook genavigeerd worden naar het wijzigen van een reservering.

Als een reservering niet gescand kan worden of er is bijvoorbeeld geen scancode aanwezig, kan er gezocht worden. Dit levert resultaten op, waardoor de gebruiker op het detailscherm komt. Vervolgens zijn dezelfde opties beschikbaar als die hierboven beschreven staan.

Lokale database

In de oude code voor het maken van een applicatie werd er gebruik gemaakt van WebSQL om de data op te slaan. Maar na enig onderzoek bleek dat WebSQL niet meer ontwikkeld werd door het W3C. Met de continuïteit van de applicatie in het achterhoofd ben ik op zoek gegaan naar een alternatief ofwel vervanger. Een goed alternatief leek IndexedDB te zijn. Na een dag onderzoek en een klein prototype gebouwd te hebben, las ik dat IndexedDB niet ondersteund wordt op iOS. Dit platform is belangrijk omdat het samen met Android een groot gedeelte van de markt in handen heeft. Om zoveel mogelijk platform te ondersteunen moeten ook technieken gebruikt worden die door zoveel mogelijk platformen worden ondersteund. Ook is onderzoek gedaan naar een abstractie laag zodat er eventueel geschakeld zou kunnen worden tussen WebSQL en IndexedDB, afhankelijk

van beschikbaarheid. Maar deze oplossing voegt veel complexiteit toe en op dit moment voldoet WebSQL aan de eisen die gesteld zijn.

AngularJS

Op de laatste dag van de sprint begon ik met kijken of het opzetten van het framework niet eenvoudiger kon. Voornamelijk het inladen van views en het routen van de ene naar de andere pagina vond ik lastig om onderhoudbaar en schaalbaar op te zetten. In mijn zoektocht naar een standaard framework dat hier in zou kunnen voorzien, kwam ik voorbij AngularJS. Meteen werd mijn aandacht getrokken door verschillende facetten van het framework zoals een nadruk op MVC en het idee van databinding. Vooral de nadruk op MVC sprak mij in eerste instantie aan, omdat dit schonere code oplevert dan de oude situatie vanwege het gebruik van controllers en views (in de vorm van templates).

Er is voorgenomen om huidige applicatie, bij wijze van experiment, de oude code voor het maken van een applicatie om te zetten naar AngularJS. Voornamelijk om te zien of het kan, of het snel gaat en of het inderdaad zo onderhoudbaar en krachtig is als verwacht.

Ionic Framework

Om het omzetten van de applicatie te kunnen vereenvoudigen is naast het ombouwen van de applicatie ook gekeken naar een CSS framework. AngularJS heeft namelijk van zichzelf geen visuele schil. Om de code te checken en om de kracht van AngularJS te ontdekken is dan een externe visuele schil nodig. Deze kan zelf geschreven worden, echter dit kost tijd die beter besteedt kan worden aan het uitzoeken van AngularJS en het vertalen van de oude code naar AngularJS. Via een collega kwam ik op het spoor van Topcoat[4], een CSS framework dat door Adobe is gemaakt. Het leek een mooie en sterk framework. Echter het framework bleek veel classes nodig te hebben om werkend te krijgen en was niet in de stijl die mij aansprak.

Na een kort vervolgonderzoek kwam de aandacht op het Ionic Framework, wat gemaakt is met en voor AngularJS. Het biedt niet alleen een CSS framework maar ook JavaScript componenten die AngularJS verbeteren en uitbreiden. Het nadeel is dat Ionic op moment van ingebruikname nog in Alpha fase zat. Echter omdat het voor nu is opgenomen als tijdelijke visuele schil, is het minder erg dat de code nog geen definitieve status heeft. Het betekend wel dat de code vrij vaak geüpdatet moet worden om bij te blijven met oplossingen en features.

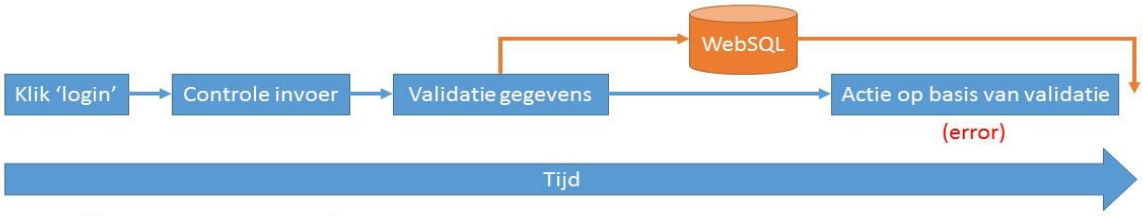
Deferred

Tijdens het omzetten van de applicatie naar AngularJS is het concept van Services naar voren gekomen. In AngularJS wordt gebruik gemaakt van controllers om de verschillende pagina's te besturen en er wordt gebruik gemaakt van services om data uit een externe bron te halen, uit een interne bron te halen of data tussen controllers door te geven. Snel werd al duidelijk dat een service gebruikt moest worden om met WebSQL te werken in AngularJS. Er staan nu 3 Services: 1 om de database aan te maken, 1 om de tabel voor de relaties aan te maken en ze te vullen en 1 om de relaties op te halen.

[4] http://topcoat.io/

Het gebruik van de eerste twee gaat goed, maar de laatste levert problemen op. De asynchrone werking van WebSQL maakt het moeilijk om met de data uit de database te werken. De rest van de code gaat door terwijl de WebSQL code wordt uitgevoerd. Ik moet een manier hebben om de data die terugkomt uit de WebSQL te gebruiken in de view.

Een schematische weergave van dit probleem is te zien in figuur 5.2. Hier is een voorbeeld genomen van een inlogmethode, afgezet tegen de tijd. Er wordt eerst gekeken of de invoer geldig is (is er een gebruikersnaam ingevoerd?), vervolgens wordt de invoer gevalideerd via de database (staat de gebruikersnaam in de database?). Echter de rest van de code gaat al verder en voordat de database een antwoord heeft gegeven, wordt er al actie ondernomen op basis van de validatie. Deze actie is echter ofwel niet mogelijk of gebaseerd op een default reactie van de validatie, wat niet altijd realistisch is en kan resulteren in een error. Uiteindelijk zal vaak pas na het uitvoeren van een actie op basis van de validatie, de database een resultaat teruggeven. Dan is het echter al te laat.



5.2 probleem met reactie uit webSQL

De oplossing ligt in een deferred[5] waarvan de `resolve` methode wordt gevuld met de gevonden relatie. Echter het correct opzetten van de deferred is nog een uitdaging, net als het uitvoeren van de code, zodat op het juiste moment de gegevens terugkomen.

5.3 Sprint 2

Deze sprint heeft gelopen van 03-02-2014 tot 14-02-2014.

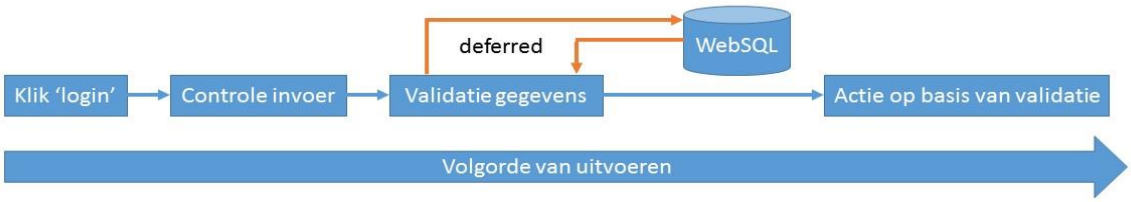
Overleg met opdrachtgever

Deze sprint is begonnen met een overleg met de opdrachtgever. Hij was heel januari op vakantie en heeft daardoor niets opgevangen van de beslissingen die genomen zijn tijdens deze periode. Tijdens het overleg heb ik onderdelen zoals AngularJS laten zien aan de opdrachtgever met de vraag of er een akkoord was om met dit framework verder te gaan. Daarnaast heb ik het idee gepresenteerd om verschillende event-apps in te kunnen laden in de applicatie.

Beide ideeën zijn goed gevallen en de opdrachtgever heeft zijn akkoord gegeven om in elk geval verder te gaan met de huidige koers, onder andere om te zien hoe dit zich ontwikkeld. Wel is aangegeven dat er goede communicatie moet zijn over de voortgang. Hierdoor heeft kan de opdrachtgever inzicht houden in de voortgang en de haalbaarheid van de nieuwe technieken.

Tevens is samen met Michiel een oplossing gevonden voor het WebSQL probleem met de services. De oplossing bleek dat wanneer elke aanroep richting de WebSQL een eigen deferred moest retourneren. Via een callback kan dan het model/ de view worden geüpdatet zodra de WebSQL code klaar is. In figuur 5.3 is deze oplossing te zien. Het voorbeeld is gelijk aan die van figuur 5.2 met enige verschillen. Zo is de tijd niet meer van belang, maar betreft het de volgorde van uitvoeren. Pas nadat de WebSQL een reactie heeft gegeven, wordt er een actie op basis van de validatie uitgevoerd.

[5] https://docs.angularjs.org/api/ng/service/$q



5.3 oplossing voor reactie uit WebSQL: deferred

Uitgebreid zoeken

Zoals in de eerste sprint besproken, is het van belang om te kunnen zoeken op een reservering. Vandaar dat er een 'uitgebreid zoeken' pagina opgenomen moet worden. De pagina heeft bovenaan een drietal invoervelden, voor de naam, het e-mailadres en de postcode.

In eerste instantie leek het mij een goed idee om de reserveringen onder elkaar te zetten en gebruik te maken van de filter functie van AngularJS om zo de lijst kleiner te maken. De reden voor die aanpak ligt vooral op het vlak van herkenning, je hoeft minder tekst in te voeren voordat je een resultaat hebt.

Als je namelijk iemand zoekt met als achternaam Slob, dan hoef je waarschijnlijk alleen maar de letter S, L en O in te typen in het filter invoerveld voordat de lijst dermate klein is dat je de reservering ziet staan.

Deze aanpak heeft echter een groot nadeel: er kunnen duizend(en) reserveringen in de lokale database staan. Deze moeten allemaal worden gepresenteerd en bij elke filter actie moet er doorheen gelopen worden. Vandaar dat er is gekozen om de lijst initieel leeg te houden en de lokale database uit te vragen op basis van de invoervelden. Dit maakt de lijst kleiner omdat het filteren/laden van de reserveringen het aantal te presenteren.

Dynamisch opzetten reserveringsformulier

Een van de requirements voor de applicatie is dat er verschillende event-apps gedraaid moeten kunnen worden binnen de applicatie. Elke event-app zou een ander format voor de reserveringen kunnen/willen hebben. Zo kan er voor gekozen worden om bepaalde informatie niet te vragen of te tonen in de applicatie.

Dit vereist een dynamisch formulier voor het wijzigen van een reservering, in het geval een wijziging gewenst is. De instructies voor het formulier worden opgehaald uit de Iventer API in JSON formaat en worden vervolgens omgezet in een formulier. Er kwamen echter snel uitdagingen naar boven, hierna vindt u er enkele.

Uitdaging: radiobuttons werken niet

Ik maakte namelijk gebruik van de data-binding functionaliteit in AngularJS en in het formulier staan twee radiobuttons om aan te geven of het een man of vrouw betreft. Beide konden aangevinkt worden maar niet uitgevinkt worden. Dit is natuurlijk geen gewenst gedrag en moet derhalve opgelost worden.

Oplossing: andere opzet

AngularJS kon geen verschil maken tussen de twee radiobuttons omdat de `name` die doorgegeven was aan de view, niet opgenomen was. Door de `name` nu op te nemen in een algemeen object dat

van te voren al bekend is bij AngularJS, wordt het verschil wel gezien en werken de knoppen weer naar verwachting.

Uitdaging: formulier valideren

AngularJS herkent formulieren automatisch en valideert deze automatisch. Of je iets doet met deze informatie is volledig een eigen keus, een foutief ingevuld formulier kan verstuurd worden. De validatiestatus wordt door AngularJS beschikbaar gesteld zodat bij het versturen het formulier tegengehouden kan worden. Echter het valideren van AngularJS van dynamische elementen werkt niet goed. Waarschijnlijk omdat het valideren van de elementen gebeurt bij het inladen van de template. Echter het opbouwen van het formulier gebeurt pas ná het inladen van de template.

Oplossing: anders valideren

Op dit moment is de enige oplossing het valideren van de informatie bij het versturen van het formulier. Wellicht dat er een betere oplossing gevonden kan worden op een later moment.

Leadsformulier opbouwen

Net als bij het reserveringsformulier, moeten de leadsformulieren dynamisch zijn, zelfs veel dynamischer. De basis van de reserveringsformulieren is veelal hetzelfde (NAW- en contactgegevens), echter een leadsformulier is per evenement waarschijnlijk volledig anders. Hierdoor is het van belang dat het opbouwen van de leadsformulieren zo generiek mogelijk is.

Leadsformulier opzet: HTML of JSON?

Om het leadsformulier opzetten zo generiek mogelijk te maken, moet besloten worden hoe het formulier aangeleverd wordt.

Aan de ene kant kan gekozen worden voor server-side rendering, wat inhoudt dat het formulier opgezet wordt op de server en in HTML vorm doorgegeven wordt door de API. Het voordeel hiervan is dat het formulier al helemaal klaar is voordat het in de applicatie gebruikt kan worden. Het nadeel is dat het lastiger is om een element te wijzigen, aangezien er in de code gedoken moet worden om deze aan te passen.

Aan de andere kant kan gekozen worden voor het opbouwen van het formulier via JSON en deze vervolgens opbouwen via een JavaScript object die de JSON opzet in een HTML formulier. Het grootste voordeel is dat de code voor het formulier overal gebruikt kan worden, met wat voor HTML dan ook. Zo kan het ook eenvoudig in een website verwerkt worden, aangezien er geen aannames worden gedaan voor lay-out o.i.d. Daarnaast is het eenvoudig om elementen en onderdelen van elementen toe te voegen en er kunnen eenvoudig nieuwe type elementen worden gebruikt omdat alle onderdelen voor de elementen al aanwezig zijn. Het nadeel is dat het redelijk complex is om een javascript object te maken dat alle gevallen kan ondervangen.

Na het wegen van de verschillende voor- en nadelen, is gekozen om het formulier door te sturen als JSON en vervolgens om te zetten in een HTML formulier. Zeker het voordeel dat het formulier bruikbaar is op verschillende plekken en manieren vanwege de genericiteit die JSON biedt, heeft het oordeel gestuurd.

Leadsformulier JSON opzet

De opzet van een leadformulier is globaal gezien het volgende:

```
Pagina(s)
    |-> rij(en)
        |-> element(en)
            |-> rules
            |-> behaviors
            |-> element(en)
                |-> rules
                |-> behaviors
                |-> rules
```

Er kunnen verschillende pagina's zijn, deze pagina's hebben 1 of meerdere rijen. Deze rijen hebben vervolgens 0 tot meerdere element(en). Een element is ofwel een invoerveld (input of textarea), een fieldset of een 'group'. De laatste is een div die als groupering van een set aan elementen dient, maar geen fieldset is. Elk element kan 0 tot meerdere rules hebben, welke dienen om de invoer te valideren. Elk element kan 0 tot meerdere behaviors hebben, welke dienen om dynamiek te geven aan het formulier. Tevens kan elk element 0 tot meerdere elementen, zoals in het geval van een fieldset of group.

5.4 Sprint 3

Deze sprint heeft gelopen van 17-02-2014 tot 28-02-2014

Dynamisch opzetten leadformulier

Om een dynamisch formulier werkend te krijgen moeten een aantal punten verwerkt worden. Samen met collega Mark en mentor Michiel ben ik tot de conclusie gekomen dat de volgende onderdelen nodig zijn:

- Dynamisch een route/state injecteren
- Dynamisch de template instellen
- Dynamisch een formulier opbouwen
- Dynamisch een controller invoegen

Dynamisch een route injecteren

Een route instellen maakt de pagina bereikbaar via de router. Echter elke route moet uniek zijn. Er moet dus een route per pagina van een formulier komen, om zo uniek te blijven.

Dynamisch de template instellen

Bij een route kan een template(view) worden opgegeven. Dit kan, zoals tot nu toe gedaan is, door aan een HTML bestand te refereren. Het is echter ook mogelijk om een de HTML als string door te geven. Dit maakt het mogelijk om de HTML op te zetten en vervolgens op te nemen in de route.

Dynamisch een formulier opbouwen:

Dit is een punt dat ook naar voren kwam bij het werken aan het reserveringsformulier. Hier werd over alle punten heen gelopen en werden de verschillende elementen toegevoegd. Maar nu hebben we te maken met een formulier waar niet alleen formulierelementen moeten worden toegevoegd maar ook rules en behaviors. Dit, gecombineerd met het feit dat we een HTML template moeten

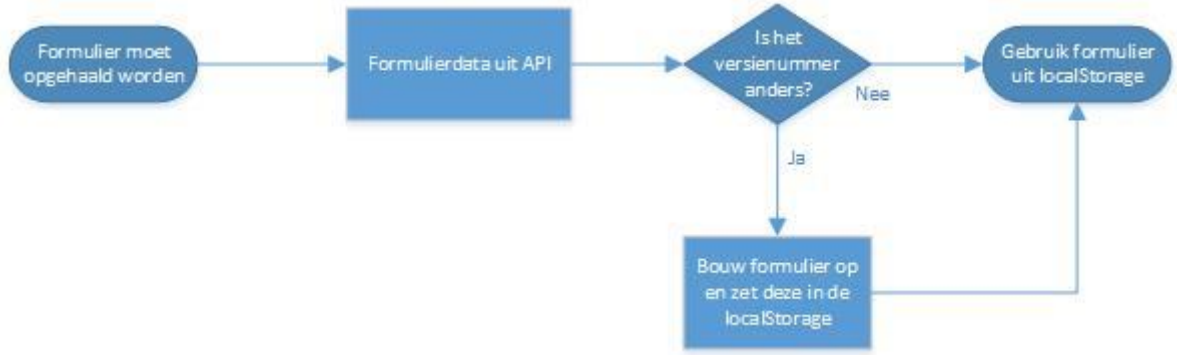
terug geven in string formaat, zorgt er voor dat er een goede manier gevonden moet worden om dynamisch het formulier op te zetten.

Dynamisch controller invoegen

Een route heeft niet alleen een template, maar ook een controller. De controller beheert het gedrag van de route, zoals het klikken op knoppen en wat er dan gebeurt. Dit is een integraal onderdeel van het werkzaam krijgen van de dynamische routes. Door het meegeven van een controller in de JSON en die uit te voeren via de `eval` methode, is het mogelijk om een controller in te voegen.

Cachen van het formulier via localStorage

Op dit moment wordt elke keer als de applicatie wordt ingeladen, het leadsformulier opnieuw opgebouwd op basis van een JSON string. Dit kost op dit moment ongeveer 250ms (milliseconden). Dit is op basis van een formulier van twee pagina's, met in totaal 5 rijen, 4 elementen en 6 sub elementen. Dit is geen realistische grootte, aangezien bij een 'normaal' leadsformulier uitgegaan kan worden van tientallen elementen, verspreid over 10 tot 20 rijen, op 1 tot 4 pagina's. Het genereren van een dergelijk formulier zal zeker meer dan 250ms in beslag nemen. Daar komt bij dat de gemeten 250ms komt van een test op een PC. De snelheid op een tablet zal waarschijnlijk lager liggen, vanwege de hoeveelheid aan processorkracht en RAM geheugen die beschikbaar is voor de applicatie.



5.4 flowchart om te bepalen of de localStorage gebruikt kan worden of (nog) niet

Om te voorkomen dat elke keer dat de applicatie gestart wordt het hele formulier opnieuw opgebouwd moet worden, moet de HTML van het formulier worden gecached. Na enig onderzoek is de keuze gevallen op de LocalStorage API[6]. Het biedt een eenvoudige manier om een string op te slaan op basis van een key. De key is in dit geval de samenvoeging van het formuliertype (leads) en het pagina nummer. Om te zorgen dat de cache niet te rigide is als er een wijziging optreedt wordt ook opgeslagen welke versie van de JSON het is. Als de versie die meekomt met de JSON anders is dan de versie die opgeslagen is in de LocalStorage, moet de LocalStorage worden geüpdatet.

Na enig werk met bovenstaande oplossing is echter besloten om de code die nodig is voor het cachen, uit te commentariëren, want het maakt het testen van wijzigingen bijster lastig als telkens het versie nummer van de JSON aangepast moet worden om de cache te vervangen.

Overerving van behaviors

De behaviors die gebruikt kunnen worden in de formulieren kunnen elk gewenst gedrag hebben. Het is echter wel van belang dat ze eenzelfde structuur hebben. Vandaar dat er standaard behavior opgezet is, die als basis moet gelden voor de verschillende behaviors.

[6] http://dev.w3.org/html5/webstorage/#dom-localstorage

BehaviorController

Om goed te kunnen werken met de verschillende behaviors die in de formulieren voorkomen, is er een JavaScript object in het leven geroepen om te werken met de verschillende behaviors. Dit object houdt een referentie van alle behaviors in zich. Omdat dit object alle behaviors in zich heeft, kan dit object hier allerlei dingen mee doen. Zo is een functionaliteit in dit object gemaakt, die alle behaviors die behoren tot een bepaalde pagina activeert. Door deze functionaliteit centraal te hebben, is er minder coupling en is de cohesie van de code vergroot aangezien alle bewerkingen met behaviors plaatsvinden binnen dit ene object.

Overleg over status

Op 21 februari is er een overleg over de status gehouden met Michiel en Marc. Tijdens dit gesprek is er gekeken naar verschillende onderdelen van de applicatie of deze in lijn liggen met de wensen voor het project.

Een van de punten die is besproken is het voornemen om in de toekomst het toe te laten om een reservering toe te voegen. Het is op dit moment geen verplichting, het is echter wel van belang om het systeem zo in te richten dat de mogelijkheid op een later tijdstip eenvoudig toe te voegen is.

Een belangrijke wijziging was dat er tot dan toe gewerkt werd met en relatieId, waarop een reservering gezocht werd. Dit is echter niet werkbaar in het licht van het toe kunnen voegen van een reservering. Als er namelijk een reservering wordt toegevoegd, is het relatieId hiervan het hoogste relatieId + 1. Echter op een ander device gebeurt iets soorgelijks, waardoor er twee reserveringen met hetzelfde id als present zijn gemeld. Als oplossing hiervoor is gekozen om het scannen te doen op basis van een barcodeTag, die uniek is en meegegeven is via de API. Wanneer een reservering toegevoegd wordt binnen de applicatie wordt aan de API kant, na het versturen van de data, een barcodeTag aangemaakt.

Tevens is er gesproken over hoe het wisselen tussen verschillende apps zou moeten werken. Vooral het opslaan van de gegevens in de WebSQL database is een belangrijk vraagstuk. Moet er per event-app een database aangemaakt worden of is er één grote reserveringentabel waarin een kolom is opgenomen met daarin de eventTag? Uiteindelijk is de keuze gevallen voor een database per event-app, omdat zo de data gescheiden is en niet kan conflicteren. En de databases blijven zo kleiner wat zoeken sneller maakt.

5.5 Sprint 4

Deze sprint heeft gelopen van 03-03-2014 tot 14-03-2014

Lead opslaan in WebSQL

Vanaf deze sprint is het mogelijk om een lead op te slaan in de WebSQL database. Tijdens het doorlopen van de verschillende stappen wordt aan het einde van elke stap de inhoud van die stap opgeslagen in de leadsService. Wanneer de laatste stap wordt verstuurd, wordt een save methode aangeroepen die de opgeslagen gegevens als JSON string in de WebSQL. De reden dat het opgeslagen wordt als JSON string is omdat op deze manier de WebSQL tabel voor de leads zo generiek mogelijk opgezet kan worden. Tevens kan de database tabel aan de kant van Iventer zo ook generiek opgezet worden, omdat er slechts een kolom opgenomen hoeft te worden waar de JSON data in gezet kan worden.

Algemeen log object

Om de applicatie beter te onderhouden en te debuggen te maken is de wens naar voren gekomen voor een algemeen log object. Het object moet op verschillende niveaus informatie over de

applicatie kunnen geven. Het moet errors tonen wanneer deze van toepassing zijn, maar wanneer er gedebugged moet worden waarom bijvoorbeeld een bepaalde JavaScript call niet lukt, moet ook elke stap van de applicatie geretourneerd kunnen worden.

Meer over dit log object is te vinden in hoofdstuk 7.5.

5.6 Sprint 5

Deze sprint heeft gelopen van 24-03-2014 tot 04-04-2014

Versturen van leads naar de API

Vanaf deze sprint het mogelijk om leads naar de Iventer API te versturen, door deze functionaliteit aan te speken in de admin. Wanneer deze wordt aangesproken worden de leads uit de WebSQL database gehaald, die nog niet verstuurd zijn. Het is namelijk niet de bedoeling dat alle leads opnieuw verstuurd worden. Wanneer alle leads opgehaald zijn worden ze in een lijst weergegeven waar vervolgens doorheen wordt gelopen. Per lead wordt een call naar de API gedaan om de gegevens aan te bieden. Wanneer deze call succesvol is wordt de lead als verzonden gemarkeerd in de WebSQL database. Wanneer de call onsuccesvol is wordt er niets weggeschreven in de WebSQL database. Vervolgens wordt de volgende lead verstuurd, net zo lang tot alle leads verstuurd zijn. Om de gebruiker inzage te geven in de voortgang is er een kolom in de lijst opgenomen die veranderd van ‘klaar om te versturen’ naar ‘verzonden’ of ‘fout met verzenden’.

app configuratie via appConfig

De configuratie van een event-app moet opgeslagen worden in de applicatie om deze bruikbaar te maken. Vandaar dat er een globale JavaScript variabele in het leven is geroepen: `appConfig` welke als functie heeft om alle gegevens van de event-app in zich te dragen en beschikbaar te maken voor de rest van de applicatie. De structuur van deze configuratie is:

```
appConfig {
 |browser:   (of het een browser of een iPad is),
 |devUuid:   (het nummer van de iPad of 'Browser'),
 |features:  (lijst met de functionaliteiten, beschikbaar in de app)
 |eventTag:  (welk evenement het is)
 |appTag:    (welke app het is)
 |version:   (welke versie van de app het is)
 |baseAPIURL: (welke api url gebruikt moet worden)
 |dbConfig:   (verschillende configuratie voor de webSQL databases)
   |databaseName:  (naam van de database van deze app)
   |importQueries: (lijst met tabellen die aangemaakt moeten worden)
     |table:   (naam van tabel)
     |APIUrl:  (waar de eventuele data vandaan moet komen)
     |creationQuery: (object voor het aanmaken van de tabel)
       |sql:        (de sql om de tabel aan te maken)
       |parameters: (eventuele parameters voor in de query)
     |importQuery:   (object voor het vullen van de tabel)
       |sql:        (de sql om de tabel te vullen)
       |parameters: (parameters voor in de query)
```

Deze configuratie wordt vervolgens opgenomen in een algemeen javascript object om zo in de applicatie gebruikt te kunnen worden.

Inloggen van gebruikers

Zoals besproken bij de aannames is het van belang dat de gegevens van de klant beveiligd zijn. Een manier om hier aan tegemoet te komen is door de gebruikers te verplichten om in te loggen. Er is enige tijd nagedacht over de mate van beveiliging waarbij de afweging tussen gebruikersgemak en veiligheid centraal stond. Uiteindelijk is besloten dat het van belang is dat de gebruiker de applicatie eenvoudig moet kunnen gebruiken. Vandaar dat er alleen maar een gebruikersnaam wordt gevraagd bij het opstarten van de applicatie en niet zowel een gebruikersnaam als wachtwoord. Het wachtwoord is wel nodig voor de acties die dieper ingrijpen de verwerking van de gegevens en de werking van de applicatie. Meer hierover is te vinden in paragraaf 7.4.

5.7 Sprint 6

Deze sprint heeft gelopen van 07-04-2014 tot 18-04-2014

Meeting over voortgang

Tijdens deze sprint heeft er wederom een meeting plaatsgevonden om de voortgang te bespreken. Er zijn een aantal punten besproken die tevens in deze sprint ook zijn behandeld:

Leads moeten opnieuw verstuurd kunnen worden

Tijdens het initieel versturen van een lead, kan het zijn dat er een serverfout optreed en de data niet juist is doorgekomen. Om dit tegen te gaan is het van belang dat álle leads opnieuw verstuurd kunnen worden. Alle leads worden verstuurd, om er zeker van te zijn dat álle data beschikbaar is op de server.

Een lead moet verwijderd kunnen worden

Het kan voorkomen dat een geworven lead niet correct is of vervangen is door een betere versie, bijvoorbeeld omdat iemand eerder niet de juiste gegevens had gegeven. Het moet mogelijk zijn om een lead te verwijderen, om zo extra werk aan het einde van een evenement te voorkomen. Het is meer werk om naderhand te onderscheiden welke lead de juiste is, dan als je het direct op het evenement doet.

Er moet documentatie geschreven worden, op een generieke manier

Er zijn verschillende onderdelen van een software ontwikkeltraject die belangrijk zijn voor het slagen ervan. Een daarvan is de documentatie, omdat dit toekomstgericht is. Door documentatie te hebben/schrijven, kan een andere ontwikkelaar er eenvoudiger mee aan de slag.

Het opzetten van de documentatie moet echter wel op een generieke manier, hiermee wordt bedoeld dat er gekozen moet worden voor een methode waardoor toekomstige documentatie stukken eenvoudig kunnen worden toegevoegd. De wijze waar op dit moet is op dit moment nog niet duidelijk.

5.8 Sprint 7

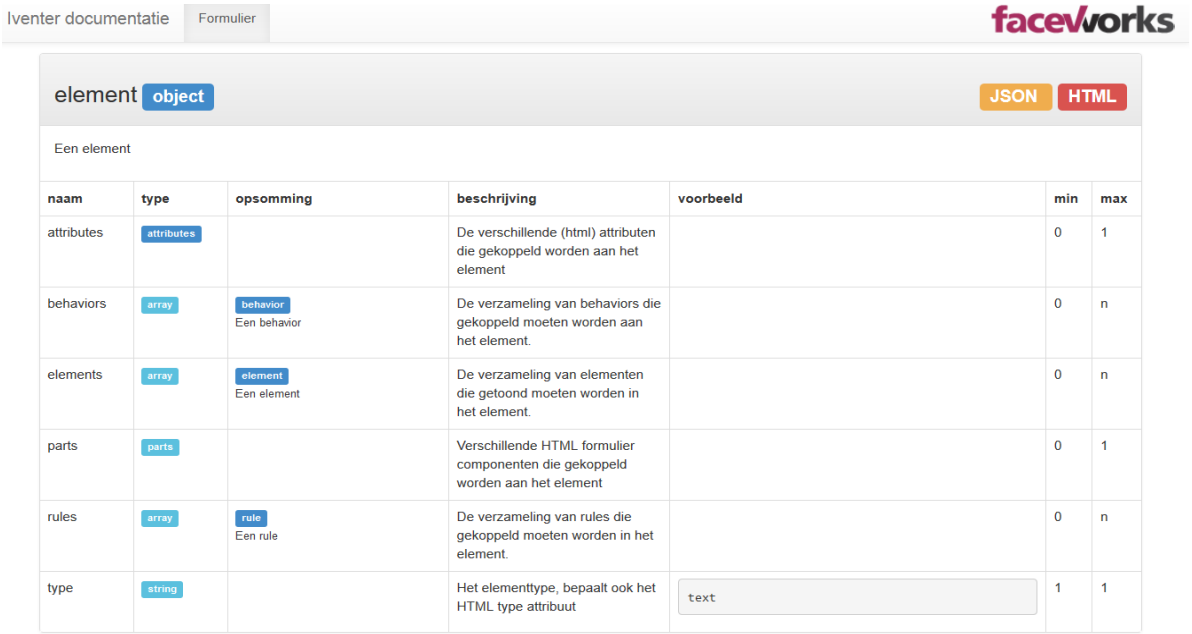
Deze sprint heeft gelopen van 22-04-2014 tot 02-05-2014

Documentatie schrijven voor de JSON

Zoals eerder beschreven is documentatie een van de belangrijke onderdelen van een software ontwikkeltraject. Om de documentatie voor deze applicatie goed op te zetten is vooralsnog vooral goed gekeken naar de documentatie die gekoppeld is aan het genereren van een formulier. Dit is een belangrijk onderdeel voor het correct werken van een event-app. Daarnaast is het ook een van

grootste punten waar een collega aan werkt bij het programmeren van een event-app, omdat zoveel andere onderdelen al vaststaan voor de applicatie.

De documentatie is opgezet in XML, dit biedt zowel structuur aan de data als een manier om de data optioneel te tonen, middels XSLT(eXtensible Stylesheet Language Transformations). Deze combinatie aan technieken maakt het bijvoorbeeld mogelijk om een zeer uitgebreide documentatie te maken voor een diepe inzage in de functionaliteit of om juist meer te focussen op de verschillende lagen van de documentatie.


Iventer documentatie Formulier facevworks

element object JSON HTML

Een element

naam	type	opsomming	beschrijving	voorbeeld	min	max
attributes	attributes		De verschillende (html) attributen die gekoppeld worden aan het element		0	1
behaviors	array	behavior Een behavior	De verzameling van behaviors die gekoppeld moeten worden aan het element.		0	n
elements	array	element Een element	De verzameling van elementen die getoond moeten worden in het element.		0	n
parts	parts		Verschillende HTML formulier componenten die gekoppeld worden aan het element		0	1
rules	array	rule Een rule	De verzameling van rules die gekoppeld moeten worden in het element.		0	n
type	string		Het elementtype, bepaalt ook het HTML type attribuut	text	1	1

6.x voorbeeld van de documentatie

Per element in de JSON staat opgenomen wat de naam ervan is, met daarbij het type. Er is, wanneer van toepassing, een mogelijkheid om een voorbeeld JSON object te zien en de HTML die hieruit resulteert. Ook is er een algemene beschrijving van het element opgenomen. Daarnaast staan er per element de verschillende variabelen opgesomd, met daarbij de volgende punten:

- Naam
- Type
- Opsomming: Wanneer het een array is, komt hier te staan wat voor types er in zitten
- Beschrijving
- Voorbeeld: een voorbeeld van een wat er verwacht kan worden als invoer
- Min: deze variabele moet minimaal zo vaak voorkomen
- Max: deze variabele mag maximaal zo vaak voorkomen (hierbij betekend ‘n’, oneindig vaak)

Door deze notatie wijze vast te leggen en door de dynamische wijze waarop deze data getoond kan worden, is dit de beste manier waarop de documentatie geregeld kan worden.

5.9 Sprint 8

Deze sprint heeft gelopen van 05-05-2014 tot 16-05-2014

Unit tests

Tot nu toe heb ik het testen vooral informeel aangepakt, door eventuele bugs in de software te vinden door exploratory testing, error guessing en door tegen fouten/onverwacht gedrag aan te

lopen. Echter om de kwaliteit beter te kunnen garanderen, is het van belang om ook unit tests te schrijven.

AngularJS is vanaf de grond af opgezet om eenvoudig(er) te testen te zijn en er zijn ook tools gemaakt door het AngularJS team om het testen eenvoudiger te maken. Zo is er ook een tool om unit tests te schrijven en te draaien: Karma. Deze tool werkt sterk samen met AngularJS en maakt testen redelijk eenvoudig. Het is opgezet in de lijn van behavior driven development, wat voornamelijk inhoudt dat er gedrag wordt gedefinieerd als testcase die vervolgens getest.

Om te zien wat het testen in zou houden heb ik als eerste unit tests voor een controller, een service en een filter geschreven. Dit heeft mij veel inzage gegeven in de testtechniek. Het testen is echter verder stopgezet, omdat er vanuit het bedrijf op dit moment geen verdere wens is om de applicatie verder te testen. De tijd en aandacht kan beter besteed worden aan het verder ontwikkelen van andere onderdelen van de applicatie. Eventueel kunnen er op een later moment meer tests geschreven worden om de kwaliteit van de applicatie verder te kunnen garanderen.

FileSystem API weer in gebruik genomen

Na enige maanden stil te hebben gelegen, is de FileSystem API weer in gebruik genomen. Nu er richting het einde van het project wordt gewerkt is de noodzaak van een manier om bestanden in de applicatie op te slaan weer naar boven gekomen.

De oude implementatie uit januari is echter volledig herzien. In het herschrijven van het object heb ik rekening gehouden met twee grote constructies die ik op andere plekken in de applicatie ook al had doorgevoerd:

1. Werken met deferred, promisses, etc. Elke functie moet ofwel een promise teruggeven en/of er een oplossen. Zo is het eenvoudig om het object te gebruiken in de applicatie.
2. Alleen bouwen wat nodig is. Eerst was er ook rekening gehouden met functionaliteit om alle bestanden in het FileSystem, te tonen in een lijst. Maar dat is niet nodig en zorgt voor veel ongebruikte code.

Door de overgang naar promises is het ervan een stuk vereenvoudigd. Een voorbeeld hiervan, staat hieronder, met eerst de oude structuur, gevolgd door de huidige.

```
filewriter.workFile('testnaam.txt', 'read', {})
localFiles.read('testnaam.txt').then(function(result){});
```

Oud

Oud

5.10 Sprint 9

Deze sprint heeft gelopen van 19-05-2014 tot 30-05-2014

End to end test met Protractor

Na verschillende unit tests ben ik in deze laatste sprint ook aangekomen bij de end to end test. Het framework dat hiervoor het meest geschikt is, is Protractor. Dit framework is geschreven door medewerkers van het AngularJS project en is specifiek bedoeld om te werken met AngularJS.

Een end to end test (of e2e test zoals het ook soms wordt geschreven) is om te zien of de flow in de applicatie voldoet aan het ontwerp. In het geval van deze applicatie moet er getest worden of de applicatie voldoet aan de user stories.

Echter vanuit Faceworks is aangegeven dat het, hoewel het belangrijk is om te testen, geen urgentie heeft om op dit moment te werken aan een end to end test. Vandaar dat ik na enkele uren onderzoek naar de werking en het nut van end to end testen, verder werk erin heb gestaakt.

6. Gebruikte software / technieken

Er zijn een aantal verschillende software en technieken gebruikt tijdens het ontwikkelen van de applicatie. Per software en techniek wordt hierna getoond waar ze bij het ontwikkelen een rol hebben gespeeld.

6.1 Apache Cordova[7]

Zoals in de opdracht is omschreven, moet er een applicatie gemaakt worden. Een applicatie houdt in dat er code geschreven moet worden die op een mobiel apparaat kan draaien. De verschillende mobiele platformen draaien op verschillende programmeertalen, Android draait bijvoorbeeld op Java en iOS draait op (Objective-) C.

6.1 De grootste mobiele platformen die ondersteund worden door Apache Cordova

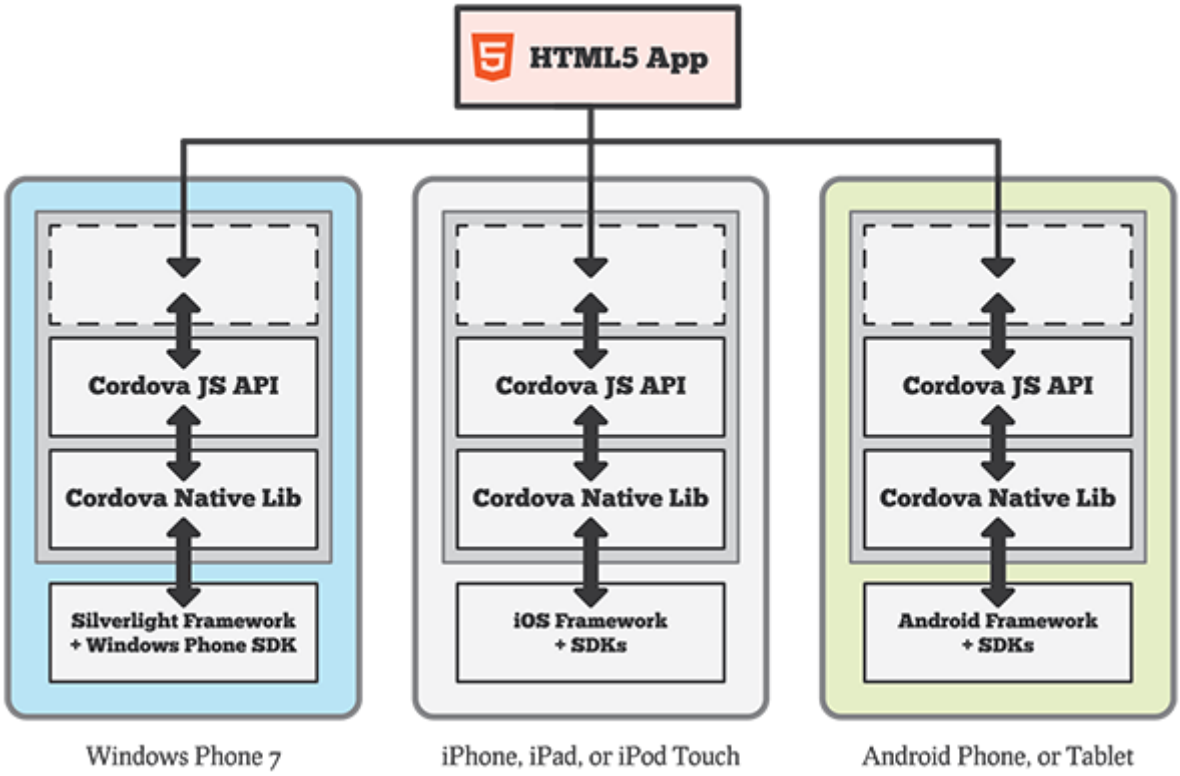
Beide talen zijn niet echt bekend binnen het bedrijf en het leren van een taal, om deze applicatie te bouwen, zou de tijdsinvestering niet waard zijn. Daarnaast is een van de requirements van de opdracht dat de applicatie op meerdere platformen moet draaien. Het leren van alle verschillende programmeertalen en het omzetten van de applicatie van de ene naar de andere taal is niet haalbaar. Vandaar dat een andere oplossing gevonden moest worden. Het antwoord is Cordova geworden.

Cordova beslaat twee kanten van het ontwikkelen voor mobiele apparaten. Aan de ene kant maakt het een applicatie aan, in de codetaal van het apparaat (bijvoorbeeld Java voor Android), waarin een webpagina getoond kan worden, de zogeheten WebView. Daarnaast biedt het een set aan API's die op basis van plug-ins toegang geven tot enkele van de features van de mobiel. Elementen als een camera of geolocatie kunnen dan aangesproken worden en gebruiken de onderdelen van de telefoon. Deze elementen kunnen via de API's, in JavaScript aangesproken en/of uitgelezen worden.

Het gebruik van Cordova zorgt er voor dat er geen specifieke kennis nodig is van de verschillende mobiele platformen, maar dat er slechts kennis nodig is van de webtechnieken (HTML, CSS en JavaScript). Vandaar dat de applicatie is ontwikkeld met behulp van HTML, CSS en JavaScript, waarbij gebruik gemaakt is van onderstaande frameworks en technieken. Dit werkt allemaal goed samen op een PC in een moderne browser als bijvoorbeeld Google Chrome, maar het is nog niet te goed te gebruiken op een telefoon.

[7] https://cordova.apache.org/

Om de applicatie vervolgens goed te laten werken op de mobiele devices wordt de applicatie opgenomen in een Cordova project die volgens gecompileerd wordt voor een of meerdere platformen. In het volgende schema is te zien hoe het de resulterende applicatie vervolgens werkt.



6.2 Schematische weergave hoe een gecompileerde Cordova-app werkt.

Hier is zichtbaar dat de html5 app is opgenomen in een app. Per platform wordt een aparte app gemaakt. Het deel met de onderbroken lijn is de WebView, een browser in een applicatie. Deze WebView communiceert met de Cordova JavaScript API's die vervolgens weer met de Cordova libraries praten die specifiek voor een bepaald platform zijn. Die libraries praten vervolgens met het platform om de applicatie te tonen en te kunnen beïnvloeden.

6.2 Lokale offline database

Omdat een van de eisen van Faceworks is dat de applicatie offline moet werken, is het niet mogelijk om een externe database aan te spreken om bijvoorbeeld een lead in op te slaan. Vandaar dat er een zoektocht is gestart naar een manier om deze data op het device op te slaan waarna het op een later tijdstip kon worden gesynchroniseerd met de online database. Er bleken twee keuzes te zijn: IndexedDB[8] en Web SQL[9], beide beschikbaar offline en beide gemaakt om data op te slaan, hoewel op radicaal verschillende manieren.

IndexedDB

Werkt met key-value paren, wat inhoudt dat er data opgeslagen wordt als een object. Een van de grootste voordelen van deze manier van opslaan is dat het dicht bij de structuur van JavaScript objecten komt, dat ook op basis van key-value paren is opgezet. Het is opgebouwd uit transacties, waardoor er een sterke gestroomlijnde structuur ontstaat.
Er zijn echter ook nadelen, zoals het feit dat er geen equivalent is voor LIKE zoals dat in SQL

[8] http://www.w3.org/TR/IndexedDB/
[9] http://www.w3.org/TR/webdatabase/

voorkomt. Daarnaast wordt het niet in alle browsers en mobiele devices ondersteund, waaronder iOS. Het is nog wel in ontwikkeling bij het W3C en de ondersteuning hiervoor in browsers wordt steeds groter.

Web SQL

Werkt met behulp van sqlLite, een lichtere, kleinere versie van SQL. Een van de grootste voordelen hieraan is **d**at dit een structuur biedt die goed te doorzoeken is. Daarnaast is het relationele model van SQL biedt veel mogelijkheden in het structureren van een database. Ook is SQL injection niet mogelijk, wanneer gebruikt wordt met het binden van een variabele in de query, door middel van `?` wat vervolgens weer gekoppeld kan worden aan een variabele.
Het grootste nadeel is dat Web SQL niet meer ontwikkeld wordt door het W3C sinds november 2010. Er zullen geen nieuwe versies meer voor komen en er zullen dus ook geen verbeteringen meer voor plaatsvinden.

6.3 Conceptueel overzicht werking databases

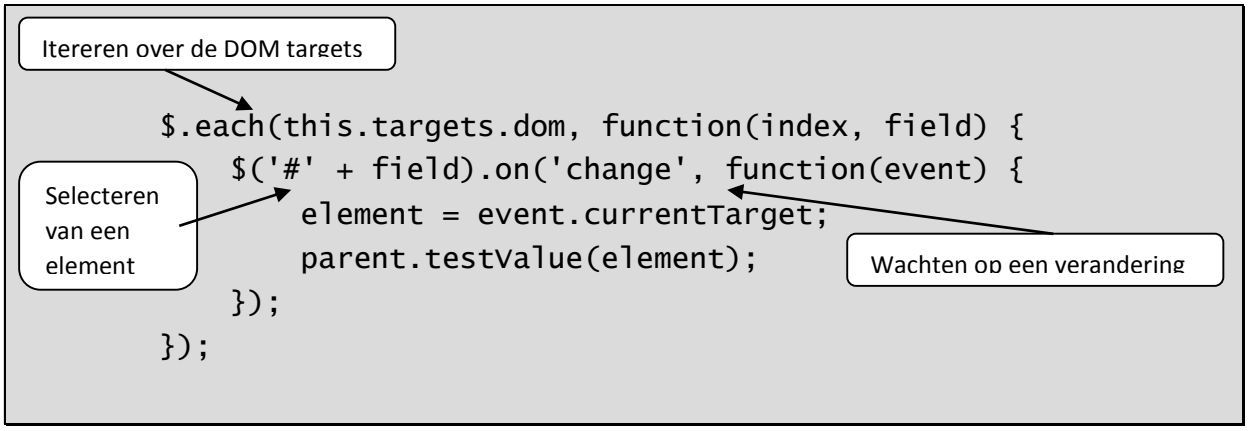
Er is gekozen om te werken met Web SQL. Deze techniek biedt op verschillende vlakken zoals doorzoekbaarheid, veiligheid en ondersteuning veel meer mogelijkheden dan IndexedDB op dit moment.

6.3 jQuery[10]

jQuery is een van de meest gebruikte JavaScript libraries ter wereld. Het is een library dat zich voornamelijk richt op de DOM, Document Object Model. Dit houdt in dat jQuery zich vooral bezig houdt met het doorlopen en manipuleren van de HTML structuur. Naast het doorlopen en manipuleren van de HTML structuur, is jQuery ook gebruikt voor de eenvoudige implementatie van Ajax, promises en events.

Selectie, iteratieve en event functionaliteiten

Een van de situaties waarin jQuery van pas komt is in de behaviors. Er worden targets en triggers meegegeven aan een behavior en hiermee moet gewerkt worden. Zo moet er naar alle triggers gekeken worden om te zien of er iets veranderd en als dat gebeurd, moet er actie worden ondernomen. Hier is een voorbeeld uit de behavior `onlyOne`.




```
$.each(this.targets.dom, function(index, field) {
    $('#' + field).on('change', function(event) {
        element = event.currentTarget;
        parent.testValue(element);
    });
});
```

[10] http://jquery.com

Hierboven is te zien dat er over een verzameling heen wordt gelopen `this.targets.dom`, wat de verschillende DOM nodes zijn waar naar gekeken moet worden en wanneer er een veranderd, moet de methode `testValue` aangeroepen worden van de parent (de behavior).

Ajax[11]

Naast het gebruik van de selectie en iteratieve functionaliteiten van jQuery, is ook gebruik gemaakt van de Ajax functionaliteit van jQuery. Het is voornamelijk gebruikt om de scripts die in de formulieren nodig zijn voor de behaviors en rules, in te laden. Deze functionaliteit maakt het eenvoudig om een request te doen naar een (extern) bestand of URL om date op te halen en vervolgens te gebruiken. Sinds jQuery 1.6 is een deferred syntax geïntroduceerd die sinds versie 1.8 als de officiële syntax is aangemerkt. Deze syntax gaat er vanuit dat er een initieel Ajax object wordt opgezet waar vervolgens verschillende methoden op worden aangesloten. De methoden die ondersteund worden zijn: `done` (als de request succesvol is afgerond), `fail` (als er een fout is opgetreden) en `always` (wordt altijd aangeroepen, zodra de request is afgerond).

6.4 AngularJS[12]

AngularJS is een framework dat wordt onderhouden Google en nog maar enkele jaren in omloop is. AngularJS noemt zichzelf een MVW framework wat staat voor Model – View – Whatever[13], wat inhoud dat zij zich richten op het model en view gedeelte en dat de programmeur voor zichzelf moet bepalen hoe hij het laatste deel ziet. Er zijn argumenten dat AngularJS een MV**C** framework is (vanwege de controllers) of juist een MV**VM** framework (omdat er een viewModel is in de vorm van de $scope die databinding regelt). Verder richt AngularJS zich voor een groot gedeelte op Single Page Applicaties, wat inhoudt dat er 1 basispagina is waar verschillende pagina's in worden geladen, iets dat voor de applicatie van groot belang is, om deze zo dynamisch mogelijk te krijgen. AngularJS ondersteund verschillende onderdelen van de applicatie die hieronder (kort) zullen worden toegelicht.

Views

Een van de functionaliteiten die gebruikt wordt, is het gebruik van views. In de oude code voor het maken van een applicatie was er één groot HTML bestand waar alle elementen voor de applicatie onder elkaar stonden. Hoewel er wel in één bestand gewerkt kan worden, is het erg moeilijk om overzicht te krijgen en te houden. Vandaar dat bij de overstap naar AngularJS, de views als een grote verbetering wordt gezien. Elke pagina in de applicatie heeft nu zijn eigen HTML bestand waar de HTML elementen van die pagina in staan. Dit past mooi in het idee van loose coupling, omdat zo de verschillende functionaliteiten op het vlak van views niet afhankelijk van elkaar zijn.

Bidirectionele data binding

Een functionaliteit die erg intensief is gebruikt, is de bidirectionele data binding die AngularJS biedt. Dit houdt in dat je een variabele kunt aangeven in de controller, die vervolgens gebruikt kan worden in de view. Maar ook dat als de variabele vervolgens wordt gewijzigd in de view (bijvoorbeeld via een formulier), wordt deze variabele ook gewijzigd is in de controller. Dit is vooral heel interessant bij het vullen van een wijzigingsformulier. Als je bijvoorbeeld een reservering wijzigt, hoef je slechts de reservering in te laden in de view en bij het opslaan kun je de vernieuwde reservering opslaan in de database. Een voorbeeld van een view is hierna te zien:

[11] http://api.jquery.com/jQuery.ajax/
[12] http://angularjs.org/
[13] https://plus.google.com/+AngularJS/posts/aZNVhj355G2

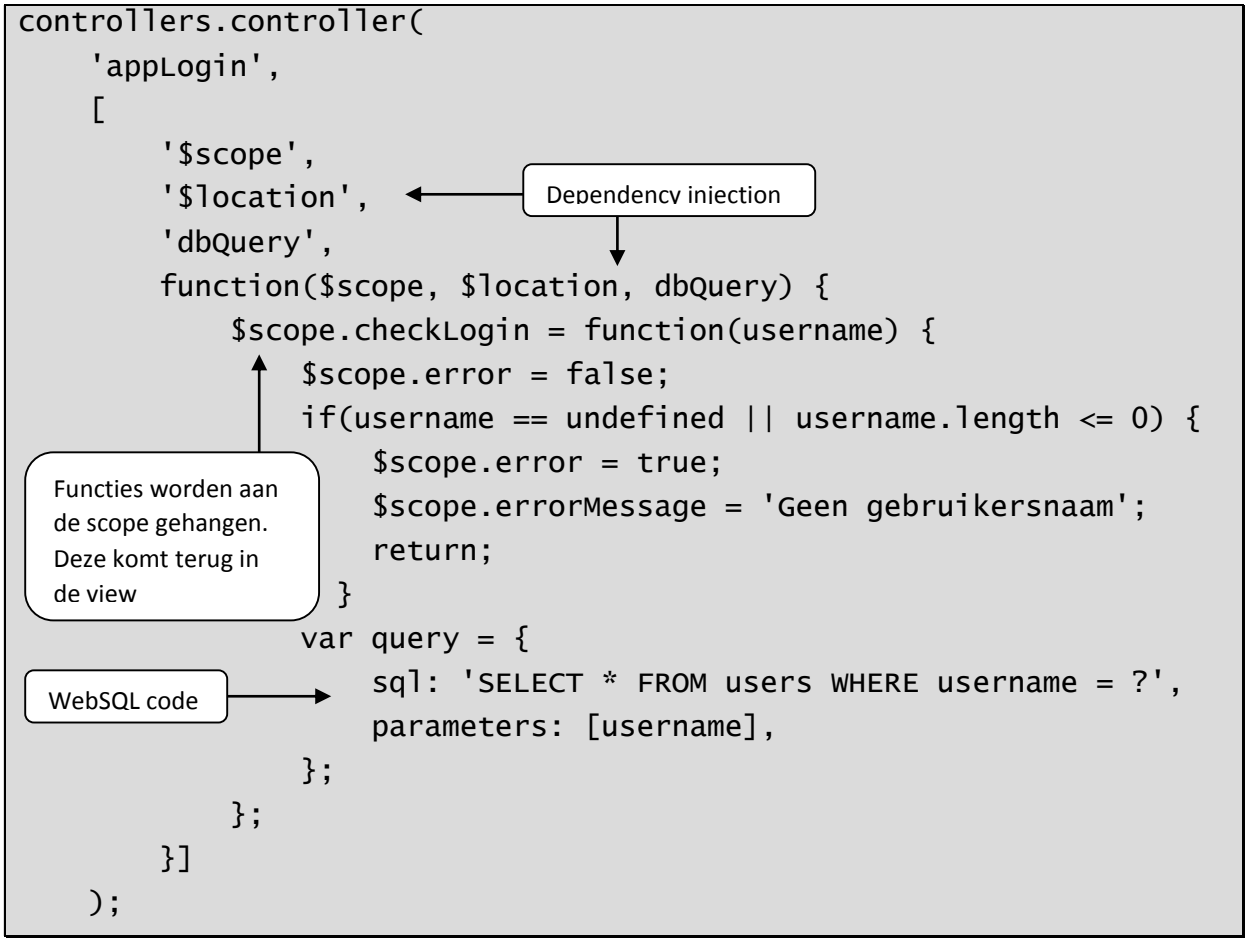
```
<ion-view title="inloggen">
    <ion-content scroll="true">
        <label class="item item-input item-input-inset">
            <input type="text"
                   ng-model="username"
                   placeholder="Gebruikersnaam" />

        </label>
        <label class="item error item-input item-input-inset">
            {{errorMessage}}
        </label>
        <button ng-click="checkLogin(username)"
                class="button button-block button-positive">
            Inloggen
        </button>
    </ion-content>
</ion-view>
```

Maakt de invoer beschikbaar aan AngularJS binnen $scope.username

Refereert aan de functie $scope.checkLogin in controller

In deze view zitten enkele onderdelen die AngularJS krachtig maken. De eerste is het gebruiken van de databinding. Door te refereren aan `{{errorMessage}}` wordt de errorMessage die ingesteld is in de controller op de pagina getoond. Door `ng-model="username"` te gebruiken, zorg ik er voor dat ik in de controller de variabele `username` kan aanspreken. Vervolgens is er een button opgenomen met als attribuut `ng-click="checkLogin(username)"` wat vrijwel hetzelfde werkt als een onclick attribuut maar is gekoppeld aan de scope in de controller. In de controller wordt een functie opgenomen met dezelfde naam en dezelfde parameters, waarin de gewenste functionaliteit is opgenomen. Hierna staat de controller voor de eerder getoonde view



In de code hiervoor komen een aantal belangrijke concepten van AngularJS terug, zonder er al te diep op in te gaan worden er hieronder een aantal van aangestipt. De eerste is Dependency Injection. Niet alle functionaliteiten binnen AngularJS zijn beschikbaar in alle controllers. Er moet van te voren worden gedefinieerd welke onderdelen nodig zijn voor het functioneren van een stuk code, in dit geval een controller. De onderdelen die in de controller worden ingeladen zijn de $scope en $location[14]. $scope verzorgt de bidirectionele databinding. Als je iets definieert binnen de $scope, zoals `$scope.error`, dan is `error` automatisch beschikbaar in de view. De AngularJS service `$location` geeft de mogelijkheid om de gebruiker naar een andere pagina te sturen. Door `$location.path('/event/relationScan');` aan te roepen wordt gebruiker na het inloggen doorgestuurd naar een andere pagina.

$q (ofwel jQuery Deferred)

Een andere belangrijke functionaliteit die is gebruikt van AngularJS is het gebruik van $q[15] wat onder andere is gebaseerd op jQuery Deferred[16]. In bijlage 9.4 is een schematische tekening opgenomen van de globale werking van het jQuery Deferred object. Door middel van deze functionaliteit kan een callback aangemaakt worden op elk stuk code. Het gebruiken van een callback is vooral handig op het moment dat er rekening gehouden moet worden met de volgorde van opdrachten. Het is veel gebruikt in de verschillende services, om zo bijvoorbeeld de aanroepen naar de WebSQL database uit te kunnen voeren, waarna de gegevens geretourneerd worden.

[14] http://docs.angularjs.org/api/ng/service/$location
[15] http://docs.angularjs.org/api/ng/service/$q
[16] http://api.jquery.com/jQuery.Deferred/

JavaScript is voor het grootste gedeelte synchroon opgezet, er is maar een doorlopende lijn en de opdrachten worden achter elkaar uitgevoerd. Er zijn echter gevallen waar het bovenstaande toch niet opgaat, zoals bij het ophalen van data uit de Web SQL database of het uitvragen van een API via AJAX. Beide zijn asynchrone processen, wat betekend dat als ze worden uitgevoerd, de browser alvast verder gaat naar de volgende expressie. Dit kan voor problemen zorgen bijvoorbeeld iemand krijgt een lege pagina te zien, omdat de data nog niet (helemaal) geladen is.

Daarom is er gebruik is gemaakt van $q, het geeft als het ware de browser een belofte dat er straks een stuk code is uitgevoerd. Deze belofte wordt afgegeven door `$q.promise`. Zodra een stuk code is afgerond kan de belofte worden vrijgegeven met behulp van `$q.resolve` als het succesvol is en `$q.reject` als het onsuccesvol is afgerond.

Deze belofte wordt vervolgens afgevangen door de `$q.then` methode die een drietal functies verwacht:

- `succes`, als de functionaliteit succesvol is uitgevoerd
- `error`, als er tijdens het uitvoeren van de functionaliteit iets fout is gegaan
- `notify`, als er tijdens het uitvoeren van de functionaliteit iets gemeld wordt vanuit de code

Hieronder is een stuk code opgenomen waar $q is gebruikt om de data van een lead op te slaan.

```
saveData: function(data) {
    this.addData(data);
    var promise = $q.defer(),
        query = buildQuery(this.lead);

    if(!query) {
        return;
    }

    dbQuery.run(query).then(
        function(data) {
            promise.resolve(true);
        },
        function(reason) {
            log.add(reason, 'error');
        },
        function(update) {
            log.add(update, 'info');
        }
    );
    return promise.promise;
}
```

Aanmaken defer object

Success

Succesvol afronden defer

Error

Notify

Promise teruggeven

In dit stuk code is te zien dat de `promise` variabele wordt aangemaakt als een `$q.defer()`. Vervolgens wordt een query opgebouwd die vervolgens in de `run` methode van `dbQuery` wordt gestopt, die zelf ook gebruik maakt van `$q.defer()`. Zodra deze succesvol is afgerond wordt `promise.resolve(true)` aangeroepen. Deze keer wordt er geen data teruggeven, omdat het in dit geval niet nuttig is.

Services

AngularJS heeft het concept van services, herbruikbare sets aan code die op verschillende plekken gebruikt kunnen worden. Over het algemeen worden services ingezet voor het ophalen van data uit een database of API of het verzenden van data daarnaartoe. In de applicatie zijn enkele services opgenomen die deze taken uitvoeren, er is bijvoorbeeld een `dbQuery` service gemaakt die als doel heeft om een query object te ontvangen, bestaand uit een SQL string en een array aan parameters.

Deze service en andere services zorgen voor een nog meer loose coupling van code en zorgt er ook voor dat code niet telkens geschreven/aangepast hoeft te worden, wat de onderhoudbaarheid zeer ten goede komt.

Naast eigen services wordt er ook gebruik gemaakt van verschillende AngularJS services, zoals `$location`, `$q` en `$resource`. De laatste is nog niet eerder aan bod gekomen, deze service is bedoeld om met REST API's te communiceren om vervolgens de data te retourneren. De $resource service komt vooral terug bij het initieel vullen van de Web SQL database, met gegevens uit de API. De code hiervoor is hieronder te vinden.

```
function retrieveData(defered)
{
    var dataQuery = $resource(
        http://test.nl/event/:eventTag,
        {},
        {
            query: {
                method:'GET',
                params:{
                    eventTag : 'eventTag'
                }
            }
        }
    )
    .get(
        {eventTag: this.eventTag},
        function(relations) {
            $.each(relations.data, function(key, value){
                importData.push(value);
            });
        }
    );
}
```

Placeholder aanmaken in URL

Placeholder koppelen aan een

Variabele koppelen aan de naam

De code hiervoor zet eerst een call klaar naar de `APIurl`, bijvoorbeeld `http://test.nl/event/:eventTag`, als GET call, waarbij de `eventTag` wordt doorgegeven die later gevuld wordt.

Vervolgens wordt er een feitelijke GET call gedaan, waarbij eerst de `eventTag` wordt gevuld met een variabele. Vervolgens word als de call volledig succesvol is afgerond, het nodige gedaan met de data die uit de call is gekomen.

Directives

Een set aan functionaliteiten die in AngularJS zit gebakken vallen onder de noemer directive. Directives zijn bedoeld om manipulaties aan de DOM mogelijk te maken en kunnen op verschillende manieren gekoppeld worden aan een DOM node. Zo kan er nieuwe tag gedefinieerd worden, zoals in het view-voorbeeld hierboven met `<ion-view>`. Het kan ook op basis van een class of op basis van een attribuut zoals `ng-if` in hetzelfde voorbeeld.

AngularJS is een aantal erg handige en interessante directives rijk waarvan de volgende ook gebruikt zijn in het project:

- ngApp: maakt het duidelijk onder welke app de applicatie valt
- ngClick: werkt hetzelfde als onclick, maar werkt wel samen met de $scope van de controller
- ngIf: creëert of verwijderd een DOM node op basis van een booleaanse expressie.
- ngModel: verbindt de waarde van een invoerveld aan een variabele binnen de $scope van een controller
- ngRepeat: loopt over een verzameling heen en maakt per iteratie een element aan.

Net als bij alle andere onderdelen van AngularJS, kun je zelf ook een directive schrijven die gebruikt kan worden in de applicatie. Tijdens het ontwikkelen is er echter geen situatie naar voren gekomen waar dit nodig bleek. Er zijn derhalve geen maatwerk directives opgenomen in het project.

6.5 Ionic framework

Ionic Framework[17] is een User Interface framework dat is gebouwd om samen met AngularJS gebruikt te worden. De styling is op zich wel apart te gebruiken, maar de JavaScript die in het framework wordt gebruikt, is op AngularJS gebaseerd. Het biedt een extra set aan directives die erg gericht zijn op het dichterbij brengen van mobiele interfaces.

Sidemenu

De applicatie heeft verschillende belangrijke componenten, zoals het scannen/aanmelden van reserveringen, het werven van leads en het versturen van deze data (via de admin). Wellicht zullen er in de toekomst nog wel componenten bij komen. Om de event-app werkbaar te houden, moet er een eenvoudige manier zijn om van pagina/component te kunnen wisselen. In het beginstadium maakte ik gebruik van een balk aan de onderkant met verschillende menu items, maar dat is iets dat al snel overweldigend zal zijn voor de gemiddelde gebruiker. Vandaar dat er een andere oplossing gevonden moest worden, welke is gevonden in de `ion-side-menu` directive[18]. Dit menu is onzichtbaar tot de gebruiker, door met de muis/vinger naar rechts te slepen, het tevoorschijn kan halen. Door naar links te slepen, wordt het menu weer verborgen. Het is zowel een handige als een mooie functionaliteit.

[17] http://ionicframework.com/
[18] http://ionicframework.com/docs/api/directive/ionSideMenus/

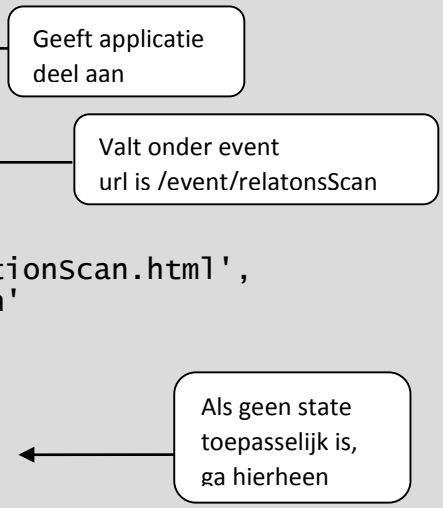
Routing

Routing is iets dat in AngularJS werkt, via `$route`, maar aangezien Ionic Framework gebruik maakt van de externe functionaliteit van UI-router uit de AngularJS-UI suite, neem ik het op bij het Ionic Framework gedeelte.

Er zijn verschillende pagina's in de applicatie die via een url benaderbaar zijn. Deze URL's kunnen allemaal worden ingesteld via de `$stateProvider`. Een voorbeeld hiervan volgt hierna

```
$stateProvider
    .state('login', {
      url: '/login',
      controller: 'appLogin',
      templateUrl: 'views/appLogin.html'
    })
    .state('event', {
      url: '/event',
      abstract: true,
      templateUrl: 'views/menu.html'
    })
    .state('event.relationScan', {
        url: '/relationScan',
        views: {
            'menuContent' : {
                templateUrl: 'views/relationScan.html',
                controller: 'relationScan'
            }
        }
    });

$urlRouterProvider.otherwise( '/login');
```



Hierboven is te zien hoe verschillende pagina's worden gedefinieerd. Zo wordt de login pagina als losse pagina gedefinieerd, dit omdat er niet naar een andere pagina genavigeerd mag worden als er nog niet ingelogd is. Ook wordt hier de controller opgegeven

`Event` wordt hier als abstract aangemerkt, dit houdt in dat het geen echte pagina is, maar dient als een basis. De `templateUrl` die hier is aangegeven, is de template die zal dienen als het menu voor alle subpagina's van `Event`.

`RelationScan` is opgezet als `event.relationScan` wat inhoud dat het een subpagina is van Event en dus valt onder het menu dat is aangegeven bij Event. Er wordt een view gedefinieerd in `menuContent`. Dit refereert aan een tag in `views/menu.html` wat er voor zorgt dat de inhoud van deze pagina wordt geladen in die tag. Het is dus mogelijk om op een pagina verschillende delen van het scherm te vullen met verschillende views met elk een eigen template en controller.

Als laatste geeft `$urlRouterProvider.otherwise` aan wat de standaard pagina is, als de gebruiker net binnen komt of wordt verwezen naar een onbereikbare/onbekende pagina.

Lijsten

In een applicatie als deze is het van groot belang om lijsten op een eenvoudige en nette manier te maken. Ionic Framework maakt het eenvoudig om een mooie en functionele lijst te maken, zowel via

CSS[19] als JavaScript[20]. Het geeft onder andere de mogelijkheid om iconen op te nemen in de lijst die pas zichtbaar worden wanneer er met de vinger of muis naar een bepaalde kant wordt geswiped. Het is vooral uit het oogpunt van scaffolding erg fijn om een framework te hebben dat met minimale moeite een mooie lijst kan genereren. Het zorgt voor snelle(re) ontwikkeling.

Overige, kleine punten

Er zijn nog elke kleine directives die gebruikt zijn en erg zichtbaar zijn of handig, maar niet dermate groot zijn om lang over uit te wijden.

- Ion-header: een snelle manier om een header samen te stellen, inclusief titel en knoppen links/rechts
- Ion-content: een handige directive om content te groeperen en om swipe scrolling te activeren, wat vooral fijn is bij lange lijsten en formulieren.
- Cards: een mooi gestijlde manier om een stuk data, zoals reservering details, te tonen.

6.6 JSON

Het staat voor JavaScript Object Notation en wordt gebruikt om data op een gestructureerde maar vrije manier te versturen en ontvangen. De notatie van JSON zit vast aan een bepaalde opzet, maar de verdere invulling ervan niet. Het is daarom een ideale taal om gegevens naar een API te sturen en ervan te ontvangen. Het is eenvoudig uitbreidbaar, eenvoudig te gebruiken en het past erg bij opzet van JavaScript als geheel, vandaar de referentie ernaar in de naam.

JSON wordt gebruikt om te communiceren met de API van Iventer. Er wordt JSON naar toe gestuurd en er komt JSON terug. Aan beide kanten word het snel omgezet naar een bruikbaar (JavaScript of PHP) object. Dit object kan vervolgens eenvoudig uitgevraagd, uitgebreid of uitgedund worden.

JSON wordt op twee grote manieren gebruikt in de applicatie, die daarna worden toegelicht:

- Configuratie
- Opslaan en verzenden van leads/reserveringen

Configuratie

De event-app is voor een groot gedeelte configureerbaar via JSON. Het volgende kan geconfigureerd worden:

- om welke event-app het gaat
- welke functionaliteiten beschikbaar zijn
- welke instellingen deze nodig hebben
- welke databasenaam er aangemaakt moet worden
- welke tabellen aangemaakt moeten worden
- hoe deze tabellen gevuld moeten worden
- en meer

Het is een makkelijke en snelle manier om een applicatie op te starten. Het JSON object wordt via de API verstuurd, omgezet naar een JavaScript object en vervolgens gebruikt om de applicatie in te stellen.

[19] http://ionicframework.com/docs/components/#list
[20] http://ionicframework.com/docs/api/directive/ionList/

Door gebruik van JSON is het eenvoudig om onderdelen toe te voegen. Het is daarnaast doorzichtig voor nieuwe ontwikkelaars om aan de applicatie te werken omdat er zoveel configureert kan worden.

Opslaan en verzenden van leads / reserveringen

Wanneer er data vanuit de API komt over de verschillende reserveringen worden delen opgeslagen in JSON formaat, zoals de relatie (hoofdboeker) en de verschillende kaarten. Het is eenvoudiger en voor nu efficiënter, om deze gegevens op te slaan in een eigen kolom versus het aanmaken van aparte tabellen voor reserveringen en kaarten die vervolgens weer relatief tot elkaar zijn.

Het opslaan van een lead werkt ongeveer hetzelfde, een lead wordt opgesteld door de verschillende pagina's van het formulier te doorlopen en uiteindelijk op de opslaan knop te klikken. Dit slaat een aantal kolommen op als zoekvelden, maar de volledige antwoorden set wordt opgeslagen als een JSON string in één kolom. Het scheelt ruimte en maakt de applicatie veel generieker. In plaats van dat er voor elk evenement veel configuratie nodig is om alle verschillende velden in het formulier een plek te geven in de database, kan het nu in één keer in de database gezet worden.

Zoals eerder gezegd wordt deze data vervolgens verstuurd als één lange JSON string om op de server verwerkt te worden en opgeslagen te worden in de database.

Bovenstaande voorbeelden geven het gebruik van JSON in de applicatie weer. Met de toepassing van JSON is het mogelijk om het gewenste generieke karakter van de applicatie invulling te geven.

6.7 Testen

Tijdens het ontwikkelen van de applicatie is er uiteraard getest, zij het minder formeel. Richting het einde van de ontwikkeling is er echter ook gekeken naar geautomatiseerde technieken om te testen. Hieronder zijn de verschillende manieren beschreven waarop getest is en wat de impact hiervan is geweest.

Exploratory testing

Zoals elke programmeur test ik alle functionaliteiten door ze in de applicatie te gebruiken. Een voorbeeld hiervan is het testen van het ophalen van een reservering uit de WebSQL database. Hiervoor schrijf ik de code in de controller, controleer ik welk record in de WebSQL database aanwezig is en test ik vervolgens de functionaliteit om te zien of het juiste record opgehaald wordt.

Bij deze vorm van testen komt het toevoegen van debug code erg van pas. Zo kan gezien worden wat een waarde was voordat deze wordt beïnvloed of wordt getest, zodat gezien kan worden of de code correct functioneert. Deze vorm van testen is niet echt officieel, maar het geeft wel een idee of de applicatie in een 'optimale' setting werkt. Met 'optimale' setting bedoel ik, het gebruik maken van een browser om de applicatie te draaien en het volgen van precies de juiste route in de applicatie. Het testen van onjuiste invoer wordt natuurlijk wel licht getest, maar daar wordt minder aandacht aan gegeven.

Mobiele applicatie testen

Naast het door klikken van de applicatie is het van groot belang om te blijven controleren of de applicatie werkt en er goed uit ziet als mobiele applicatie. Er is vanaf de eerste sprint, vrijwel elke week minstens één build van de applicatie gemaakt in Apache Cordova.

De applicatie reageert heel anders op een tablet dan op een PC. Zo is er een andere manier van invoer, heb je niet de precisie van een muis en toetsenbord. Daarnaast is het scherm een heel ander

formaat en moet er het scherm horizontaal houden ook rekening gehouden worden met de situatie dat het scherm verticaal gehouden kan worden.

Al met al is het wederom een niet officiële manier van testen, het biedt vooral een wijze om te zien of de applicatie op de tabel overeenkomt met de applicatie op de PC en/of dat het voldoet aan de verwachtingen.

Karma[21] voor unit tests

Naast de onofficiële manieren van testen, is er ook gebruik gemaakt van een test framework voor unit tests. Karma is een test framework dat is ontworpen opgezet door het AngularJS team om unit tests te draaien voor AngularJS. Karma is een JavaScript applicatie die draait binnen NodeJS en ingesteld kan worden via een configuratiebestand. In dit bestand kan aangegeven worden waar de testbestanden staan, maar ook waar de bronbestanden staan die nodig zijn voor het testen. Daarnaast kan aangegeven worden welk test framework gebruikt moet worden, standaard wordt er gebruik gemaakt van Jasmine[22]. Wanneer de tests worden gestart, wordt er een browser opgestart waar de tests in gedraaid worden. Er wordt dus gebruik gemaakt van een realistische situatie om de tests in te draaien, dit is vooral van belang om elementen zoals het navigeren tussen pagina's te testen.

Uit tijdoverwegingen heb ik slechts één instantie van een AngularJS onderdeel getest. Dit houdt in dat ik slechts één controller, service en filter heb getest. Dit vooral om het principe te bekijken en om genoeg te leren om op een later tijdstip meer tests te kunnen schrijven.

Jasmine is een behavior driven test framework, wat inhoudt dat er uitgegaan wordt van een gedrag dat getest moet worden. De syntax van de tests is hier ook op afgestemd. Hierna is een voorbeeld van een testopzet. Het is een test om te zien of het filter `stringToDate` doet wat het moet doen. Voor elke test moet de Iventer module worden ingeladen, dat is de namespace waar de applicatie en alle code daarbinnen onder valt. Vervolgens moet het filter in de code worden geïnjecteerd via de `Inject` methode die aangeboden wordt binnen Karma. Deze kan AngularJS services inladen om te gebruiken. Zo word hier de `$filter` service ingeladen, de service waar alle gedefinieerde filters in staan. Zodra het filter dat getest moet worden, stringToDate, opgeslagen is in een globale variabele kan het testen beginnen.

Er zijn twee behaviors gedefinieerd die in de `it` methode zijn opgenomen, de naam conventie is vervolgens om de naam van de behavior te formuleren in de vorm 'het zou [gedrag] moeten doen'. Vervolgens wordt dit getest in de opvolgende functie. Er wordt code klaargezet en uitgevoerd en aan het aan het einde van de functie wordt er een verwachting neergezet in de `expect` methode. Het resultaat van deze methode moet true zijn, oftewel wat verwacht wordt moet overeenkomen met het feitelijke resultaat.

[21] http://karma-runner.github.io/0.12/index.html
[22] http://jasmine.github.io/

```
describe('Filter: String to date', function () {
    var stringToDate;

    beforeEach(module('iventer'));

    beforeEach(
        inject(function($filter) {
            stringToDate = $filter('stringToDate');
        })
    );

    it('should form a date correctly', function () {
        var dateString = '2014-05-07 23:08:33',
            dateFormatString = 'HH:mm (dd-MM)',
            result;
        result = stringToDate(dateString, dateFormatString);
        expect(result).toEqual('23:08 (07-05)');
    });

    it('should just parse a date, if no format is provided',
function () {
        var dateString = '2014-05-07 23:08:33',
            dateFormatString = '',
            result;

        result = stringToDate(dateString, dateFormatString);
        expect(result).toEqual(1399496913000);
    });
});
```

Naam van de test

Voor elke test de nodige code inladen / aanmaken

Het gedrag dat getest wordt

Verwachte resultaat, moet true zijn om te slagen

Naast het filter heb ik de controller `reservationScan` getest, omdat dit een belangrijke controller is vanwege het feit dat het een van de hoofddoelen van applicatie mogelijk maakt. Door het scannen van een reservering kan naar het detailscherm worden genavigeerd, waar de reservering present gemeld kan worden. Tijdens het testen van deze controller heb ik gemerkt dat de huidige applicatie niet goed testbaar was. Dit kwam omdat bij het opstarten van de applicatie er een check werd gedaan of er een event-app ingeladen was en als dit niet zo was, wordt de gebruiker naar een andere pagina gestuurd. In een unit test situatie is er geen event-app ingeladen, onder andere omdat het niet van toepassing is voor de code. Deze situatie heeft er in geresulteerd dat er een onderscheid is gemaakt tussen de test module `iventerBase` en de normale applicatie module `iventer`. Deze code is alleen van toepassing op de tests en heeft verder geen effect voor de werking van de applicatie.

Daarnaast heb ik de service `dbQuery` getest, omdat dit de belangrijkste service is. Alle WebSQL query's worden door deze service uitgevoerd, met uitzondering van de query's bij het opslaan van een event-app in de Iventer WebSQL database. Vandaar dat het belangrijk is om te testen of deze service doet wat er van verwacht wordt. Bij deze tests is erg van belang geweest wat de volgorde was. Door de tests heen is er namelijk een tabel aangemaakt, gevuld en de laatste twee tests proberen deze uit te vragen. Tijdens het testen is er gemerkt dat de service verbeterd kon worden. Zo is het punt toegevoegd dat als een query niet goed is opgezet, dat de promise moet worden geresolved met `false`. Daarnaast moest er nog een controle komen of de query überhaupt correct is, voordat de database wordt geraadpleegd.

Protractor[23]

Protractor is een end to end testing framework ontworpen en opgezet door het AngularJS team. De werking ervan ligt in de lijn van Karma, het maakt gebruik van de WebDriver om de testen realtime in een browser uit te voeren om zo te zien of het gewenste effect wordt bereikt. De code voor het schrijven van een end to end test is vrijwel gelijk aan die van de unit testen, het is ook opgebouwd volgens het principe van behavior driven development.

Een end to end test is bedoeld om te zien of de applicatie doet wat het moet doen, door verschillende scenario's te testen. Het is een equivalent van een integratie test, omdat de verschillende unit geteste onderdelen samen worden gepakt en worden getest. Door het draaien van een end to end test kan meer gezegd worden over de kwaliteit van de applicatie.

Zoals gezegd lijkt de syntax van een end to end test erg op die van een unit test, met wat uitgebreide functionaliteit:

- Het is mogelijk om eenvoudig van pagina te wisselen via het `browser` object
- Ook is er het `protractor` object, waarmee bijvoorbeeld getest kan worden op welke URL de gebruiker nu is.
- Daarnaast is het `by` object, waarmee elementen gevonden kunnen worden op basis van een selector, op basis van een model of op basis van de data-binding.
- En er is het `element` object, waarmee verschillende checks gedaan kunnen worden op een geselecteerd (HTML-) element.

Door deze uitgebreide functionaliteiten dragen bij aan de mate van de eenvoud waarmee deze tests uitgevoerd kunnen worden.

Zoals eerder al is besproken in hoofdstuk 5.10 heb ik geen end to end test kunnen maken en draaien, ik zie echter wel in dat het een krachtige tool kan zijn om de kwaliteit van het eindproduct extra te controleren.

[23] https://docs.angularjs.org/guide/e2e-testing

7. Uitdagingen en oplossingen

Elk project heeft zijn eigen set aan uitdagingen die overkomen moeten worden, bij dit project net zo. In dit hoofdstuk wordt ingegaan op een aantal uitdagingen die tijdens het project naar voren zijn gekomen en hoe er mee om is gegaan.

7.1 De event-app moet offline werken

Een van de belangrijkste eigenschappen van de event-app is dat deze altijd moet kunnen werken. Tijdens het ontwikkelen van de huidige generatie aan applicaties, is geconstateerd dat de beschikbaarheid van internet beperkt is. Het inzetten van een 3G verbinding is een prijzig alternatief, zowel in de initiële aanschafkosten van het device als in de kosten die gemaakt worden tijdens het gebruik.

Vandaar dat de applicatie voornamelijk offline moet werken en alle taken moet vervullen zonder dat er internet beschikbaar is. Het gaat in dit geval om het present melden van bezoekers en het werven van leads. Beide zijn uitermate belangrijk, maar dienen zonder internet gerealiseerd te kunnen worden. De vraag is echter hoe? Hoe wordt er gecontroleerd of de bezoeker die aan de balie staat ook feitelijk ingeschreven is? Hoe worden de leadgegevens ontvangen en opgeslagen? Dit zijn vragen die centraal staan bij deze uitdaging.

Oplossing

Het eenvoudigste antwoord op deze uitdaging is ‘geen internet gebruiken’. Maar dat is eenvoudiger gezegd dan gedaan. Uiteindelijk is het niet gelukt om de applicatie 100% offline te krijgen. De basisfunctionaliteiten kunnen offline. Echter het present melden van een reservering kan alleen als de reservering is geverifieerd. En de gegevens hiervoor staan primair in de Iventer database.

De oplossing hiervoor is gevonden in een Web SQL database die de gegevens per reservering in zich houdt. Ook de leads en de gebruikers van de applicatie worden in een Web SQL database opgeslagen. Dit houdt in dat het device lokaal deze data in zich heeft, want deze vorm database werkt volledig offline. Echter de gegevens, zoals de leads en welke reserveringen present zijn, moeten nog wel naar de Iventer database worden gestuurd.

Dat is het deel waarvoor een internetverbinding nodig is. In het admin gedeelte van de applicatie zit de mogelijkheid om de gegevens naar de server te sturen. Dit kan alleen als er een internet verbinding is. Door middel van het Apache Cordova `network` object kan uitgevraagd worden of het device met internet is verbonden. Als dat zo is, dan kunnen de gegevens worden verzonden. Anders is de mogelijkheid van verzenden niet zichtbaar en/of wordt de gebruiker naar het basisscherm van de applicatie gestuurd.

Uiteindelijk is dus het belangrijkste gedeelte van de applicatie offline beschikbaar gemaakt. Dit betekend wel dat de gegevens die in de lokale database staan, voor waar aangenomen moeten worden. Dus als er iemand aan de balie staat met een scancode, dan wordt de lokale database geraadpleegd en als de reservering daar niet in voorkomt, is de reservering klaarblijkelijk niet correct.

7.2 De applicatie moet zo generiek mogelijk zijn

Een eigenschap van de oude code voor applicaties was dat deze erg gericht waren op een specifieke klant. Hoewel de globale opzet vast staat, wordt er een groot gedeelte op maat gemaakt voor de klant. Dit kost tijd en daardoor geld. De wens is om zoveel mogelijk van dit werk te verleggen naar instellingen in plaats van naar de feitelijke code.

Natuurlijk is niet alles generiek te coderen. Als een klant liever geen header heeft of een grote wijziging aan de standaard wil, zal een versie van de applicatie specifiek voor deze klant gebouwd moeten worden. Maar de wens is om de applicatie zo generiek te maken dat de basis applicatie maar één keer gebouwd hoeft te worden en dat door het inladen van een andere configuratie, de applicatie inzetbaar is voor een andere klant. De vragen die hierbij centraal staan, liggen dus vooral op het gebied van de inzet en overdracht van deze configuratie. Wat is er allemaal configureerbaar? Hoe wordt de configuratie overgezet? Hoe kan gewisseld worden tussen configuraties? Hoe heeft de applicatie toegang tot deze configuratie?

Oplossing

Na veel refactoring om de applicatie zo generiek mogelijk te maken, is deze op een punt beland dat het meeste werk hiervoor gedaan is. Het is mogelijk om de database tabellen in te laden en te vullen, om de zoek-/lijstvelden voor leads en reserveringen in te stellen, om de verschillende formulierconfiguraties in te laden en op te bouwen (hierover hieronder meer). Ook is het mogelijk om de styling van een applicatie grotendeels specifiek voor de klant te maken. Het is in elk geval mogelijk om een stuk CSS door te geven dat wordt toegepast op de applicatie. Door naar de Super admin te gaan binnen de applicatie kan eenvoudig tussen de verschillende beschikbare event-apps geschakeld worden. Deze functionaliteit is echter alleen beschikbaar voor medewerkers van Faceworks, door middel van een wachtwoord.

Waar het generieke van de applicatie op dit moment stopt is in de opbouw van de views en de harde HTML. De verschillende indelingen van de pagina's zoals het detailscherm van de reservering staan hard in een html bestand. Het is in de toekomst mogelijk om dit ook door te geven via configuratie, maar het is op dit moment een dermate kleine winst dat dit het werk niet waard is. Daarnaast zijn er performance overwegingen waar vanuit gegaan moet worden. Het laden van een HTML bestand uit de cache is vrijwel altijd sneller dan het opbouwen van dezelfde code uit een JavaScript object.

Voor nu is er zoveel configuratie mogelijk dat het voor de simpele applicaties mogelijk is om slechts de configuratie te laden om het device klaar te maken voor gebruik. Voor de grotere en ingewikkeldere applicaties, eventueel met Pdf's die getoond moeten worden e.d., moet er nog een deel aan specifieke code worden bijgeschreven.

7.3 Het leadsformulier moet volledig instelbaar zijn

Over het algemeen zullen de applicaties dezelfde indeling hebben (hoewel er wel andere styling en headers zullen worden toegepast). Wat er vrijwel zeker anders is per applicatie, is het leadsformulier. Een lead is anders op een autobeurs, waar bijvoorbeeld een proefrit wordt aangeboden, dan op een golfevenement van een bank, waar bijvoorbeeld een afspraak met de private banker wordt aangeboden.

Deze grote verschillen maken het noodzakelijk dat er verschillende formulieren aangemaakt kunnen worden per applicatie. Maar dit moet wel zo generiek mogelijk worden opgelost, om terug te komen bij de vorige uitdaging met betrekking tot het genericiteit. De vraag die hierbij centraal staat, gaat, vooral over de mogelijkheid tot het creëren van verschillende formulieren op een eenduidige manier. Is het mogelijk om een formulier aan te maken zonder de broncode (grondig) te wijzigen?

Oplossing

Dit punt heeft veel innovatie en verbeteringen in zich zitten met betrekking tot de huidige generatie. Het bouwen van dezelfde functionaliteiten als de huidige generatie heeft echter voor heel wat uitdagingen gezorgd.

Als eerste moesten er behaviors en rules kunnen worden gedefinieerd en aangeroepen, waarbij elementen gekoppeld moesten worden aan deze behaviors en rules. Het definiëren was niet het probleem, maar het initialiseren van deze scripts gebeurde veelal voordat ze feitelijk waren ingeladen (aangezien dit laatste gebeurt via AJAX). Dit is uiteindelijk opgelost door gebruik te maken van het jQuery Defered object.

Een formulier moet uit meerdere pagina's kunnen bestaan. Dit is een onderdeel dat in de huidige generatie opgelost wordt door de pagina als een afzonderlijke div op te nemen die vervolgens aangesproken wordt als de gebruiker op een 'volgende' knop drukt. Echter door de werking van AngularJS, is dit een stuk moeilijker geworden. AngularJS heeft namelijk altijd maar 1 view in de pagina. Dit is gedaan om de DOM zo schoon mogelijk te houden en omdat het anders de applicatie te langzaam zou maken. Maar het maakt het lastiger om gegevens tussen pagina's te versturen, wat moet kunnen. Het moet bijvoorbeeld mogelijk zijn om op pagina 1 iets aan te vinken waardoor iets zichtbaar wordt op pagina 2. De oplossing is uiteindelijk gevonden in het bijhouden van een `state` bij een behavior. Hierover is in hoofdstuk 4 'requirements en aannames' meer over beschreven.

Om de gegevens te blijven onthouden wordt gebruik gemaakt van de data-binding die in AngularJS zit. Door na elke pagina de data in een Service te zetten en bij het laden van de pagina de data weer uit te service te halen en te koppelen aan het formulier, kunnen gegevens van de ene naar de andere pagina worden meegenomen. Op deze manier is het ook mogelijk om een formulier voor een deel voor in te vullen. Eventueel met de gegevens van de hoofdboeker van de reservering.

Om de HTML van het formulier te genereren is een generiek JavaScript object geschreven dat over een JSON object loopt. Door het gebruiken van een generiek object kan eenvoudig en reproduceerbaar formulier worden gemaakt dat eenvoudig te stijlen is.

7.4 De data moet niet voor iedereen inzichtelijk/bewerkbaar zijn

Om de bezoekers van een evenement present te melden moeten er gegevens van hen worden opgevraagd. Het betreffen NAW gegevens, gegevens die zijn toevertrouwd aan de organisatie van het evenement, die de gegevens vervolgens heeft toevertrouwd aan Iventer. Het 'kwijtraken' van deze gegevens, omdat een derde partij deze kan inzien of downloaden, is onwenselijk en moet daarom voorkomen worden.

De vragen die hier mee samen hangen liggen vooral op het gebied van authenticatie. Maar wat zijn hier goede manieren voor? Hoe kan voorkomen worden dat mensen er met de data op de devices vandoor gaan? Zijn de gebruikers zelf volledig te vertrouwen met alle functionaliteiten en gegevens in de applicatie. En zo niet, wat kan er gedaan worden om hen gewaarborgde toegang te geven?

Oplossing

Het is niet mogelijk om de applicatie op te starten zonder in te loggen. De keuze hiervoor is gemaakt om de gegevens van de reserveringen veilig te kunnen stellen. Maar om het gebruikersgemak toch hoog te houden, is er voor gekozen om de applicatie te kunnen ontgrendelen door de invoer van een gebruikersnaam. Gebruikersnamen zijn over het algemeen eenvoudiger te onthouden, gebaseerd op bekende data en worden traditiegetrouw ingevoerd in een standaard tekstveld in plaats van in een wachtwoordveld.

Echter het gebruik van een gebruikersnaam is niet (of in elk geval minder) veilig dan het gebruiken van een wachtwoord. Vandaar dat de toegang tot de admin, waar onder andere inzage is in de leads

en deze verstuurd kunnen worden naar de Iventer database, achter een wachtwoord login schuil gaat. Ook het verwijderen van een lead is gebonden aan een wachtwoord login. Dit laatste omdat leads een belangrijk onderdeel zijn van de event-app en het verwijderen van een lead een verlies in business kan betekenen.

Naast het onderscheid in gebruikersnaam en wachtwoord logins, is er ook onderscheid in wat een gebruiker mag zien/doen. Niet elke gebruiker heeft toegang tot de admin en/of tot het verwijderen van een lead. Deze functionaliteiten zijn zelfs niet eens zichtbaar als de gebruiker er geen toegang toe mag hebben. Dit zorgt voor extra beveiliging en controle met betrekking tot de data en werking van de applicatie.

Hoewel het overkomt alsof de gebruiker wel erg vaak moet inloggen om de applicatie te gebruiken, dit valt erg mee. De gebruiker zal over het algemeen alleen maar af en toe met zijn of haar gebruikersnaam hoeven in te loggen, als de tablet bijvoorbeeld in slaapstand is geraakt of als men uit de applicatie is genavigeerd. Het invoeren van het wachtwoord is pas nodig als men de data naar de API gaat versturen, maar deze actie wordt doorgaans één keer per dag gedaan. Vandaar dat de hoeveelheid gebruikersgemak die ingeleverd moet worden voor de veiligheid minimaal.

7.5 De applicatie moet eenvoudig te debuggen zijn

De applicatie is voor een groot deel dynamisch, instelbaar via JSON. Hierdoor kan het zijn dat een situatie voorkomt die niet werd verwacht. Een voorbeeld hiervan is dat er een maatwerk behavior wordt ingeladen bij een formulier, maar dat deze code niet correct is. Hoe kan deze fout worden opgespoord? Hoe kan de applicatie gedebugged worden als deze zo dynamisch is?

Oplossing

Om een stuk JavaScript te debuggen wordt doorgaans gebruik gemaakt van de `console.log()` functie. Dit stuurt de doorgegeven parameter(s) naar het console scherm van de ontwikkelaars programma's van de browsers als Google Chrome, Firefox en Internet Explorer. Op deze manier kan eenvoudig worden doorgegeven wat het resultaat van een stuk code is of wat de huidige waarde van een variabele is. De functionaliteit mist echter nog een aantal functionaliteiten om debugging en gebruik van de applicatie te vereenvoudigen.

Om aan deze punten te voldoen is er een JavaScript object `log` in het leven geroepen. Het maakt intern gebruik van het `console` object, maar bouwt hier nog iets verder op voort. Overal in de code kan de methode `log.add` aangeroepen worden die twee parameters verwacht: de boodschap en het type melding. Dit voegt de melding toe aan een array in het object. Daarnaast wordt de melding direct uitgevoerd in het console scherm. Er zijn 3 verschillende type meldingen, elk met een eigen gebruik, mate van belangrijkheid en een `console` methode:

Naam	**Belangrijkheid**	**Gebruik**	**`Console` methode**
Error	10	Bij een fout in de code die afgevangen had moeten worden of opgelost moet worden.	console.error
Warn	50	Bij een situatie die onverwacht is, zoals het retourneren van 0 rijen na een database query.	console.warn
Info	100	Alle informatie die goed is om te weten, zoals de melding dat een script is ingeladen of dat men zich nu in een bepaalde view bevindt.	console.info

Het gebruik van de `console` methoden zorgt er voor dat de applicatie op alle grote platformen te debuggen is, aangezien deze de `console` methoden ondersteunen. Daarnaast bieden deze

platformen een visuele wijze van het onderscheiden van de verschillende niveaus, bijvoorbeeld door het gebruik van een waarschuwingsicoon bij een `console.warn`.

Het concept rond de belangrijkheid komt vooral naar voren vanwege het feit dat niet altijd alle berichten gezien en gevolgd hoeven te worden. Het kan voor veel ruis zorgen bij het zoeken naar een fout, als tientallen info meldingen worden getoond. Vandaar dat het `log` object de mogelijkheid biedt om in te stellen welke meldingen getoond worden. De meldingen worden sowieso toegevoegd aan het object, maar ze worden niet direct getoond. Hoe lager het niveau wordt gezet, hoe minder er gezien wordt. Als het niveau op 60 wordt gezet, worden de info meldingen niet meer getoond en als het niveau op 0 wordt gezet, komen er überhaupt geen meldingen meer uit het `log` object.

Door dit centrale object is het mogelijk om bijvoorbeeld Chrome developer tools, in het geval van Android, de applicatie te debuggen en te testen waarom een onverwachte situatie of fout zich voordoet.

7.6 Er moet een uniforme wijze van documenteren komen

Voornamelijk het JSON stuk van de applicatie is erg dynamisch en het is de bedoeling dat dit wordt gedocumenteerd. De documentatie moet goed leesbaar zijn door (nieuwe) collega's. Ook moet de opbouw van de documentatie het toelaten om andere onderdelen van de JSON en/of applicatie op dezelfde wijze te omschrijven.

Hoe moet deze documentatie worden opgezet? Van welke techniek(en) moet gebruik gemaakt worden, HTML of JSON of XML? Hoe maak je de documentatie visueel aantrekkelijk en overzichtelijk voor gebruikers?

Oplossing

Wat betreft documenteren zijn er verschillende opties. Er zijn automatische documentatie scripts die een stuk code omzetten in documentatie of een REST service uitvragen om zo te zien welke waarden terugkomen via een bepaalde URL. Je kunt ook alles handmatig typen door bijvoorbeeld een HTML bestand te maken per documentatie. Of je kunt gaan voor een middenweg tussen dynamiek en uitschrijven, zoals een uitgeschreven XML bestand dat wordt weergegeven via een XSLT.

De laatste optie is waar wij voor gekozen hebben. Het biedt een gestructureerde manier van documentatie schrijven, de XML. En door middel van de XSLT is het zowel dynamische als visuele manier. In eerste instantie is er een mogelijkheid om documentatie te schrijven voor de JSON van het formulier.

De structuur van de XML is opgedeeld in nodes. Deze nodes staan voor de verschillende objecten van een formulier, hierbij moet gedacht worden aan de pages, rows, elements, enzovoorts. De node heeft verschillende eigenschappen, hieronder staan een aantal, met daarbij de functie:

- Id: De unieke identificatie van de node. Dit wordt ook gebruikt om binnen de visualisatie naar de node te navigeren
- Type: Het type van de node, zoals object of array
- Description: (korte) beschrijving van de node
- Example: de wijze waarop de node terugkomt in de JSON
- Result: de resulterende HTML, indien van toepassing.
- Variables: verschillende variabelen die van toepassing zijn, zoals elements bij de node row.

Deze en meer eigenschappen worden uitgelezen en via een XSLT gepresenteerd. Om de documentatie eenvoudig en aantrekkelijk te maken, is er voor gekozen om een CSS framework te

gebruiken om de gegevens te tonen. Er is gekozen voor Bootstrap, vanwege de eenvoud en het uiterlijk.

Binnen de documentatie kan er genavigeerd worden tussen de verschillende nodes. Zo kan een row elements bevatten. Door binnen de documentatie van row te klikken op elements om te gaan naar de documentatie van de elements.

Door het op deze manier te structureren is er een standaard voor het documenteren van JSON ontstaan. Uiteindelijk zullen er meer van dergelijke XML's en XSLT's gemaakt moeten worden, voor de verschillende documentaties van de applicatie zoals voor het documenteren van de configuratie JSON.

8. Evaluatie

In dit hoofdstuk is een evaluatie te vinden van het project. Deze evaluatie is opgedeeld in twee delen, een productevaluatie waarin de kwaliteit van het product wordt besproken en een procesevaluatie waarin de positieve als ontwikkelpunten worden besproken.

8.1 Productevaluatie

Aan de hand van de verschillende requirements wil ik de kwaliteit van het eindproduct meten. De kwaliteit van een product hangt namelijk af van de toepasbaarheid van het product en de overeenkomstigheid van het product met de vooropgestelde requirements.

De applicatie moet offline werken

In essentie is de applicatie offline te gebruiken. Er is een initiële connectie nodig met het internet om de configuratie voor een event-app op te halen, maar daarna is er geen internetverbinding meer nodig. Het present melden van een reservering en het werven van leads is volledig offline te doen. De dynamische formulieren worden offline gecached en de afbeeldingen e.d. worden lokaal en (dus) offline opgeslagen.

Voor het doorsturen van de gegevens is nog wel een internetverbinding nodig, aangezien de gegevens naar de API gestuurd moeten worden. Maar de basisfunctionaliteit van de applicatie is offline te gebruiken.

De applicatie moet op verschillende platformen kunnen draaien

Door het gebruik van Apache Cordova kan zeker voldaan worden aan deze requirement. De applicatie is zo opgezet dat deze werkt op zowel iOS en Android. Er is nog geen noodzaak geweest om het op Windows Phone en BlackBerry te testen, deels omdat er in de directe werkomgeving geen dergelijk toestel aanwezig is.

Als echter in de toekomst een van de secundaire platformen nodig is, kan de applicatie vrij eenvoudig voor dat platform worden gemaakt.

Gegevens voor de applicatie moeten uit de API gehaald worden

Op het moment dat de applicatie voor het eerst wordt gestart, staat er geen informatie in de applicatie. De enige bestanden die aanwezig zijn, bieden een schil voor een event-app die ingeladen kan worden.

Vanaf het moment dat er een event-app geselecteerd moet worden, komen alle gegevens uit de API. Er wordt opgehaald welke event-apps er beschikbaar zijn en wat voor configuratie zij hebben. Vervolgens wordt bij het installeren van een event-app alle gegevens betreffende de beschikbare reserveringen, de formulieren en de bestanden (zoals afbeeldingen) opgehaald vanuit de API.

Alle communicatie met Iventer moet veilig gebeuren

De verbinding met de Iventer API is beveiligd door SSL. Op dit moment is er verder nog geen restrictie aan het gebruik, maar deze komt in de nabije toekomst.

Scannen van een barcode moet ondersteund zijn

De barcode kan gescand, gelezen en geïnterpreteerd worden. Het wordt vervolgens gebruikt om een reservering op te zoeken. Het kan altijd nog zijn dat een reservering niet voorkomt met die (bar-)code, dus er zal sowieso eerst worden gecontroleerd of de reservering voorkomt

Meerdere event-apps binnen één applicatie

Er is bij het ontwikkelen van de applicatie vanaf het begin rekening gehouden met het feit dat er configuratie is die een event-app definieert en deze in de applicatie kan laten draaien. Het is mogelijk om in de applicatie een JSON string in te lezen vanuit de API die vervolgens wordt geïnterpreteerd door de applicatie om zo de nodige gegevens voor een event-app aan te maken. Hierbij moet gedacht worden aan het aanmaken en vullen van de verschillende databasetabellen, het aanmaken van de nodige formulieren en het laden van de afbeeldingen die in de event-app gebruikt worden.

De applicatie moet eenvoudig te debuggen zijn

Zowel iOS en Android hebben manieren om een WebView applicatie, wat Apache Cordova applicaties altijd zijn, te debuggen door de applicatie aan een PC/Mac te koppelen. In de applicatie is een object opgenomen voor het loggen van de handelingen in de applicatie. Met behulp van dit object en de debug mogelijkheden van de WebView is het mogelijk om de applicatie te debuggen in het geval van fouten.

De applicatie is bedoeld voor tablets

Er is bij het ontwikkelen van de applicatie alleen rekening gehouden met tablets. Bij het gebruik van Ionic als CSS framework is dan ook gekozen voor meerdere knoppen op een rij, zoals bij het detailscherm van de reservering. Bij een tablet is er namelijk meer ruimte en kunnen er dus meerdere elementen bij elkaar worden gezet.

Conclusie

Zoals hierboven te lezen is, zijn de verschillende requirements verwerkt in het product. -------

8.2 Procesevaluatie

Tijdens het project zijn er beslissingen gemaakt en ervaringen opgedaan. Sommige hiervan heb ik als zeer positief ervaren maar ik zou andere dingen wel anders hebben gedaan. In deze paragraaf behandel ik verschillende beslissingen en ervaringen en hoe ik deze in een volgend project zou integreren.

Het werken met sprints werkt erg goed

Wat ik erg goed vond werken tijdens het project was het werken met sprints. Deze manier van werken geeft een goede manier om gestructureerd een opdracht te maken. Het iteratieve karakter van sprints geeft de mogelijkheid om stuk voor stuk een product op te bouwen.

Door het opbreken van de opdracht in verschillende sprints van twee weken heb ik tijdens die weken kunnen focussen op de onderdelen die in die sprint centraal stonden. Dit zorgt er voor dat er meer focus is op deze onderdelen, wat zorgt voor een beter ontwerp en beter product.

Eerder beginnen met en continue testen

Aan het einde van de opdracht ben ik begonnen met het schrijven van (unit) tests. Dit vooral omdat ik er tot dan toe geen tijd voor had genomen en deze tijd voornamelijk in het bouwen van het systeem had gestopt. Echter dit zorgde ervoor dat het schrijven van tests lastiger was dan als ik het eerder zou hebben geïntegreerd.

Wat wel belangrijk is om dan te bedenken, is dat de tijd die ik nu in de applicatie heb zitten, anders verdeeld zou zijn. Het is daardoor niet zeker of ik even ver zou zijn gekomen, als ik ook overal tests voor zou hebben moeten schrijven.

Opdrachtgever betrekken bij ontwikkeling is erg krachtig

Wat ik als zeer prettig heb ervaren is het feit dat de opdrachtgever op dezelfde verdieping zit als ik. Het is eenvoudig om snel even binnen te lopen en een verhelderende vraag te stellen. Het wordt zelfs erg op prijs gesteld, want het doet niemand goed om een (lange) tijd met een vraag rond te lopen.

Wat ook goed heeft gewerkt in deze samenwerking, is het tonen van de applicatie aan de opdrachtgever. Zo heeft hij een gevoel gekregen bij de staat van de applicatie en de werking ervan.

Regelmatiger Cordova builds maken

Een van de requirements is dat de applicatie op een mobiel device moet draaien. Er is aan het begin van het bouwen een build gemaakt en toen enkele weken niet. Dit heeft geen verstrekkende gevolgen gehad voor het project, maar ik kan mij goed voorstellen dat het in een toekomstig project belangrijk kan zijn om te zien hoe de applicatie draait en werkt op het uiteindelijke apparaat.

De Google Chrome browser is representatief voor een device wat betreft de functionaliteiten en ondersteunde technieken. Maar het is niet hetzelfde als het testen op een tablet. Een goed voorbeeld hiervan is het invullen van een formulier. Op de PC is het eenvoudig, vanwege het toetsenbord. Maar het toetsenbord op een tablet is lastiger om te bedienen, zeker als een van de handen gebruikt wordt om de tablet vast te houden waardoor slechts één hand beschikbaar is om te typen.

Nogmaals, in dit project heb ik er geen nadelen van ondervonden om een aantal weken geen build te maken, dit punt dient vooral als toekomstige referentie.

9. Bijlage

In dit hoofdstuk zijn de verschillende bijlagen opgenomen die uitleg bieden over de gebruikte termen ofwel het document uitbreiden.

9.1 Gebruikte termen

Applicatie

Dit is een term die veel is voorkomt in dit verslag. Met applicatie wordt de Apache Cordova build van Iventer bedoeld, oftewel de applicatie die op het device staat, het icoon dat te zien is in de lijst.

Event-app

Dit een term die veel lijkt op de vorige wat betreft naamgeving, maar een andere lading dekt. Een event-app betreft de set aan instellingen voor een evenement. Een event-app is uiteindelijk gekoppeld aan een klant als de Rabobank of BMW. En deze event-app draait uiteindelijk binnen de applicatie.

Device

Hiermee wordt het fysieke product bedoeld waarmee de eindgebruiker de applicatie mee gebruikt. In de eerste instantie is de applicatie gemaakt om gebruikt te worden met een tablet. Overal waar device staat kan dus ook tablet worden gelezen. Maar in de toekomst kunnen ook telefoons of laptops (beter) worden ondersteund.

Reservering

Dit is een aanspraak op een set aan kaarten die is gedaan door een relatie die uit het Iventer systeem komt. De kaarten zijn gekoppeld aan tijdsloten die gedefinieerd zijn in het Iventer systeem.

Relatie

Een relatie is de hoofdboeker van de reservering. De hoofdboeker is de enige persoon die met zijn NAW gegevens in de applicatie staat (indien zo ingesteld).

Gebruiker

Een gebruiker is een medewerker van de partij die de applicatie gebruikt om reserveringen present te melden of om leads te werven.

9.2 Verklarende woordenlijst

API

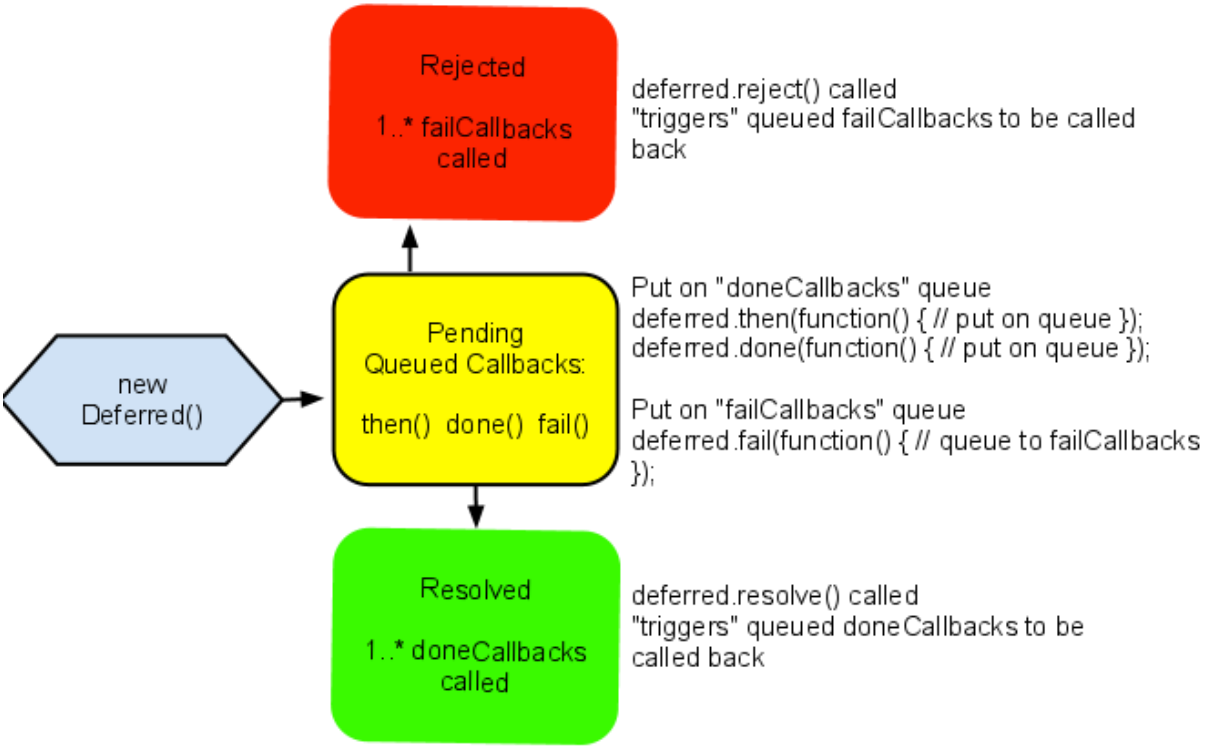
Application Programming Interface, een verzameling definities op basis waarvan een computerprogramma kan communiceren met een ander programma of onderdeel.[24]

Middleware

Omvat de systeemsoftware die de informatie-uitwisseling regelt tussen de cliënt-software en de software die de bedrijfsgegevens beheert.[25]

[24] http://nl.wikipedia.org/w/index.php?title=Application_programming_interface&oldid=40848525
[25] http://nl.wikipedia.org/w/index.php?title=Middleware&oldid=35464816

9.3 schematische weergave jQuery Deferred[26]



9.4 Overige technieken en software

Grunt[27]

De pitch voor Grunt is dat het een manier is om verschillende taken die een (web)developer geregeld uitvoert, automatisch te kunnen uitvoeren. De taken waaraan gerefereerd wordt zijn bijvoorbeeld het verkleinen van de JavaScript, HTML en CSS code en het optimaliseren van afbeeldingen. Deze taken kunnen handmatig uitgevoerd enkele minuten in beslag nemen, maar door middel van een simpele Command Line Interface (CLI) command kan er een sequentie aan opdrachten worden verzonden.

De verschillende taken die uitgevoerd worden binnen dit project zijn bedoeld om de laadtijden zo optimaal mogelijk te maken. Voornamelijk de hoeveelheid aan scripts is hierbij belangrijk. Vandaar dat alle javascript bestanden van Iventer in één bestand worden gezet. Daarnaast worden alle bestanden die afkomstig zijn van AngularJS, jQuery en Ionic in een tweede bestand gezet. Dit komt vanwege de verschillende verkleinmethoden op het Iventer specifieke script, deze hebben nadelige gevolgen voor de library scripts.

Naast het verkleinen en samenvoegen van de JavaScript bestanden, worden de verschillende CSS bestanden ook samengevoegd en verkleind.

[26] http://colonelpanic.net/2011/11/jquery-deferred-objects/

[27] http://gruntjs.com/

Als laatste optimalisatie worden alle HTML bestanden geminimaliseerd door alle witruimte uit de pagina te halen.

Vervolgens worden al deze geoptimaliseerde bestanden gekopieerd naar de map waarin het Cordova project staat. Deze hoeven dus niet met de hand te worden overgezet, er kan direct een nieuwe build gemaakt worden.

9.5 Voorbeeld JSON code

Deze code is bedoeld als voorbeeld van de JSON code die gebruikt binnen de applicatie, om formulieren aan te maken. In dit geval zou het de situatie zoals geschetst in figuur 4.4 kunnen genereren, twee radiobuttons met een label en een behavior met daaronder een verborgen e-mail veld.

```
{
  “pages”: [
    {
      “title”: “eerste pagina”
      “rows”: [
        {
          “elements”: [
            {
              “type”: “fieldset”,
              “parts”: {
                “label”: “Wilt u contact opnemen?”
              }
              “behaviors”: [
                {
                  “type”: “alterTarget”,
                  “script”: “/behaviors/alterTarget.js”,
                  “parameters”: {
                    “triggers”: “dom::antwoord1”
                    “targets”: “dom::email”,
                    “alter”: {
                      “positive” : {
                        “dom”: {
                          “style”: “display:block;”
                        }
                      }
                      “negative” : {
                        “dom”: {
                          “style”: “display:none;”
                        }
                      }
                    }
                  }
                },
              ]
              “elements”: [
                {
                  “type”: “radio”
                  “attributes”: {
                    “id”: “antwoord1”,
                    “name”: “contactOpnemen”,
                    “value”: “ja”
                  },
                  “parts”: {
                    “content”: “ik wil contact opnemen”
```

```
                }
              },

              {
                “type”: “radio”
                “attributes”: {
                  “id”: “antwoord2”,
                  “name”: “contactOpnemen”,
                  “value”: “nee”
                },
                “parts”: {
                  “content”: “ik wil geen contact opnemen”
                }
              }
            ]
          }
        ]
      },
      {
        “elements”: [
          {
            “type”: “email”
            “attributes”: {
              “id”: “email”,
              “name”: “email”,
              “style: “display:none”
            },
            “parts”: {
              “label”: “E-mailadres”
            },
            “rules”: [
               {
                  "type": "isEmail",
                  "script": "/rules/isEmail.js",
                  "id": "emailIsEmail",
                  "elements": "email"
               }
            ]
          }
        ]
      }
    ]
  }
}
```

Plan van aanpak

Mobiele applicatie voor evenementensysteem

Bram Slob
11050365
Informatica deeltijd

Opdrachtomschrijving:

• Bedrijf / organisatie

Faceworks is een bedrijf dat zich richt op het leveren van (online) communicatiemiddelen in de vorm van websites, webshops, webapplicaties en (mobiele) apps. De klanten variëren in grootte, met als grootste klanten BMW, MINI en de Rabobank en aan de andere kant kleinere klanten zoals verschillende golfclubs verspreid door het land en enkele ZZP'ers. Het bedrijf bestaat uit 9 man, een commerciële man, een ontwerper en zeven developers. De taken zijn verdeeld onder de werknemers op basis van kennis en ervaring. Er wordt hoofdzakelijk in PHP geprogrammeerd, waarbij technieken als HTML, CSS, Javascript en SQL natuurlijk ook essentieel zijn.

• Probleembeschrijving

Faceworks heeft een systeem voor het aanmaken en organiseren van evenementen. Dit omvat zowel de registratie van mogelijke bezoekers en het toewijzen van de bezoekers aan het evenement. Er is echter nog geen mogelijkheid om in het systeem op te nemen welke bezoekers er daadwerkelijk zijn geweest en of ze bijvoorbeeld naar alle workshops zijn gegaan waarvoor ze zijn ingeschreven. Dit is iets wat voor de klanten van Faceworks van groot belang is. Daarnaast is er een wens om leads te kunnen verzamelen op het evenement, om zo de klantenbasis te vergroten en eventuele nieuwe werk te generen. Voorbeelden van leads zijn het maken van een afspraak voor een proefrit of het aanmelden voor een ander evenement.

Op dit moment moeten tijdens een evenementen papieren lijsten worden ingevuld om de presentie bij te houden of er moet een lijst worden bijgewerkt in Excel om een bezoeker aan te merken als 'aanwezig'. Het genereren van leads gaat nu volgens eenzelfde principe, met papieren lijsten die ingevuld moeten worden en/of Excel lijsten die moeten worden bijgewerkt. Daarnaast moeten fysieke visitekaartjes worden verzameld en verwerkt worden.
Beide handelingen kosten erg veel tijd en zijn foutgevoelig, iets wat voorkomen moet worden.

Als oplossing wil Faceworks een mobiele applicatie, geïnstalleerd op tablets die door medewerkers worden gebruikt, waarmee bezoekers kunnen worden registreert doormiddel van het scannen van een barcode die de bezoeker via e-mail heeft ontvangen. De applicatie moet een keer worden gebouwd en vervolgens voor alle evenementen ingezet kunnen worden. Er moet zonder internet gewerkt kunnen worden en alle mobiele apparaten moeten een up-to-date lijst hebben met de aanwezige bezoekers. De applicatie moet ook leads kunnen verwerven en verwerken, via eenvoudig te bedienen formulieren die per soort lead verschillend moeten kunnen zijn. De applicatie moet alle informatie die nodig is voor en tijdens een event uit het evenementensysteem halen en naderhand de aanwezigheidsdata en leads weer naar het evenementensysteem sturen.

Hier worden de problemen (in de huidige situatie) beschreven, die door het uitvoeren van de opdracht zullen verminderen, zo niet verdwijnen. Let op dat je hier alleen problemen schetst, dus geen oplossing.

• Doelstelling van de opdracht

Aan het einde van de afstudeeropdracht heeft Faceworks een mobiele applicatie, waarmee op een evenement presentie kan worden bijgehouden en leads kunnen worden verworven.

Afbakening opdracht:

- **Mix front- en backend werk**
  De opdracht is voor een gedeelte gericht op het bouwen van een applicatie, op basis van HTML, CSS en Javascript, in Apache Cordova.
  Daarnaast zal er ook werk verzet moeten worden aan de serverkant, omdat er onderdelen bij de bestaande API's moeten worden gebouwd om een mobiele applicatie te ondersteunen.
- **In eerste instantie gericht op tablets**
  De applicatie is gericht op tablets, het wordt niet op telefoons geïnstalleerd. Daarnaast is het ook grotendeels op een laptop te gebruiken, aangezien het opgebouwd is met HTML, CSS en Javascript. Het kan echter wel zijn dat onderdelen van de applicatie niet of minder goed werken (zoals ondersteuning voor de camera) omdat die worden geregeld via Apache Cordova.

Randvoorwaarden:

- **Testmogelijkheden**
  Niet alleen moet er getest kunnen worden op de computer waarop wordt ontwikkeld, er moet ook getest kunnen worden op een tablet.
- **Duidelijke en tijdige feedback**
  Aangezien dit project overeenkomsten vertoond met Extreme Programming, is het belangrijk dat de opdrachtgever onderdeel is van het ontwikkelproces.

Risicofactoren:

Er zijn verschillende risicofactoren te bedenken bij dit project. Hieronder zijn een aantal ervan opgenomen met daarbij een beschrijving van het risico en hoe dit vermeden en/of opgelost kan worden.

Het herkennen van de mogelijke risico's gaat niet automatisch en vereist een constante controle op de planning, de requirements en de algemene voortgang. Dit wordt enigszins vereenvoudigt door de globale opzet van het project, omdat ik namelijk per sprint van 2 weken opnieuw inventariseer hoe ver ik ben en wat ik de volgende sprint precies ga doen.

- **Uitlopen**
  Het grote voordeel van een iteratieve Systeemontwikkelmethode is dat gewijzigde requirements eenvoudiger zijn toe te passen op het product. Dit neemt echter wel met zich mee dat door de nieuwe situatie, de doorlooptijd van het project wordt vergroot. Dit kan zorgen voor uitlopen, wat niet gewenst is.
  Door constant zicht te houden op de sprintplanning(en) en de globale planning moet een dergelijke situatie geïdentificeerd en behandeld worden. Dit is een probleem wat samen met de opdrachtgever besproken moet worden, aangezien het kan betekenen dat het project later wordt opgeleverd of dat er onderdelen van de applicatie worden geschrapt.
- **Werk overnieuw doen**
  Een andere bijwerking van het wijzigen van de requirements is dat er een onderdeel dat al is gemaakt, aangepast moet worden om aan de nieuwe requirements te voldoen.

  Dit is een risico dat vrijwel niet valt te vermijden. Wanneer een requirement wijzigt zal code die daar op van toepassing is, moeten wijzigen. Wel kan er met de opdrachtgever worden overlegd over het feit dat de wijziging impact heeft op de bestaande code.

- **Niet voldoen aan de requirements**
  Of er tijdens het ontwikkelen requirements wijzigen of niet, er bestaat altijd het risico dat het eindproduct niet voldoet aan de requirements die de opdrachtgever heeft gesteld.

  Door veel overleg met de opdrachtgever en veel (tussentijdse) oplevering van de applicatie kan dit voor het grootste gedeelte worden ondervangen. Doormiddel van een Acceptatietest enkele tijd voor de deadline kunnen eventuele laatste (gemiste) requirements worden ingebouwd.

Projectorganisatie:

Deze opdracht zal ik vervullen als medewerker in de organisatie. Ik ben al ruim een half jaar werkzaam binnen de organisatie waardoor de relatie met de organisatie is gevestigd. Dit zorgt voor enkele afspraken en situaties die zich bij een nieuwe werkomgeving wellicht minder snel zouden voordoen.

Zo zal ik niet 100% van mijn werktijd kunnen besteden aan de opdracht. Er lopen ook andere projecten en er kunnen werkzaamheden tussendoor komen die mijn aandacht nodig hebben. Er wordt gestreefd naar minimaal 24 effectieve uren per week voor mijn opdracht. Dit kan echter per week verschillen, afhankelijk van werkdruk en overige activiteiten.

In dit project zal ik vasthouden aan verschillende onderdelen van de systeemontwikkelmethode Extreme Progamming. Dit doe ik vooral omdat dit flexibiliteit biedt aan de ontwikkelaar en de opdrachtgever. Daarnaast is continue releases een onderdeel van de methode, wat de kwaliteit van het uiteindelijke product ten goede kan komen, omdat zo eventuele onvoorziene features of problemen ondervangen kunnen worden. Echter zijn niet alle onderdelen van toepassing op dit project. Hieronder vindt u een overzicht van de verschillende onderdelen die terugkomen in dit project.

- **Iteratief werken**
  Dit project zal opgedeeld zijn in sprints van twee weken. Aan het begin van elke sprint zal er een korte planning gemaakt worden voor wat er in die sprint gedaan moet worden. De sprints hebben allemaal een gelijke indeling, daarover meer bij het onderdeel planning. De inhoud van de sprints staat nog niet vast, deze zullen worden vastgesteld aan het begin van elke sprint op basis van de vastgestelde en/of bijgewerkte user stories.
- **Klant is onderdeel van het ontwikkelteam**
  De klant van dit systeem is mijn huidige werkgever, dat maakt dit onderdeel van Extreme Programming eenvoudig te implementeren. De wensen van de opdrachtgever, mochten deze wijzigen tijdens het ontwikkelproces, zijn eenvoudig te bespreken aangezien de opdrachtgever in hetzelfde pand zit.
- **User Stories**
  Voor het ontwikkelen moet er uitgegaan worden van realistische situaties waarin het product gebruikt gaat worden. Er worden user stories gemaakt die beschrijven het verwachtte gebruik is van het product, aan de hand waarvan de software geschreven moet worden.
- **Een zo simpel mogelijk ontwerp**
  Door een simpel ontwerp te hanteren is het systeem beter te onderhouden en te refactoren.

- **Codeerstandaard**
  Om de onderhoudbaarheid en overdraagbaarheid van het product te garanderen wordt gebruik gemaakt van een codeerstandaard. In dit geval zal dat de codeerstandaard zijn van mijn werkgever.
- **Continue refactoring**
  De wensen van de opdrachtgever kunnen wijzigen, wat nieuwe of gewijzigde features tot gevolg kan hebben. Dit vereist vervolgens dat de code wordt aangepast of aangevuld.
- **Continue integratie**
  Software is een som der delen. Door de verschillende delen van de code bij elkaar te zetten en een build te maken, kan gezien worden of het systeem nog doet wat gewenst is.
- **Regelmatige releases**
  Door regelmatig een oplevermoment te hebben, kan er door de opdrachtgever goed gezien worden hoe het project vordert. Daarnaast bied het voor de programmeur een goed inzicht in de huidige status en kan er beter gepland worden naar de toekomst.

Ik zal dit project voornamelijk zelfstandig doen, zeker qua programmeren. De opdrachtgever zit niet ver bij mij vandaan en ik zal zeker af en toe langslopen om inzage te geven in mijn voortgang, maar de productie van de code ligt in mijn handen. De opdrachtgever zal voornamelijk adviserende en corrigerende rollen hebben. Dat laatste omdat er in dit project verschillende elementen van Extreme Programming naar voren komen, zo ook het betrekken van de opdrachtgever bij het ontwikkelen.

Tijdens het project zullen er meerdere documenten gemaakt worden (user stories, planningen, verslagen, etc.). Deze zullen worden opgeslagen op een Dropbox account om altijd toegang te hebben tot alle documenten.

De code die geschreven wordt, zal elke dag worden geüpdatet op een versiebeheerserver. Zo worden versies bijgehouden, is er altijd een back-up en kan er vanaf meerdere locaties worden gewerkt.

Wijze van rapporteren:

Dit project bestaat uit een aantal sprints, elk van twee weken. Aan het einde van elke sprint wordt een rapport opgeleverd met daarin een globaal overzicht van de verrichtte werkzaamheden, de geslaagde (en eventuele mislukte) testen en eventuele belangrijke mededelingen/vragen die naar boven zijn gekomen tijdens de sprint.

Benodigde mensen/ middelen:

Mensen

Opdrachtgever

Middelen

PC om op te ontwikkelen op het werk en thuis
Software om code mee te schrijven
Instantie van SVN
Installatie van Cordova
Apparaat met XCode erop geïnstalleerd
IOS en Android tablet

Planning:

De planning van dit project is opgedeeld in twee delen: Het plannen en het uitvoeren. De planningsfase is nodig om het project op een goede en gestructureerde manier te beginnen. De uitvoerende fase bestaat uit sprints van twee weken, volgens een vaste indeling.

De indeling is als volgt:

Activiteit / Dag	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Planning												
Development												
Testen												
Integratie												
Oplevering												
Rapportage												

Hierboven is te zien dat er per sprint verschillende activiteiten zijn die al dan niet naast elkaar lopen. Hieronder een kleine uiteenzetting van de verschillende activiteiten.

- Planning
  Elke sprint heeft dezelfde indeling maar niet dezelfde inhoud. Door planningsmomenten in te bouwen kan de planning worden bijgewerkt om een accurater beeld te hebben van de situatie. Deze momenten zijn na een oplevering en rapportage gepland, zodat de vernieuwde inzichten kunnen worden verwerkt in de planning
- Development
  Continue door het project zal er development plaatsvinden om de het doel te bereiken
- Testen
  Naast de development zal de gebouwde code ook getest worden
- Integratie
  Sowieso aan het einde van elke dag zal er een nieuwe build gemaakt worden van de applicatie.
- Oplevering
  Eens per (werk)week zal er een oplevering van de applicatie zijn aan de opdrachtgever.
- Rapportage
  Naar aanleiding van de oplevering van de oplevering aan de opdrachtgever, diens feedback en het verzette werk die week, zal er eventueel gerapporteerd worden over de huidige situatie en het verloop van het project.

Activiteit / week	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Plan van Aanpak																	
User stories																	
Iteratief ontwikkelen			1	1	2	2	3	3	4	4	5	5	6	6	7	7	
Afronden																	

Hierboven is de globale planning opgenomen voor het project. Om de start van het project soepel te laten verlopen, zal er eerst een week gereserveerd worden voor het opzetten en uitschrijven van het plan van aanpak. Daarna zal een week zijn gereserveerd om de user stories helder te formuleren en te accorderen met de opdrachtgever. De user stories zijn belangrijk omdat ze de basis vormen voor de iteraties (volgens Extreme Programming). Daarna zullen de verschillende iteraties doorlopen worden om het doel te bereiken van het project. Aan het einde is een week gereserveerd om de laatste oplevering te doen, de laatste documentatie te schrijven en alles in te leveren.

Uitloop op deze planning is mogelijk en zullen met de opdrachtgever worden overlegd.

Hoewel per sprint bekeken wordt wat er gedaan gaat worden, kan van te voren al enigszins worden aangegeven wat er de eerste paar sprints verwacht kan worden.

Sprint	werkzaamheden
1	Opbouwen eerste formulier Opbouwen lijst met gebruikers Uitbouwen en verbeteren huidige framework
2	Controleren + verzenden van formulier Uitbouwen en verbeteren huidige framework

Beschrijving mijlpaalproducten:

- **Plan van aanpak**
  Het plan van aanpak zal inhouden wat er moet gebeuren in het project, wat het resultaat is en hoe het project georganiseerd moet worden.

  Het biedt zowel voor de ontwikkelaar als de opdrachtgever houvast wat betreft afspraken en het biedt kaders voor het project.
- **User stories**
  De user stories zijn de basis voor de iteraties en zullen worden gebruikt om de iteraties te verdelen. Daarnaast dienen ze om zowel de ontwikkelaar als de opdrachtgever inzicht te geven in de gewenste functionaliteit en bieden ze een basis voor uit te voeren testen.

- **Per iteratie**:
  Per Iteratie zullen ook documenten opgeleverd/gemaakt moeten worden, die qua inhoud gebonden zijn aan de iteratie waar ze bij horen.
    - **Iteratieplanning**
      Elke iteratie zal anders zijn qua inhoud en dus zal er ook een aparte planning gedaan gemaakt moeten worden. In deze planning zullen de voorgenomen werkzaamheden zijn opgenomen en eventuele werkzaamheden die zijn blijven liggen van eerdere iteratie(s) zullen ook zo worden aangemerkt.

      De iteratieplanningen hebben vooral als doel om de werkzaamheden doorzichtig te houden en de voortgang inzichtelijk te houden voor de opdrachtgever. En omdat er per iteratie wordt gepland, is de planning flexibel en kan deze meebuigen met de actuele voortgang en (werk)druk.
    - **Vernieuwde codebase**
      Logischerwijs zal de codebase vernieuwd zijn per iteratie en door te tijd heen groeien/actualiseren.
    - **Testrapport**
      Elke iteratie wordt er getest. Het testrapport zal verschillende testen documenteren en hierover rapporteren.
- **Uiteindelijke applicatie**
  Aan het einde van het project is er een applicatie, zoals omschreven in de doelstelling, die opgeleverd wordt aan de opdrachtgever.
- **Afrondend testrapport**
  Aan het einde van project is er een acceptatietest door de opdrachtgever, in navolging daarop zal een afrondend testrapport worden opgesteld.

User stories

Mobiele applicatie voor evenementensysteem

Bram Slob
11050365
Informatica deeltijd

Revisiehistorie

Versie	Revisiedatum	Wijzigingen
001	03-12-2013	Initiële versie
002	05-06-2014	Uiteindelijke versie

Distribution

Dit document is bedoeld voor:

Naam	Functie	Datum van uitgave	Versie
Michiel de Haan	Bedrijfsmentor	03-12-2013	001
Gerard Mijnarends	Begeleidend examinator	03-12-2013	001
Michiel de Haan	Bedrijfsmentor	05-06-2014	002
Gerard Mijnarends	Begeleidend examinator	05-06-2014	002
Ben Kuiper	Tweede examinator	05-06-2014	002

10. Inleiding

Het document dat u voor u hebt liggen user stories die horen bij het afstudeertraject van Bram Slob op de mobiele Iventer applicatie. De user stories beschrijven het gewenste gedrag van de applicatie en geven hier acceptatiecriteria aan mee.

In hoofdstuk twee zijn de verschillende user stories te vinden, op gebouwd volgens een vast format. Eerst wordt de algemene user story opgezet, waarna sub user stories worden opgezet die de verschillende paden van de algemene user story ondersteunen.

In hoofdstuk drie zijn de bijlagen te vinden. Deze bijlagen zijn de verschillende flowcharts die de verschillende user stories schematisch weergeven.

11. User stories

Hieronder staan de verschillende user stories opgenomen van de applicatie. Deze zijn ingedeeld per hoofdonderdeel. Elk hoofdonderdeel heeft minstens 1 hoofd user story, die vervolgens wordt verdeeld in meerdere detail user stories. De detail user stories zullen onder andere dienen om de happy flow en diens uitzonderingen duidelijk uiteen te zetten. Per detail user story zijn ook acceptatie criteria opgenomen.

11.1 Meerdere event-apps inladen

> Als een gebruiker van de applicatie, wil ik meerdere event-apps kunnen inladen zodat ik de applicatie voor meerdere evenementen kan gebruiken.

Hierboven staat een van de belangrijke onderdelen van de applicatie, de mogelijkheid om verschillende event-apps in de applicatie te hebben staan en de mogelijkheid om hiertussen te kunnen wisselen. Hieronder staan de verschillende routes die genomen kunnen worden om de user story hierboven uit te voeren. Voor een schematische uitwerking, zie Bijlage 1.

11.1.1 Er zijn nog geen event-apps opgeslagen

> Als een gebruiker van de applicatie, wil ik een event-app kunnen opslaan zodat ik meerdere event-apps kan inladen zodat ik de applicatie voor meerdere evenementen kan gebruiken.

De bezoekers zien een lege lijst met event-apps en klikken op een knop om de keuze voor event-apps gepresenteerd te krijgen. Deze event-apps moeten uit de Iventer API komen. Na een event-app geselecteerd te hebben moet de data ervan in de database worden geschreven en de naam als item in de lijst worden opgenomen.

Acceptatiecriteria

- Er moet een lijst zijn waar de event-apps in worden getoond
- Er moet een knop zijn om event-apps op te kunnen slaan
- Er moeten event-apps uit de API van Iventer gehaald kunnen worden
- De event-app en zijn configuratie moet in de lokale database opgeslagen worden.

11.1.2 Er wordt een event-app geselecteerd

> Als een gebruiker van de applicatie, wil ik een bezoeker present kunnen melden zodat ik de presentie van het evenement kan meten.

De bezoekers zien een lijst met event-apps en klikken op een van de event-apps om zo de details ervan te zien.

Acceptatiecriteria

- Er moet een detailscherm zijn voor een event-app
- Er moet een koppeling komen tussen een event-app en zijn detailscherm

11.1.3 Er wordt een event-app geïnstalleerd

> Als een gebruiker van de applicatie, wil ik een bezoeker present kunnen melden zodat ik de presentie van het evenement kan meten.

De bezoekers zien een detailscherm van een event-app en heeft hierbij de optie om deze te installeren. Het installeren houdt in dat de configuratie die is opgeslagen in de lokale database wordt uitgevoerd en elementen zoals databases worden aangemaakt.
Hierna kan de gebruiker eventueel naar de event-app toe, of terug naar de event-app lijst.

Acceptatiecriteria

- Er moeten knoppen komen om:
- Een event-app te installeren
- Terug te gaan naar de event-app lijst
- Naar de event-app te gaan
- Bij het klikken op ‘installeren’ moeten alle onderdelen die in de configuratie beschreven staan worden uitgevoerd/ingeladen/aangemaakt.

11.2 Presentie opnemen

> Als een gebruiker van de applicatie, wil ik een bezoeker present kunnen melden zodat ik de presentie van het evenement kan meten.

Hierboven staat een van de belangrijke onderdelen van de applicatie, registreren wie er allemaal op het evenement zijn (geweest). In de user story hierboven is nog geen rekening gehouden met hoe de gebruiker iemand present kan melden. Hieronder staan de verschillende routes die genomen kunnen worden om de user story hierboven uit te voeren. Voor een schematische uitwerking, zie Bijlage 1.

11.2.1 De bezoeker heeft een correcte code om te scannen

> Als een gebruiker van de applicatie, wil ik een bezoeker present kunnen melden, door zijn barcode te scannen, zodat ik de presentie van het evenement kan meten.

De bezoekers krijgen van te voren een e-mail met daarin een (bar)code die ze mee moeten nemen. Deze code is aan hen gebonden en zodra deze gescand is kan de bezoeker als present worden gemeld

Acceptatiecriteria

- Er moet een code gescand kunnen worden
- Een code moet gekoppeld zijn één bezoeker
- Het scannen moet de juiste bezoeker naar voren brengen
- De bezoeker moet met één druk op de knop op present gezet kunnen worden
- Je moet weg kunnen klikken, als de code niet juist blijkt te zijn

11.2.2 De bezoeker heeft een incorrecte code om te scannen

> Als een gebruiker van de applicatie, wil ik een bezoeker present kunnen melden, door hem in het systeem op te zoeken, zodat ik de presentie van het evenement kan meten.

Het kan om verschillende redenen mogelijk zijn dat een code als incorrect wordt gezien, het kan zijn dat de gestuurde code incorrect is, dat er problemen waren met het printen van de code of dat de scanner het niet goed doet. Er moet een secundaire manier zijn om gebruikers present te kunnen melden, door de gebruiker in het systeem op te zoeken door middel van bijvoorbeeld postcode.

Acceptatiecriteria

- Er moet gezocht kunnen worden op een of meerdere datapunten (bijvoorbeeld postcode)
- De bezoeker moet met alle beschikbare data in scherm komen
- De bezoeker moet met één druk op de knop op present gezet kunnen worden
- Je moet weg kunnen klikken, als het niet de juiste gebruiker blijkt te zijn

11.2.3 De bezoeker heeft geen code om te scannen

> Als een gebruiker van de applicatie, wil ik een bezoeker present kunnen melden, door hem als nieuwe bezoeker in te voeren, zodat ik de presentie van het evenement kan meten.

Mocht de gebruiker via het zoeken niet gevonden kunnen worden of betreft het een bezoeker die zich niet had aangemeld, dan moet er een mogelijkheid zijn om de bezoeker in te voeren als nieuwe bezoeker.

Acceptatiecriteria

- De minimale verplichtte velden van een bezoeker moeten invulbaar zijn
- De nieuwe bezoeker moet aan beschikbare tijdsloten kunnen worden gehangen
- Als de gebruiker toch al bestaat, moet hier op zijn minst melding van worden gemaakt.

11.3 Leads werven

> Als een gebruiker van de applicatie, wil ik een lead kunnen werven op een evenement, zodat daar na het evenement acties op kunnen worden ordernomen

Hierboven staat nog een van de belangrijke onderdelen van de applicatie, leads genereren. In de user story hierboven is geen rekening gehouden met hoe de lead moet worden verworven en onder wie. Hieronder staan de verschillende routes die genomen kunnen worden om de user story hierboven uit te voeren. Voor een schematische uitwerking, zie Bijlage 2.

11.3.1 De persoon stemt in met lead werving

> Als een gebruiker van de applicatie, wil ik voor een persoon een lead kunnen werven op een evenement, zodat daar na het evenement acties op kunnen worden ordernomen.

Naast het bijhouden van de presentie moet er ook een lead geworven kunnen worden. Dit gebeurt door te vragen aan een voorbijganger of het lead-formulier tezamen doorlopen kan worden. Tijdens het werven van de lead is het ook gewenst dat er meerder foto's gemaakt kunnen worden van bijvoorbeeld een visitekaartje, zodat deze later gebruikt kan worden in het vervolgproces.

Acceptatiecriteria

- Er moet een lead-specifiek formulier gegenereerd worden, op basis van mapping data uit de REST API.
- Er moeten meerdere foto's gemaakt, opgeslagen en meegestuurd kunnen worden via de REST API.
- Alle gegevens moeten opgeslagen en verstuurd worden via de REST API.

11.3.2 De persoon is bekend

> Als een gebruiker van de applicatie, wil ik voor een geregistreerde bezoeker een lead kunnen werven op een evenement, zodat daar na het evenement acties op kunnen worden ordernomen.

Het is mogelijk dat er meer mensen op het evenement zijn dan dat er in het systeem staan, bijvoorbeeld omdat er meerdere organisatoren zijn. As de persoon al in het systeem staat, moet het mogelijk zijn om de persoon op te zoeken of zijn code te scannen. Zo kan alvast een deel van het lead-specifieke formulier automatisch worden ingevoerd.

Acceptatiecriteria

- Er moet een lead-specifiek formulier gegenereerd worden, op basis van mapping data uit de REST API.
- Er moet gezocht kunnen worden op een of meerdere datapunten (bijvoorbeeld postcode)
- Er moet een code kunnen worden gescand waarna er direct wordt gezocht of deze code bekend is.
- Als een gebruiker gevonden is via een van de zoekmethoden, moet alle beschikbare data worden ingevuld in het formulier.
- Er moeten meerdere foto's gemaakt, opgeslagen en meegestuurd kunnen worden via de REST API.
- Alle gegevens moeten opgeslagen en verstuurd worden via de REST API.

11.4 Inloggen admin gedeelte

> Als een gebruiker van de applicatie, wil ik als admin kunnen inloggen zodat ik de gegevens van het evenement op de tablet kan beheren.

Het is belangrijk om als gebruiker te kunnen inloggen, omdat zo ingezien kan worden hoeveel mensen er present zijn gemeld en hoeveel leads er zijn geworven. Deze user story is compleet genoeg om direct uit te werken. Voor een schematische uitwerking, zie Bijlage 3.

Acceptatiecriteria

- Er moet een inlogformulier getoond worden, waar de gebruiker zijn gebruikersnaam en wachtwoord invoert.
- Als de inloggegevens incorrect zijn, moet de gebruiker terug worden geleid naar het inlogformulier met een foutmelding
- als de inloggevens correct zijn, gaat de gebruiker verder naar het admin gedeelte.

11.5 Lijsten in het admin gedeelte

> Als een admin van de applicatie, wil ik toegang hebben tot lijsten met de gegevens van het evenement zodat ik het evenement kan beheren.

Tijdens het evenement worden er verschillende handelingen uitgevoerd: er worden mensen present gemeld en er worden leads geworven. Deze acties resulteren in data die terug te vinden en te manipuleren moet zijn. Er zijn verschillende handelingen die terug moeten komen, die hierboven niet direct naar voren komen. Hieronder staan de verschillende routes die genomen kunnen worden om de user story hierboven uit te voeren. Voor een schematische uitwerking, zie Bijlage 4.

11.5.1 Het wijzigen van een individueel record

> Als een admin van de applicatie, wil ik toegang hebben tot lijsten met de gegevens en wil ik een individueel record kunnen wijzigen zodat ik het evenement kan beheren.

Tijdens een evenement kan het redelijk hectisch zijn en is er altijd een mogelijkheid tot het maken van fouten. Vandaar dat het van belang is dat een individueel record aangepast kan worden.

Acceptatiecriteria

- Er moet een 'wijzig' knop komen bij elk record
- Het wijzigen moet gaan via een soortgelijk formulier als waar de initiële gegevens in zijn ingevoerd

11.5.2 Het verwijderen van een individueel record

> Als een admin van de applicatie, wil ik toegang hebben tot lijsten met de gegevens en wil ik een individueel record kunnen verwijderen zodat ik het evenement kan beheren.

Tijdens een evenement kan het redelijk hectisch zijn en is er altijd een mogelijkheid tot het maken van fouten. Vandaar dat het van belang is dat een individueel record verwijderd kan worden.

Acceptatiecriteria

- Er moet een 'verwijder' knop komen bij elk record
- Na het klikken hierop moet om bevestiging worden gevraagd of het verwijderen echt moet plaatsvinden.

11.5.3 Het uploaden van alle niet geüploade records

> Als een admin van de applicatie, wil ik toegang hebben tot lijsten met de gegevens en wil ik alle niet geüploade records kunnen uploaden zodat ik het evenement kan beheren.

Een van de belangrijkste functies van het admin gedeelte van de applicatie is de mogelijkheid om de records te uploaden via REST naar de server. Zo worden de gegevens opgeslagen en kan er echt wat gedaan worden met de verzamelde gegevens.

Acceptatiecriteria

- Er moet een 'upload alle' knop komen boven de lijst
- Na het klikken hierop moet om bevestiging worden gevraagd of het uploaden echt moet plaatsvinden.

11.5.4 Het verwijderen van alle geüploade records

> Als een admin van de applicatie, wil ik toegang hebben tot lijsten met de gegevens en wil ik alle geüploade records kunnen verwijderen zodat ik het evenement kan beheren.

Er zijn verschillende redenen waarom alle geüploade records verwijderd zouden moeten worden. Zoals het opnieuw beginnen met registreren (aan het begin van een dag), omdat er (te) veel records in de applicatie zitten, etc. Het is belangrijk dat een dergelijke optie is opgenomen.

Acceptatiecriteria

- Er moet een 'verwijder alle' knop komen boven de lijst
- Na het klikken hierop moet om bevestiging worden gevraagd of het verwijderen echt moet plaatsvinden.

11.5.5 Het resetten van alle geüploade records

Als een admin van de applicatie, wil ik toegang hebben tot lijsten met de gegevens en wil ik alle geüploade records kunnen resetten zodat ik het evenement kan beheren.

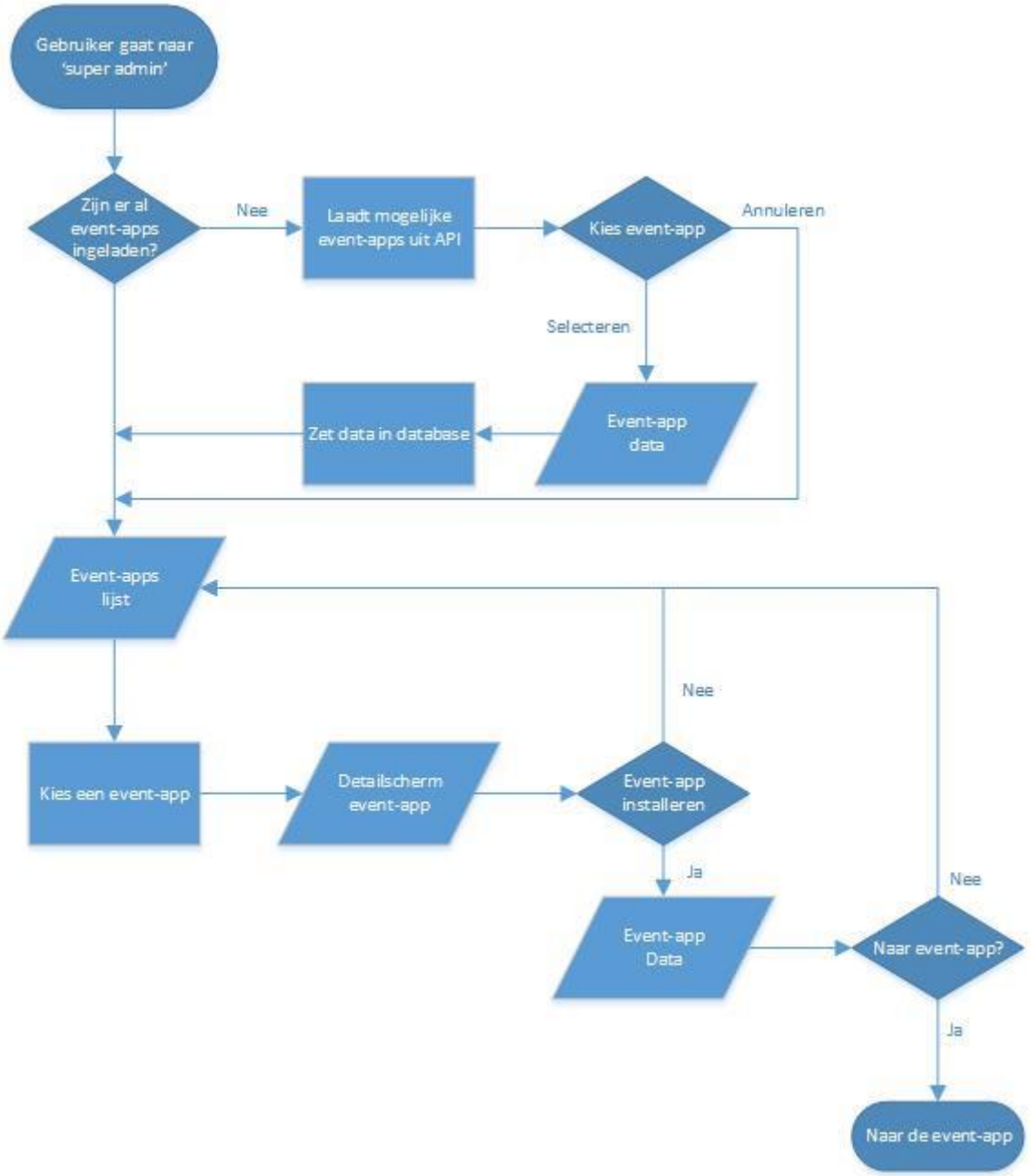
Door de records te resetten, kunnen deze opnieuw geüpload worden. Er zijn verschillende redenen waarom alle geüploade records gereset zouden moeten worden. Er kan bijvoorbeeld iets mis zijn gegaan de serverkant. Het is in ieder geval van belang dat deze optie is opgenomen.

Acceptatiecriteria

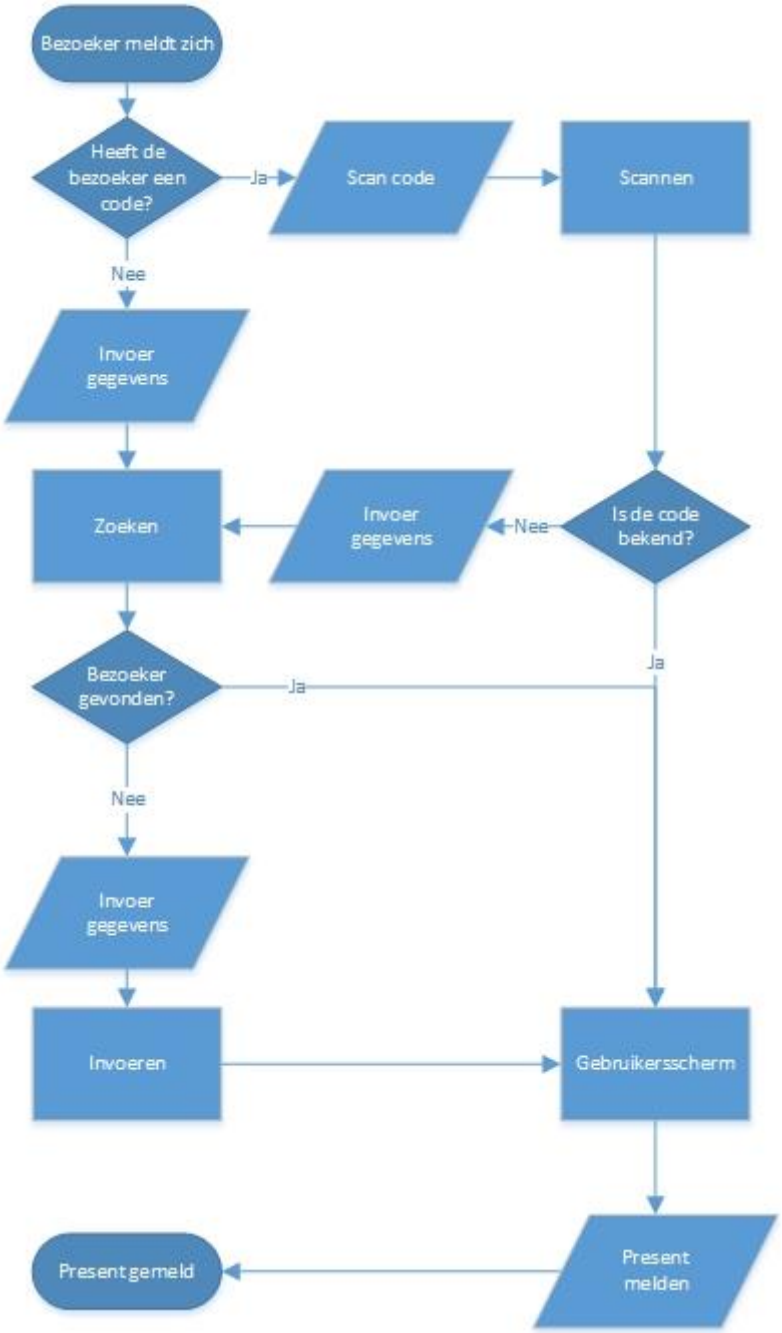
- Er moet een 'reset alle' knop komen boven de lijst
- Na het klikken hierop moet om bevestiging worden gevraagd of het resetten echt moet plaatsvinden.

12. Bijlagen

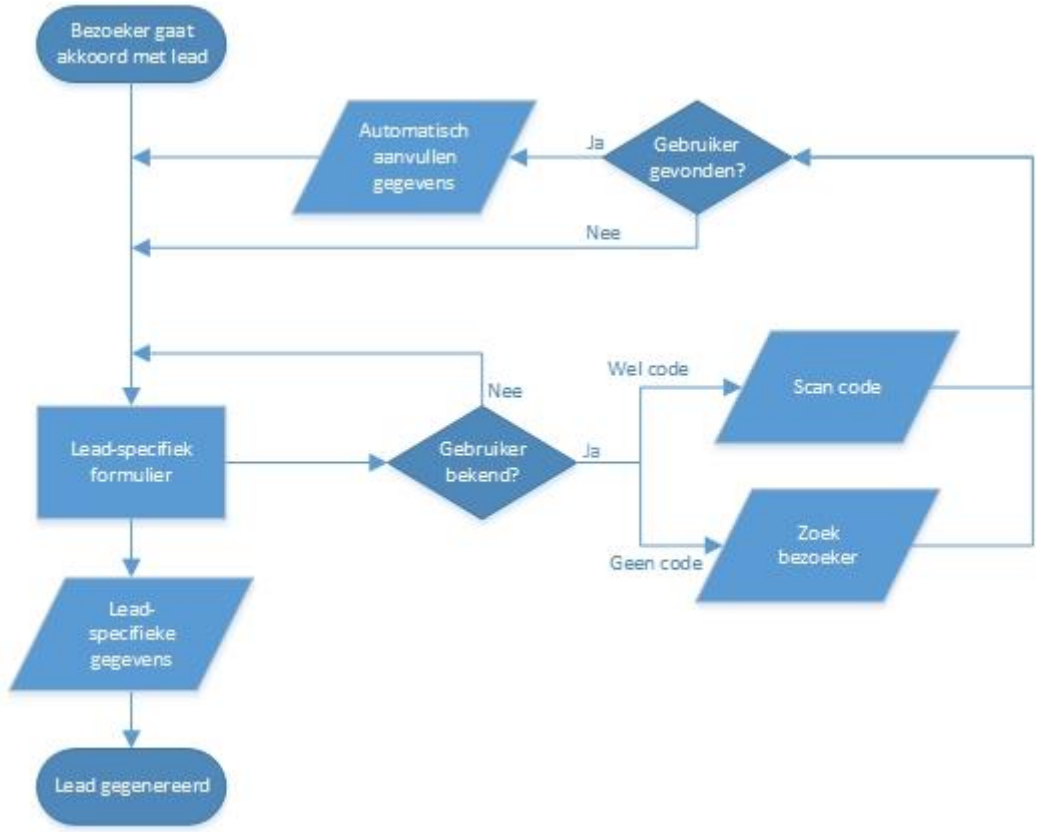
12.1 Flowchart user story 2.1



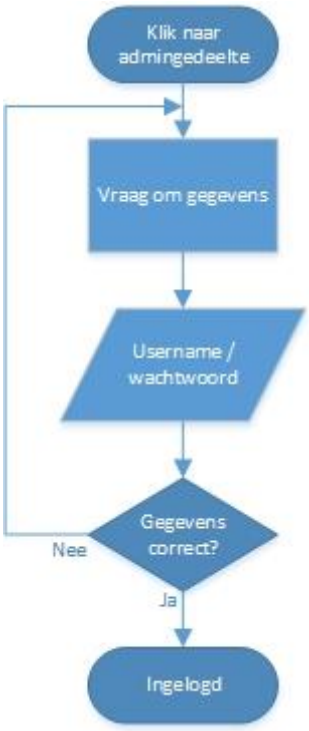
12.2 Flowchart user story 2.2



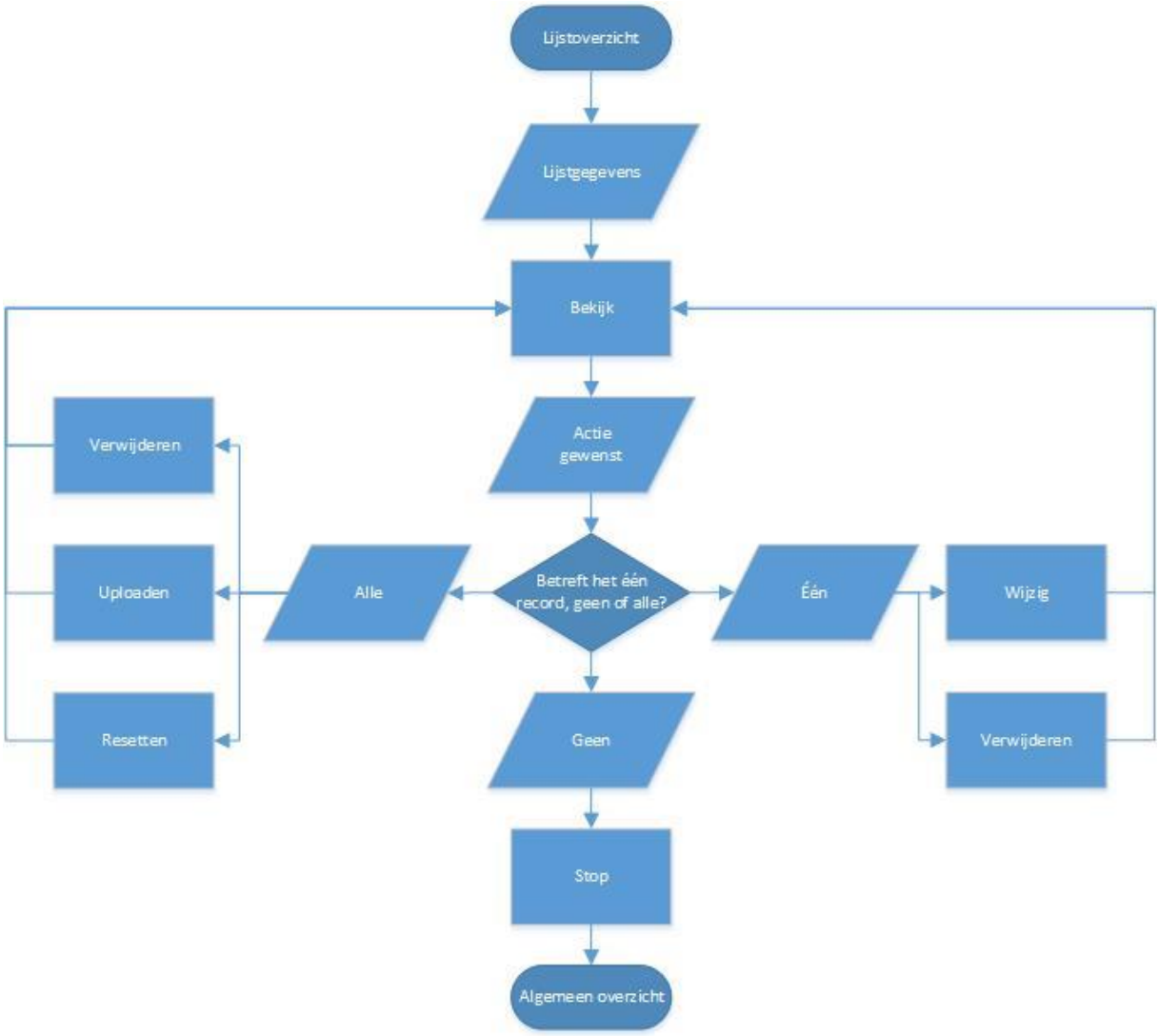
12.3 Flowchart user story 2.3



12.4 Flowchart user story 2.4



12.5 Flowchart user story 2.5



Testrapport

Mobiele applicatie voor evenementensysteem

Bram Slob
11050365
Informatica deeltijd

Revisiehistorie

Versie	Revisiedatum	Wijzigingen
1.0	05-06-2014	Eerste oplevering

Distribution

Dit document is bedoeld voor:

Naam	Functie	Datum van uitgave	Versie
Gerard Mijnarends	Begeleidend examinator	05-06-2014	1.0
Ben Kuiper	Tweede examinator	05-06-2014	1.0

13. Inleiding

Het document dat u voor u hebt liggen is het testrapport voor de verschillende testcases die op de mobiele applicatie voor het evenementensysteem van toepassing zijn. Deze testcases zijn opgesteld op basis van de user stories, eventueel aangevuld met cases om de requirements te testen.

In hoofdstuk twee zullen de verschillende testcases worden opgesomd. Per testcase is een titel opgenomen die kort beschrijft wat het te testen onderdeel is. Daarnaast staat er per testcase, welke user story en/of requirement het betreft en wat de eventuele stappen zijn om deze punten te testen. Als laatste kan aangegeven worden wat de resultaten van de test zijn.

In hoofdstuk drie worden de resultaten van de test kort besproken en eventuele uitspringende resultaten bekeken.

In hoofdstuk vier wordt, indien van toepassing, ingegaan op het eventuele vervolg op het testen, zoals welke wijzigingen er gedaan zouden moeten worden om te voldoen aan de requirements.

14. Acceptatietest

Om de kwaliteit van de applicatie goed te kunnen meten, is deze acceptatietest opgezet. De bedrijfsmentor voert deze uit om te zien of de verschillende requirements en user stories naar behoren in de applicatie zijn verwerkt.

14.1 De applicatie draait op een mobiel platform

Omschrijving			
Er wordt getest of de applicatie werkt op een mobiel platform. Dit zal voor deze test een Android tablet betreffen.			
Betreft			
User stories	Geen		
Requirements	'De applicatie moet op verschillende platformen kunnen draaien'		
Te ondernemen stappen			
○ Laadt de applicatie op de Android tablet			
Resultaat			
◉ GOED	○ GOED met wijziging	○ TE WIJZIGEN	○ NIET GOED
Opmerkingen			

14.2 Er kan een event-app worden ingeladen

Omschrijving			
Er wordt getest of de applicatie een event-app kan inladen, de configuratie hiervoor kan uitvoeren en er naar de event-app genavigeerd kan worden.			
Betreft			
User stories	Geen		
Requirements	'Meerdere event-apps binnen één applicatie'		
Te ondernemen stappen			
○ Klik op 'voeg een app toe' ○ Selecteer de 'FaceGames app' ○ Klik in de lijst op de app ○ In het nieuwe scherm, klik op inladen ○ Zodra het zwarte scherm weg is, klik op 'naar app'			
Resultaat			
◉ GOED	◉ GOED met wijziging	○ TE WIJZIGEN	○ NIET GOED
Opmerkingen			
Het is nog niet mogelijk om kleuren aan te geven voor de verschillende stukken van een event-app. De CSS die wordt doorgegeven via de API moet nog goed worden opgenomen.			

14.3 De applicatie kan offline werken

Omschrijving			
Er wordt getest of de applicatie offline kan werken.			
Betreft			
User stories	Geen		
Requirements	‘De applicatie moet offline werken’		
Te ondernemen stappen			
○ Zet de WIFI van het apparaat uit ○ Zorg dat de applicatie (weer) aan staat ○ Log in met de gebruikersnaam ‘michiel’			
Resultaat			
◉ GOED	○ GOED met wijziging	○ TE WIJZIGEN	○ NIET GOED
Opmerkingen			

14.4 Een reservering kan present gemeld worden

Omschrijving			
Er wordt getest een reservering present gemeld kan worden.			
Betreft			
User stories	Presentie opnemen		
Requirements	Geen		
Te ondernemen stappen			
○ Voer in het bovenste invoerveld, naast ‘zoeken’, de code 123456789 in ○ Klik op zoeken ○ In het detailoverzicht van de reservering klik op ‘present melden’			
Resultaat			
◉ GOED	○ GOED met wijziging	○ TE WIJZIGEN	○ NIET GOED
Opmerkingen			

14.5 Een reservering kan gewijzigd worden

Omschrijving			
Er wordt getest of een reservering gewijzigd kan worden.			
Betreft			
User stories	Geen		
Requirements	Geen		
Te ondernemen stappen			
○ In het detailoverzicht, klik op wijzigen ○ In het formulier, wijzig een gewenst onderdeel van de relatie. ○ Klik onderaan op versturen ○ Zie dat de wijziging is doorgevoerd in het detailoverzicht			
Resultaat			
◉ GOED	○ GOED met wijziging	○ TE WIJZIGEN	○ NIET GOED
Opmerkingen			

14.6 Een reservering kan gezocht en gevonden worden

Omschrijving			
Er wordt getest of een reservering gevonden kan worden via ‘uitgebreid zoeken’			
Betreft			
User stories	Geen		
Requirements	Geen		
Te ondernemen stappen			

○ Open het menu door met de vinger van links naar rechts te schuiven/swipen
○ Klik op ‘scannen’
○ Klik in het resulterende scherm op de onderste knop ‘uitgebreid zoeken’
○ Vul in het resulterende scherm bovenin een van de volgende gegevens in:
  ○ Achternaam: Slob
  ○ E-mailadres: b.slob@faceworks.nl
  ○ Postcode: 2806 KD
○ Er komt een resultaat in beeld			

Resultaat			
◉ GOED	○ GOED met wijziging	○ TE WIJZIGEN	○ NIET GOED
Opmerkingen			

14.7 Een lead kan geworven worden

Omschrijving			
Er wordt getest of een lead geworven kan worden			
Betreft			
User stories	Leads werven		
Requirements	geen		
Te ondernemen stappen			
○ Open het menu door met de vinger van links naar rechts te schuiven/swipen ○ Klik op 'Leads' ○ Loop door het formulier heen ○ Houdt er rekening mee dat 'naam' op pagina twee verplicht is ○ Klik op de tweede pagina op 'versturen'			
Resultaat			
○ GOED	◉ GOED met wijziging	○ TE WIJZIGEN	○ NIET GOED
Opmerkingen			
Het formulier lijkt niet zich volledig te resetten. Het is geen groot probleem, maar het maakt het gebruik van het formulier wel eenvoudiger Daarnaast is het fijner als er een bedankpagina is na het invullen van een lead, waarna de gebruiker automatisch na enkele seconden naar de eerste pagina van het formulier wordt gestuurd.			

14.8 De leads kunnen verstuurd worden

Omschrijving			
Er wordt getest of de lead(s) verstuurd kunnen worden			
Betreft			
User stories	Lijsten in het admin gedeelte		
Requirements	Geen		
Te ondernemen stappen			
○ Open het menu door met de vinger van links naar rechts te schuiven/swipen ○ Klik op 'Admin' ○ Vul als wachtwoord 'michiel' in ○ In het resulterende scherm, veeg met de vinger van rechts naar links in de regel met 'Leads' ○ Klik op 'Gegevens versturen'			
Resultaat			
◉ GOED	○ GOED met wijziging	○ TE WIJZIGEN	○ NIET GOED
Opmerkingen			

14.9 De leads kunnen opnieuw verstuurd worden

Omschrijving			
Er wordt getest of de lead(s) opnieuw verstuurd kunnen worden			
Betreft			
User stories	Lijsten in het admin gedeelte		
Requirements	Geen		
Te ondernemen stappen			
○ Open het menu door met de vinger van links naar rechts te schuiven/swipen ○ Klik op 'Admin' ○ Vul als wachtwoord 'michiel' in ○ In het resulterende scherm, veeg met de vinger van rechts naar links in de regel met 'Leads' ○ Klik op 'Gegevens opnieuw versturen' ○ Vul als wachtwoord 'michiel' in			
Resultaat			
◉ GOED	○ GOED met wijziging	○ TE WIJZIGEN	○ NIET GOED
Opmerkingen			

14.10 Een lead kan verwijderd worden

Omschrijving			
Er wordt getest of een lead verwijderd kan worden			
Betreft			
User stories	Lijsten in het admin gedeelte		
Requirements	Geen		
Te ondernemen stappen			
○ Open het menu door met de vinger van links naar rechts te schuiven/swipen ○ Klik op 'Admin' ○ Vul als wachtwoord 'michiel' in ○ In het resulterende scherm, veeg met de vinger van rechts naar links in de regel met 'Leads' ○ Klik op 'Lijst' ○ In het resulterende scherm, veeg met de vinger van rechts naar links in een regel ○ Klik op verwijderen ○ Vul als wachtwoord 'michiel' in			
Resultaat			
◉ GOED	○ GOED met wijziging	○ TE WIJZIGEN	○ NIET GOED
Opmerkingen			

14.11 De reserveringen kunnen verstuurd worden

Omschrijving			
Er wordt getest of de reservering(en) verstuurd kan worden			
Betreft			
User stories	Lijsten in het admin gedeelte		
Requirements	Geen		
Te ondernemen stappen			
○ Open het menu door met de vinger van links naar rechts te schuiven/swipen ○ Klik op 'Admin' ○ Vul als wachtwoord 'michiel' in ○ In het resulterende scherm, veeg met de vinger van rechts naar links in de regel met 'Reservations' ○ Klik op 'Gegevens versturen'			
Resultaat			
◉ GOED	○ GOED met wijziging	○ TE WIJZIGEN	○ NIET GOED
Opmerkingen			

14.12 De reserveringen kunnen opnieuw verstuurd worden

Omschrijving			
Er wordt getest of de reservering(en) opnieuw verstuurd kunnen worden			
Betreft			
User stories	Lijsten in het admin gedeelte		
Requirements	Geen		
Te ondernemen stappen			
○ Open het menu door met de vinger van links naar rechts te schuiven/swipen ○ Klik op 'Admin' ○ Vul als wachtwoord 'michiel' in ○ In het resulterende scherm, veeg met de vinger van rechts naar links in de regel met 'Reservations' ○ Klik op 'Gegevens opnieuw versturen' ○ Vul als wachtwoord 'michiel' in			
Resultaat			
◉ GOED	○ GOED met wijziging	○ TE WIJZIGEN	○ NIET GOED
Opmerkingen			

15. Testresultaten

In dit hoofdstuk worden de resultaten van het testen kort beschreven.

Zoals aan de verschillende resultaten van de tests te zien is, zit de applicatie erg in de lijn der verwachtingen. Er zijn echter ook een aantal punten die verbeterd kunnen worden.

Zo is het geheel stijlen van een event-app nog niet zo ver als verwacht werd. Het is mogelijk om een CSS bestand naar de applicatie te sturen, maar deze doet op dit moment nog niets. Dit is een punt dat belangrijk is om te maken voordat een event-app in productie genomen gaat worden.

Daarnaast is het ook belangrijk om extra aandacht te geven aan de werking van het leadformulier. Het opnemen van een lead werkt en gaat goed. Echter aan het einde van het leadformulier wordt nu de alert methode van JavaScript gebruikt om aan te geven dat het formulier succesvol is verstuurd. Dat kan mooier en de wens is daar ook naar. Het is wenselijker om een soort van bedankpagina te hebben, waar de gebruiker op uit komt. Daarna is het de bedoeling dat de gebruiker ofwel zelf naar de eerste pagina van het formulier gaat, of na 3 seconden automatisch wordt daarnaartoe wordt gestuurd.

Eenmaal daar aangekomen kan een ander punt worden gevonden, namelijk dat het leadformulier niet volledig is gereset. Er zijn geen gegevens meer ingevuld, maar de behaviors zijn niet gereset en draaien nog, alsof ze aangezet zijn. Dit hoort niet en moet opgelost worden, voordat er een event-app in productie genomen gaat worden.

Naast deze punten is de applicatie conform de requirements en user stories, vandaar dat er met enige zekerheid gezegd kan worden dat de kwaliteit van de applicatie voldoende tot goed is.

16. Vervolg op testen

In dit hoofdstuk worden de resultaten in het licht gehouden van toekomstige ontwikkelingen en wordt gekeken wat er aan gedaan kan en moet worden.

De punten die naar voren zijn gekomen zijn belangrijk en hebben impact op de applicatie, ze moeten opgelost worden voordat de applicatie in productie genomen moet worden. Wanneer er verder gewerkt gaat worden aan de Iventer applicatie zal ook gekeken gaan worden naar deze punten.

De punten die verwerkt moeten worden zijn hierboven besproken. De werkzaamheden, die nodig zijn om deze punten te verhelpen zullen zijn:

Test	**Taak**
2.2	CSS bestand inladen en uitvoeren
2.2	CSS bestand kunnen opslaan en weer kunnen ophalen bij wisselen van event-app
2.8	Bedank scherm na succesvolle leadwerving
2.8	Automatisch na 3 seconden naar eerste pagina van het formulier gaan
2.8	Behaviors resetten en opnieuw uitvoeren om zo het formulier volledig schoon op te leveren.