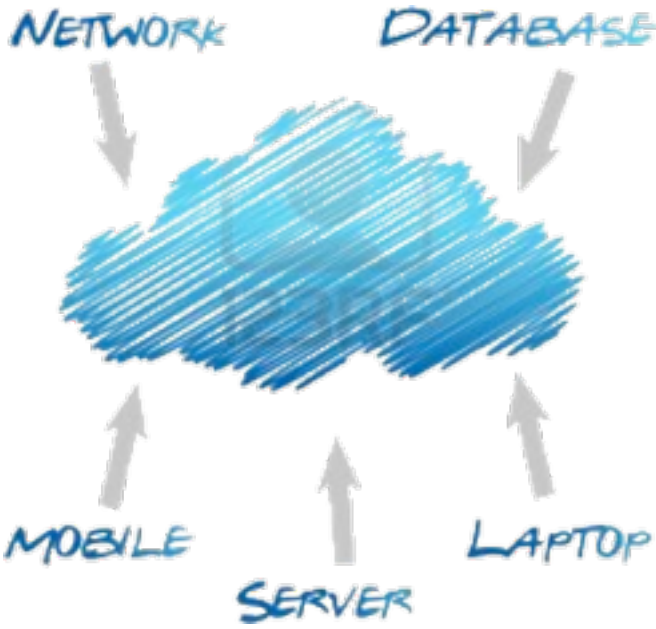
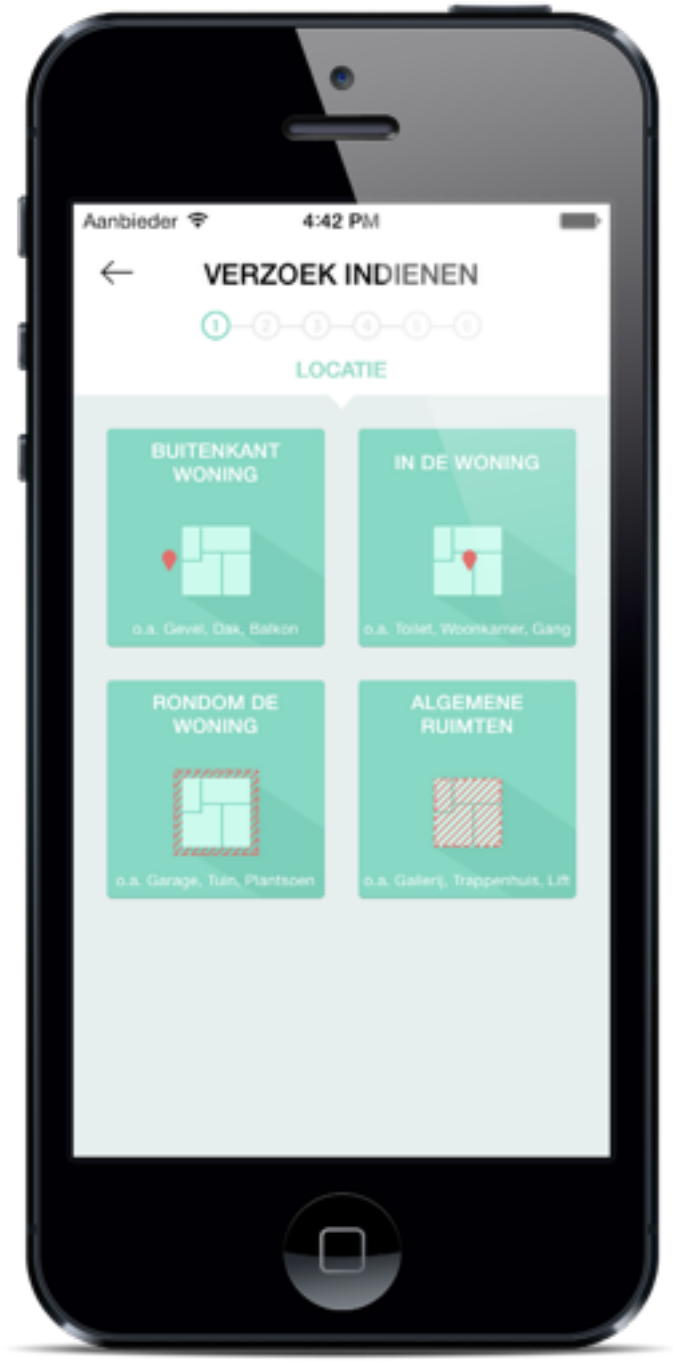
Reparatieverzoeken melden & beheren

Technische Informatica
Stephan de Bakker

31 Oktober 2013





Referaat

Afstudeeropdracht van Stephan de Bakker voor de studie Technische Informatica. In dit verslag wordt beschreven hoe een mobiele applicatie voor het iOS platform is opgezet inclusief een REST API op het web platform. Voor het mobiele platform is gebruik gemaakt van de taal Objective-C, daarnaast voor het web-platform is gebruik gemaakt van HTML, CSS, JSON en voornamelijk PHP. Ook het ontwerp van een database zal behandeld worden in dit verslag.

Tags: Objective-C, Databases, Gedistribueerd systeem, PHP, UML, MySQL, CoreData, MultiThreaded systeem.

Voorwoord

In dit verslag worden mijn bevindingen beschreven en wordt een onderbouwing gegeven voor de keuzes die ik tijdens de afstudeeropdracht voor Technische Informatica heb gemaakt. Deze opdracht wordt uitgevoerd om te bewijzen dat ik voldoende kennis heb opgedaan tijdens de studie Technische Informatica.

Dit verslag heeft als doel de lezer een indruk te geven van het gehele traject dat ik heb doorlopen. In dit traject komen problemen voor die te maken hebben met gedistribueerde systemen en de communicatie met meerdere onderdelen van dit systeem. Hierin zal vooral het ontwikkelen van een mobiele applicatie voor het iPhone OS beschreven worden. Daarnaast zal er uitleg worden gegeven over een REST API dat als communicatiemiddel dient voor deze mobiele applicatie.

In dit verslag zal een onderbouwing worden gegeven over hoe de doelstellingen van de afstudeeropdracht behaald en uitgevoerd zijn. Daarnaast wordt bewezen dat ik de competenties die ik heb opgesteld voor deze opdracht gehaald zijn op HBO niveau. Deze competenties zijn terug te vinden in het afstudeerplan (Zie bijlage X).

Ik wil graag Micheal van Lier bedanken en alle ontwikkelaars van Solware voor de kans om bij het bedrijf af te studeren en voor de hulp die ik hierbij kreeg. Ik heb veel geleerd van de professionele aanpak bij Solware en ben door deze afstudeeropdracht veel wijzer geworden.

Inhoudsopgave

Inleiding

De opdracht die ik zal uitvoeren voor het bedrijf Solware zal zich richten op het opzetten van een gedistribueerd systeem dat bestaat uit onder andere een API, een mobiele applicatie geschreven voor het iPhone platform en een beheer systeem dat bereikbaar is via een web browser. Ik zal onder andere onderzoek doen naar het opzetten van een MultiThreaded systeem geschreven in Objective-C dat op de meest efficiënte en snelle manier netwerk verzoeken kan verwerken. Daarnaast zal de beveiliging van de applicatie ook een grote rol spelen in de uitwerking van dit project, er zal bijvoorbeeld nagedacht moeten worden over gebruik van SSL en authenticatie met de server. Tijdens het doorlopen van de afstudeeropdracht zal antwoord gegeven worden op de volgende deelvragen uit het afstudeerplan (Zie bijlage X):

- Een gedistribueerd systeem opzetten dat bewoners van huurhuizen een mogelijkheid geeft om via een mobiele applicatie reparatieverzoeken te melden.
- Het creëren van een voorstel dat beschrijft wat de beste methode is om met gevoelige informatie om te gaan door middel van dit gedistribueerde systeem.
- Het ontwikkelen en realiseren van een API systeem dat als communicatiemiddel dient voor het gehele gedistribueerde systeem.
- De verschillende onderdelen van het systeem zullen zo worden opgezet eventuele toekomstige uitbreidingen eenvoudig te kunnen garanderen.

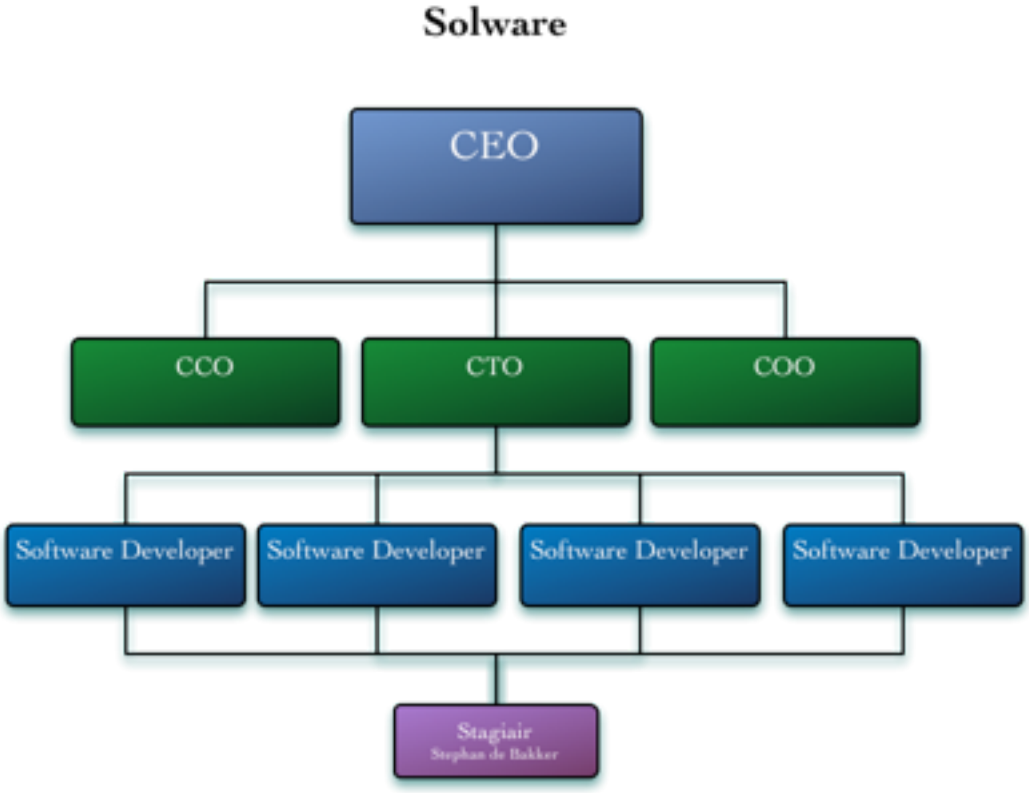
Solware ontwikkelt software voor zowel het web platform als een aantal mobiele platformen. Solware heeft een opdracht van het aannemersbedrijf BouwMeester gekregen om een mobiele applicatie te ontwikkelen. Deze applicatie moet het mogelijk maken om bewoners van huurhuizen reparatieverzoeken te laten indienen. Samen met het bedrijf BouwMeester en Solware zullen functionele eisen worden opgesteld om zo een product te kunnen ontwikkelen waar beide partijen gelukkig mee zijn.

Vooronderzoek

Organisatie

Het gedistribueerde systeem zal ontwikkeld worden voor het bedrijf Solware[1], dit bedrijf zal als proxy dienen voor het bedrijf waar de opdracht daadwerkelijk voor wordt ontwikkeld. Dit is het aannemersbedrijf BouwMeester[2] en is gevestigd in Den Haag. BouwMeester zal tijdens de ontwikkeling van het systeem een aantal keren met Solware en de afstudeerder vergaderen om zo functionele eisen te kunnen bepalen.

Solware is een softwarebedrijf dat zich bezighoudt met het ontwerpen en ontwikkelen van mobiele applicaties in combinatie met daartoe behorende beheersystemen die bereikbaar zijn via een browser. Het bedrijf telt in totaal 5 programmeurs die zich met het implementeren hiervan bezighouden. De afstudeerder zal meedraaien met de programmeurs van Solware en zal meedoen met alle bezigheden die te maken hebben met de afstudeeropdracht, dat wil zeggen deelnemen aan de interne reviews en planning fases. De afstudeerder zal als individu de opdracht uitvoeren onder begeleiding van het Solware Team. In onderstaande afbeelding is de organisatie van het bedrijf uitgebeeld. Hier is te zien dat de stagiair onder de ontwikkelaars van Solware staat. CCO, CTO en COO staan respectievelijk voor Chief Commercial Officer, Chief Technical Officer en Chief Operations Officer.



Figuur 1: Organisatie Solware. (Zie bijlage A).

[1] Bron: http://www.solware.nl/

[2] Bron: http://www.bouwmeester.org/

Doelstelling

De doelstelling is het voor bewoners eenvoudiger maken om werkzaamheden voor vastgoedbeheerders in te plannen. Dit heeft als voordeel dat er minder direct contact nodig is met de vastgoedbeheerder. Voor de complete doelstelling en verwachte resultaten verwijs ik u naar het afstudeerplan (zie bijlage X).

Achtergrond

Deze applicatie wordt ontwikkeld om het voor bewoners van huizen die onder een woningcorporatie vallen makkelijker te maken om reparatieverzoeken te melden bij de corporatie. Er is hierdoor geen direct contact meer met de woningcorporatie vereist. Foto's kunnen direct aantonen wat het probleem is en door gebruik te maken van een datum planner voor een afspraak is het gemakkelijk om snel een datum te kiezen die voor beide partijen (Bewoner & BouwMeester aannemers) geschikt is.

Door veel contact met de opdrachtgever zal bovenstaand worden gerealiseerd. Dit contact is van belang omdat op deze manier software kan worden opgeleverd die precies voldoet aan de wensen.

Randvoorwaarden

Een belangrijke randvoorwaarde is dat de mobiele applicatie ontwikkelt zal moeten worden op het iPhone platform. Dat betekent dat er gebruik gemaakt zal moeten worden van de Objective-C ontwikkel taal en het door Apple ontwikkelde Framework Cocoa Touch [3]. Dit is namelijk op moment van schrijven de enige methode om een "Native" applicatie te schrijven voor het Operating Systeem.

Tevens zal het beheersysteem en de API ontwikkeld worden op basis van het door Solware ontwikkelde SEF Framework (Zie bijlage C). Aan de hand van dit Framework is het eenvoudig om snel formulieren en andere onderdelen voor een beheersysteem te ontwikkelen.

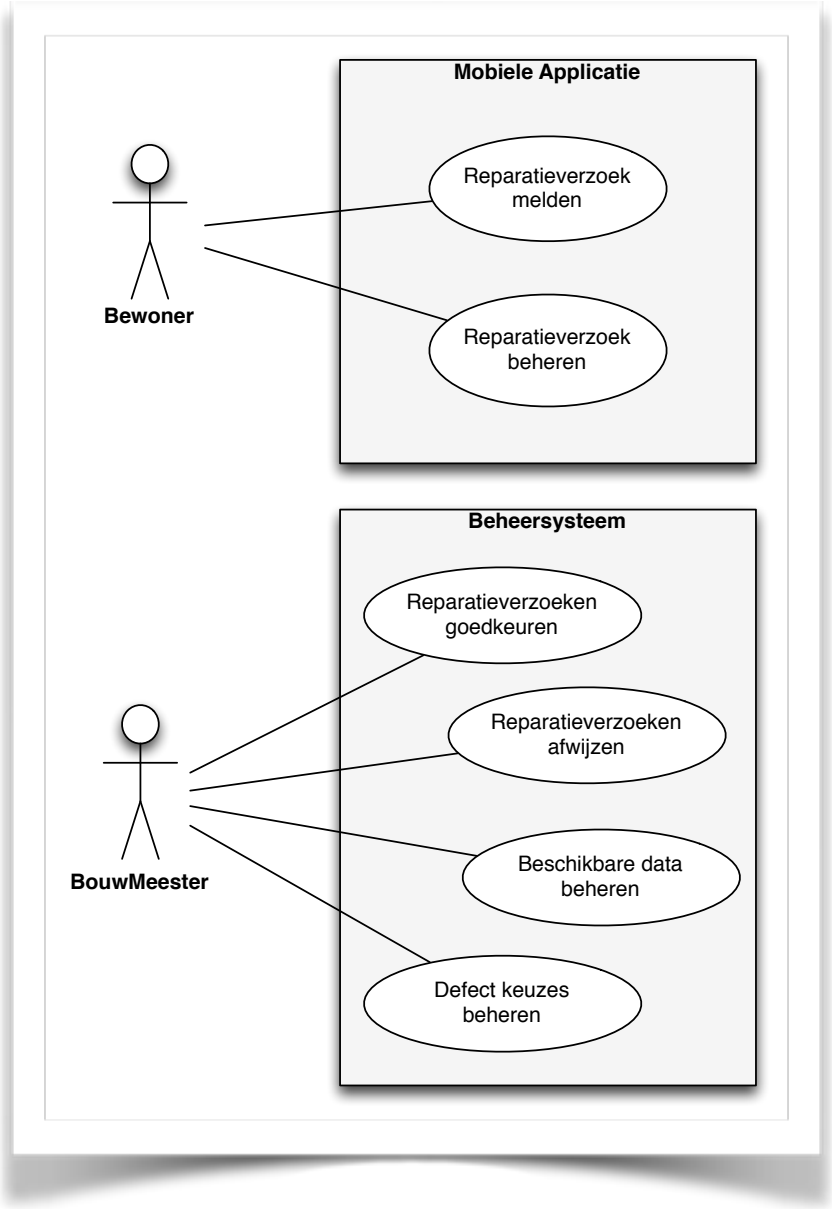
Systeem architectuur

Om een beeld te krijgen van de architectuur die opgeleverd zal moeten worden kunt u in figuur 2 een overzicht zien hoe het gehele systeem opgebouwd zal worden. Kort beschreven bestaat het systeem uit een REST API die benaderd kan worden vanuit de mobiele applicatie. Deze API evenals het beheersysteem communiceert op zijn beurt met de achterliggende database.

[3] Bron: https://developer.apple.com/technologies/ios/cocoa-touch.html

Figuur 2: Systeem architectuur (Zie bijlage D).

Daarnaast beschrijft het onderstaande use case diagram (Figuur 3) de acties die gebruikers kunnen uitvoeren. Een bewoner van een huurhuis kan bijvoorbeeld via de mobiele applicatie een reparatieverzoek melden of beheren. Een medewerker van BouwMeester heeft de mogelijkheid om via het beheersysteem reparatieverzoeken te beheren, beschikbaarheid van datums te beheren en de defect keuzes die gebruikt worden in de mobiele applicatie aan te passen.



Figuur 3: Use cases (Bijlage AA)

Ontwikkelmethode

Tijdens de gehele afstudeer periode zal de student meelopen met het ontwikkel team van Solware. Er is ook voor gekozen om gebruik te maken van de ontwikkel methode die dit team gebruikt voor alle projecten die uitgevoerd worden bij Solware. Deze methode is Agile Scrum en is uiteindelijk ook gekozen voor dit project aangezien de mogelijkheid dan ontstaat om gelijk te lopen met het Solware team voor interne reviews, planning meetings en andere bijeenkomsten [4].

Scrum

Scrum bestaat uit een aantal onderdelen die bij elke Sprint wordt doorlopen, een Sprint mag maximaal 1 maand duren maar in ons geval wordt elke Sprint in tijdsvakken van twee weken uitgevoerd. Dit aantal is gekozen omdat wederom het Solware Team ook Sprints van twee weken hanteert. Alle onderdelen van de Sprint zullen hier besproken worden in volgorde van uitvoering. Elk van deze onderdelen komt minimaal een keer voor tijdens een Sprint.

[4] Schwaber, K., & Beedle, M. (2001). *Agile Software Development with Scrum.* Prentica Hall

Sprint Planning Meeting
Tijdens de planning meeting worden alle onderdelen bepaald die tijdens de Sprint ontwikkelt zullen worden. Bijvoorbeeld het ontwikkelen van een pagina op de mobiele applicatie waar de gebruiker een beschrijving kan invullen of een API methode implementeren die beschikbare data terugstuurt. Per onderdeel wordt de tijd bepaald die de ontwikkelaar mag nemen om dit onderdeel te ontwikkelen. Op deze manier kan precies genoeg werk ingepland worden voor een Sprint, mits er er een goede tijd ingeschat wordt voor het werk. Een specifieke handeling bij Solware is het opsplitsen van onderdelen waarvan verwacht wordt dat er meer dan 13 uren nodig zijn om ze af te ronden. Hierdoor is het gemakkelijker om een precies aantal uren te berekenen. In het geval van de afstudeerder is het ideale aantal uur dat ingepland dient te worden rond de 71 uur van de maximaal 80 uur per Sprint, aangezien de rest van de tijd nodig zal zijn voor plannen, reviews en de andere Scrum activiteiten.

Sprint
Tijdens de Sprint zelf worden alle geplande onderdelen uitgevoerd worden. Deze onderdelen kunnen onderzoeken zijn of het implementeren van bepaalde functionaliteit. Voor dit project zal de afstudeerder in zijn eentje het team vormen, er zal van de student verwacht worden om zelfstandig te werken.

Daily Stand-Up
De daily stand-up is een korte dagelijkse samenkomst van het ontwikkelteam van Solware, tijdens deze bijeenkomst dient elke ontwikkelaar te vertellen wat hij de dag ervoor heeft gedaan, tegen welke problemen hij aan is gelopen en welke onderdelen hij de huidige dag gaat oppakken. Door dit dagelijkse onderdeel is elke ontwikkelaar op de hoogte van de ontwikkelingen van de andere ontwikkelaars.

Sprint Review
De Sprint review vind plaats aan het einde van elke Sprint, tijdens dit onderdeel worden alle geïmplementeerde functionele eisen getoond aan de rest van het team. Hier wordt feedback verwacht op het tussenproduct die tijdens de volgende Sprint verwerkt kan worden. Bij de Sprint review wordt kritisch gekeken naar de applicatie vanuit een gebruiker oogpunt, deze feedback zal geleverd worden door de Product Owner en het ontwikkelteam van Solware.

Sprint Retrospective
De Sprint retrospective is het laatste onderdeel van een Sprint, tijdens dit onderdeel wordt teruggekeken naar de afgelopen Sprint om zo te beoordelen wat er fout is gegaan en welke onderdelen positief zijn afgehandeld. De onderdelen die hieronder vallen hoeven niets met de huidige Sprint te maken hebben maar kan bijvoorbeeld ook met de sfeer te maken hebben of over hoe er wordt samengewerkt. Een aantal punten worden geselecteerd, waarbij specifiek naar oplossingen wordt gezocht, op deze manier kan extra aandacht worden besteedt aan deze onderdelen tijdens de volgende Sprint.

Rollen

Scrum heeft een aantal rollen die uitgevoerd moeten worden, zo heb je bijvoorbeeld team leden maar ook de zogenaamde Scrum Master. Wat deze rollen inhouden zal hier worden beschreven.

Scrum Master
De Scrum master is verantwoordelijk voor de juiste uitvoer van Scrum, dit is een belangrijke management taak die zal worden uitgevoerd door de begeleider binnen Solware. Hij zorgt ervoor dat elke dag de Daily Stand-Up wordt gehouden en plant de reviews en retrospective onderdelen van de Sprint in. Ook zorgt de Scrum master voor de belangrijke connectie met de Product Owner en maakt normaal gesproken in combinatie met de Product Owner een lijst met functionele eisen genaamd de Product Backlog. In dit geval zal de Product Backlog door de afstudeerder en COO

worden opgesteld en worden gevalideerd door de Product Owner. De rol van Scrum Master zal uitgevoerd worden door de COO van Solware, dit is een van de ontwikkelaars van Solware die bij de meeste projecten deze rol op zich neemt.

Product Owner
De product owner is in andere woorden een medewerker van het bedrijf waar de opdracht voor wordt uitgevoerd. In ons geval maken we gebruik van een zogenaamd proxy Product Owner, de CEO van Solware zal de rol van de proxy op zich nemen. Dat wil zeggen dat hij als tussenpersoon dient naar BouwMeester.

Team
Het team is de rest van de medewerkers die aan het project meewerken, in dit geval bestaat het volledige team alleen uit de afstudeerder. Normaal gesproken wordt aangeraden om minimaal 7 programmeurs in het team te hebben maar dat is voor dit project niet mogelijk.

Backlogs

Een Product Backlog is een lijst met functionele eisen die geïmplementeerd dienen te worden in de te ontwikkelen applicatie. Deze lijst wordt continue bijgewerkt naarmate het project vordert aangezien sommige eisen kunnen veranderen. De Product Owner kan bijvoorbeeld eisen wijzigen omdat tijdens de vorige review een andere implementatie dan verwacht is opgeleverd. In eerste instantie bestaat het bestand uit de functies van het project die in grote lijnen beschreven zijn en steeds verder worden uitgewerkt, dat wil zeggen dat het bestand altijd gewijzigd kan worden. De complete Product Backlog en alle individuele Sprint Backlogs zijn te vinden in bijlage E.

Waarom Scrum?

Zoals eerder al werd vermeld is er gekozen voor Scrum aangezien Solware deze ontwikkelmethode in alle projecten gebruikt. Op deze manier is het eenvoudig om de afstudeerder goed te beoordelen en eventueel te helpen tijdens het gehele ontwikkeltraject. Ook zal de afstudeerder hierdoor mee kunnen doen met daily stand-ups en interne reviews, zo wordt er snel feedback geleverd die van grote waarde is voor het succes van deze opdracht. Ook zullen de medewerkers alle geschreven code van de afstudeerder reviewen door middel van een SVN[5] systeem om zo ook feedback te kunnen leveren op code kwaliteit en de werking van deze code.

Scrum heeft ook als voordeel tegenover andere ontwikkelmethodes dat er zoals eerder genoemd snel feedback wordt geleverd. Bij andere methodes zoals RUP kunnen iteraties door bepaalde keuzes veel langer duren dan bij Scrum waardoor er in latere stadia pas feedback verkregen wordt van de opdrachtgever. Door scrum gebeurt dit om de twee weken wat ons de kans geeft om sneller op wijzigingen in te spelen. Dit is ook een van de redenen waarom we voor Sprints van twee weken hebben gekozen.

[5] Bron: http://subversion.apache.org

1. De Walking Skeleton

Sprint Planning Meeting

Tijdens de eerste Sprint planning meeting is uitleg gegeven over de werking van de Scrum ontwikkelmethode. De eerste stap van een Sprint is altijd de planning meeting, hierin wordt een planning gemaakt van alle onderdelen die ontwikkelt moeten worden in de eerste Sprint. Een van de meest interessante onderdelen is het Plan van Aanpak (*Zie bijlage F*). Verwacht wordt dat dit document in de eerste paar dagen wordt opgesteld om zo een beeld te krijgen van het gehele verloop van het afstudeertraject. In het Plan van Aanpak is achtergrond informatie van het zowel Solware als Bouwmeester opgenomen, evenals een planning van de komende Sprints tot het einde van het afstudeertraject.

In de eerste sprint zal de volgende functionaliteit gerealiseerd worden, daarnaast zullen documenten worden opgesteld die benodigd zijn voor het project zoals een Plan van Aanpak. De volgende stappen zullen in de eerste sprint uitgevoerd worden:

- Schrijven van een Plan van Aanpak.
- Het implementeren van de Walking Skeleton versie van de mobiele applicatie.
  - De mobiele applicatie zal de schermen bevatten om zo de flow te kunnen begrijpen.
  - De mobiele applicatie moet een probleem kunnen versturen inclusief een beschrijving en gekozen datum.
- Het bedenken en uitwerken van de verschillende User stories / functionele eisen.
- De eerste versie van de Product Backlog maken.

Door een Walking Skeleton te maken krijgt de opdrachtgever een eerste indruk van de applicatie die ontwikkelt zal worden. Op deze manier kan na deze ontwikkelfase snel feedback worden geleverd door de opdrachtgever over de initiële gedachte van de applicatie. Voordat een Walking Skeleton kan worden opgezet moeten eerst de User Stories (Zie bijlage G) en product Backlog worden opgesteld.

Een User Story beschrijft een actie in grote lijnen die een gebruiker kan uitvoeren met behulp van de applicatie. Dit kan geschreven worden in de volgende vorm: “Als gebruiker wil ik een foto kunnen toevoegen aan een defect.”. Door de functionele eisen in grote lijnen op deze manier te beschrijven is het makkelijker voor de ontwikkelaar om een applicatie te ontwikkelen die deze acties ook ondersteunt. De User Stories worden op zijn beurt verwerkt in het Product Backlog, dit bestand bestaat op het hoogste niveau uit deze User Stories.

Figuur 4: Eerste versie van de User Stories.

Bij Solware wordt vaak gebruikt gemaakt van een whiteboard met daarop alle User Stories die voor de applicatie van belang zijn. In figuur 4 is te zien hoe dit tot zijn recht komt voor de iPhone applicatie die in dit project ontwikkeld zal worden. De bovenste rij oranje briefjes duiden op de minst gedetailleerde eisen die geïmplementeerd moeten worden zoals het melden van een probleem. Onder het melden van een probleem bevinden zich meer gedetailleerde briefjes. Deze gele briefjes bevatten taken zoals een foto nemen of het toevoegen van een beschrijving. Bij de ontwikkeling van de applicatie zal per Sprint een horizontale rij van gele briefjes ontwikkeld worden. Deze horizontale rij is te zien als een Sprint Backlog. Per nieuwe sprint is het mogelijk dat nieuwe taken worden toegevoegd of oude taken worden verwijderd / verder uitgewerkt.

Zoals eerder te lezen is het de bedoeling om een mobiele applicatie te maken op het iPhone OS platform. In de eerste Sprint wordt focus gelegd op de Walking Skeleton van de mobiele applicatie. De applicatie zal al wel met een eenvoudige API moeten kunnen communiceren om een beschrijving en geselecteerde datum toe te voegen. Dit hebben we besloten omdat er dan bij de Sprint Review dan een werkende applicatie kan worden getoond. Met zeer weinig features, maar een defect kan wel worden beschreven en een afspraak kan ook worden ingepland.

Database

Om ervoor te zorgen dat de eerste Sprint kan werken is een database op de Web API benodigd en een manier om informatie van de telefoon te verzenden. Daarom is ervoor gekozen om voor de eerste Sprint een ontwerp van de achterliggende database te maken en een klasse model van hoe het verzenden en versturen op de telefoon zou werken.

Database typen & keuze

De achterliggende database zal gekoppeld moeten worden aan een huidig bestaand ontwerp met verschillende relaties naar reeds bestaande tabellen. Zo zal bijvoorbeeld een nieuwe tabel gekoppeld moeten worden aan een bestaande tabel die alle onderaannemers bevat. Deze onderaannemers moeten geselecteerd kunnen worden vanuit de mobiele applicatie. De huidige database is volledig geïmplementeerd door middel van MySQL en de InnoDB tabel engine. Voor de uitbreiding van de database kan een keuze worden gemaakt tussen de engines InnoDB en MyISAM. De voor- en nadelen van beide engines worden in dit hoofdstuk met elkaar vergeleken om een goede keuze te kunnen maken.

MyISAM[6]
Dit is een MySQL engine die al langer beschikbaar is en is daarom het meest geoptimaliseerd in bepaalde onderdelen. Een ander groot voordeel hiervan is dat het in vergelijking met InnoDB relatief weinig geheugen en weinig harde schrijf ruimte gebruikt. Alleen de primaire sleutels van de database worden opgeslagen in het geheugen waarna rijen die geselecteerd worden van de schijf gelezen moeten worden. Ook kan MyISAM extra geheugen besparen door tabellen volledig te comprimeren, hierdoor wordt de tabel in alleen-lezen gezet maar is het opzoeken van informatie uit deze tabel heel snel.

Een groot nadeel van MyISAM is dat er geen ondersteuning is voor Foreign Keys, hierdoor is het niet mogelijk om relaties op te geven tussen de verschillende tabellen. Terwijl deze mogelijkheid wel benodigd is omdat er een koppeling moet komen naar een InnoDB database van gebruikers. Zo zal een onderaannemer gekoppeld moeten worden een een nieuw reparatieverzoek dat is toegevoegd vanuit de mobiele applicatie. Door middel van Foreign Keys kan dit gerealiseerd worden.

Een ander nadeel bij MyISAM is het op tabel niveau locken van de database. Wanneer een tabel gelockt wordt door bijvoorbeeld een langdurige schrijfactie kan het zo zijn dat andere processen lang moeten wachten tot ze ook toegang krijgen tot deze tabel om te schrijven. In tegenstelling tot InnoDb, deze engine ondersteund namelijk de mogelijkheid om dit op rij-niveau uit te voeren.

InnoDB[7]
Deze MySQL engine heeft als groot voordeel dat het ondersteuning biedt voor gebruik van Foreign Keys. Door deze ondersteuning is het mogelijk om relaties tussen tabellen door te geven en wat er gebeurt wanneer de gekoppelde relatie wordt aangepast of verwijdert. Dit betekent dat wanneer je een rij verwijdert in bijvoorbeeld een gebruiker type tabel, je het zo kan instellen dat alle gebruikers van een bepaald type worden verwijdert.

Een heel belangrijk voordeel van de InnoDB engine is het gebruik van transacties en een log bestand voor deze transacties. Een transactie wordt pas toegepast op de database wanneer deze klaar is. Dit heeft als voordeel dat wanneer de database server crasht dat de informatie nog niet is weggeschreven maar in de cache is weggeschreven, de cache informatie is dan verloren maar de database zal niet corrupt zijn. Dit kan bij MyISAM wel gebeuren waardoor de database niet meer

---

[6] Bron: http://dev.mysql.com/doc/refman/5.0/en/myisam-storage-engine.html

[7] Bron: http://dev.mysql.com/doc/refman/5.0/en/innodb-storage-engine.html

bruikbaar zal zijn. Daarnaast is het dan dus mogelijk voor meerdere gebruikers om tegelijkertijd een bepaalde tabel te wijzigen. Ook maakt InnoDB gebruik van locking op rij-niveau in plaats van op tabel-niveau. Dit heeft als voordeel dat andere processen van gebruikers minder lang of niet hoeven te wachten tot een tabel vrij is, de kans is namelijk minder groot dat twee gebruikers tegelijkertijd naar een bepaalde rij schrijven.

	MyISAM	InnoDB
Ondersteunt transacties?	Nee	Ja
Geheugen verbruik	Laag	Hoog
Ondersteunt relaties? (Foreign Keys)	Nee	Ja
Herstelgedrag	Slecht	Uitstekend
Locking	Per tabel	Per rij

Figuur 5: Vergelijking MyISAM vs InnoDB

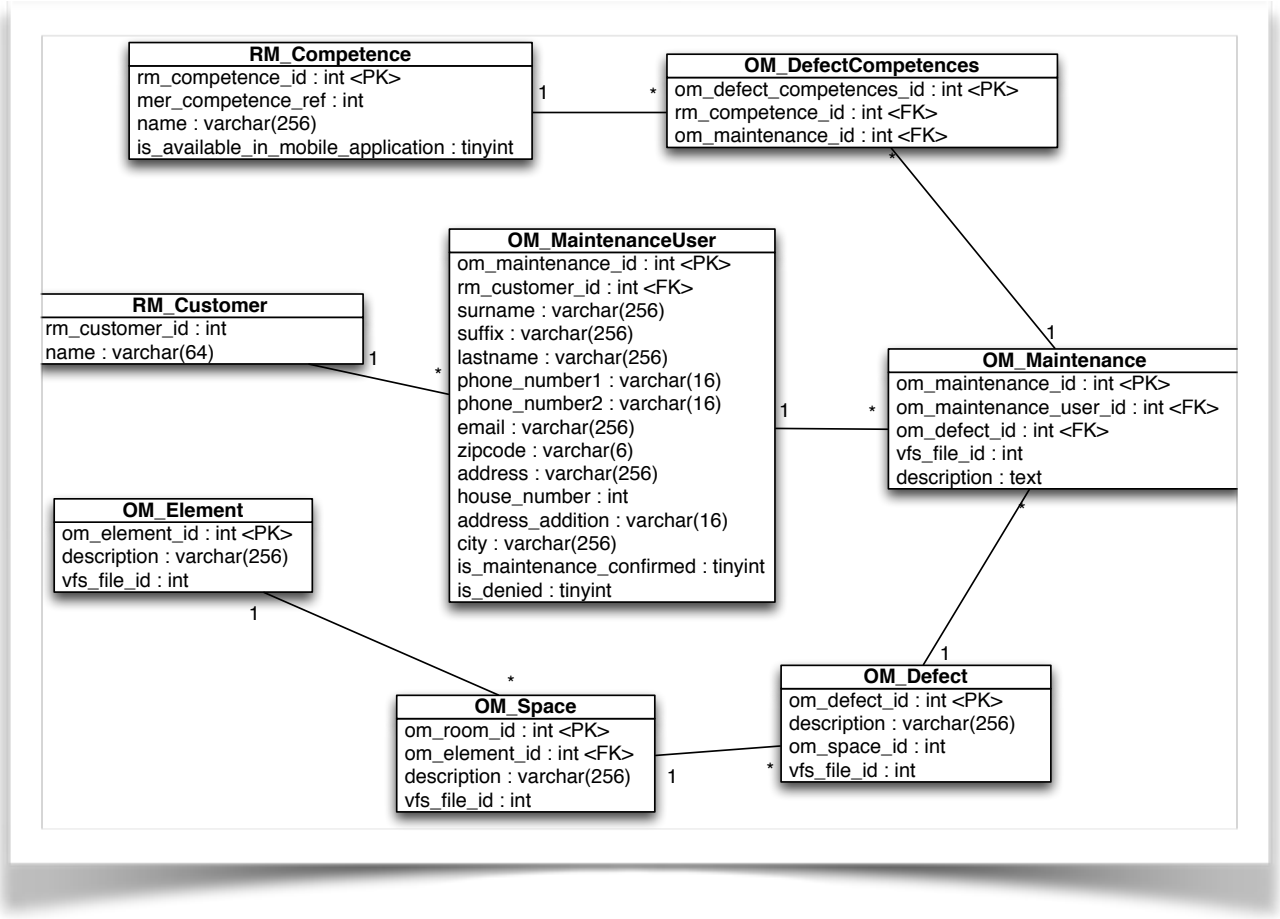
Figuur 5 zet de verschillende voor-en nadelen van InnoDB en MyISAM tegenover elkaar, aan de hand van deze tabel is te zien dat InnoDB in ons geval een betere keuze is. De ondersteuning van Foreign Keys is reden genoeg om gebruik te maken van InnoDB omdat deze nodig is om een relationele database op te zetten. Ook moet de database op bepaalde punten gekoppeld worden aan een reeds bestaande InnoDB database met relaties.

Ontwerp

In dit verslag zal veel gerefereerd worden naar objecten en methoden. Een klasse wordt door middel van de stijl *Klasse* gedefinieerd en een methode of functie als *methodeOfFunctieNaam*. Deze notatie wordt ook gebruikt voor objecten die in de database opgeslagen worden, een object wordt dan gezien als een tabel. De code is in de bijlagen te zien, voor een inhoudsopgave om snel de code op te vinden van een bepaalde klasse verwijs ik naar bijlage H. Objecten met de prefix "*NS*" zijn niet geschreven door de afstudeerder maar door Apple en zijn onderdeel van het Cocoa Touch Framework. Informatie over deze objecten zijn te vinden via de Apple Developer Library[8].

Het eerste database ontwerp zal gebouwd worden met een minimaal aantal tabellen. Voor een groot deel zullen alleen de tabellen die in deze Sprint van belang kunnen zijn worden ontworpen. In de Sprint Backlog is te zien dat de belangrijkste onderdelen van de huidige Sprint het toevoegen van een beschrijving en datum voor de afspraak zijn. Hier zal in het eerste ontwerp vooral op gefocust worden. Een aantal versies zijn voorafgegaan aan het ontwerp dat gebruikt werd voor de eerste Sprint, de uiteindelijke versie voor de eerste Sprint staat in figuur 6 afgebeeld.

[8] Bron: https://developer.apple.com/library/ios/navigation/



Figuur 5: Database ontwerp Sprint 1 (Zie bijlage I)

Zoals u kunt zien bestaat de database uit een hoofdtabel met de meeste informatie die gekoppeld is aan meerdere tabellen die onder andere het type van het defect beschrijven, een tabel voor de woningcorporatie van de gebruiker en een tabel die de beschrijving en afbeelding zal bevatten van het defect. Ook de relaties tussen de tabellen staat beschreven in figuur 5. De tabel *OM_MaintenanceUser* bevat alle informatie van de gebruiker, zoals naam en email-adres maar ook de adresgegevens die gebruikt zullen worden voor de locatie van het defect. De *OM_Maintenance* tabel bevat de informatie van een defect, namelijk een beschrijving en een koppeling naar het bestandssysteem van het huidige platform waar een afbeelding wordt opgeslagen. Als laatste bevat deze tabel een koppeling naar een *OM_Defect*, dit is de beschrijving van een defect. Dit defect is op zijn beurt een onderdeel van een *OM_Space* (Ruimte) die op zijn beurt weer een onderdeel is van een *OM_Element*. Dit leidt uiteindelijke tot een probleem zoals 'Licht in portiek is kapot.'.

In volgende Sprints is het de bedoeling om de mobiele applicatie uit te breiden met een selectie van elementen, ruimtes en defecten. Daarom zijn deze tabellen in deze versie reeds toegevoegd. De volgende versie van de database zal verder uitgebreid worden met tabellen die het mogelijk maakt om een datum en tijdsvak in te delen bij een defect van een gebruiker. Er zullen meerdere onderdelen pas later worden uitgebreid omdat dit nu nog niet nodig is. Dit is een belangrijk onderdeel van Scrum omdat volgens Scrum per Sprint alleen de functionaliteit opgeleverd en gemaakt wordt in hoeverre dat is ingepland.

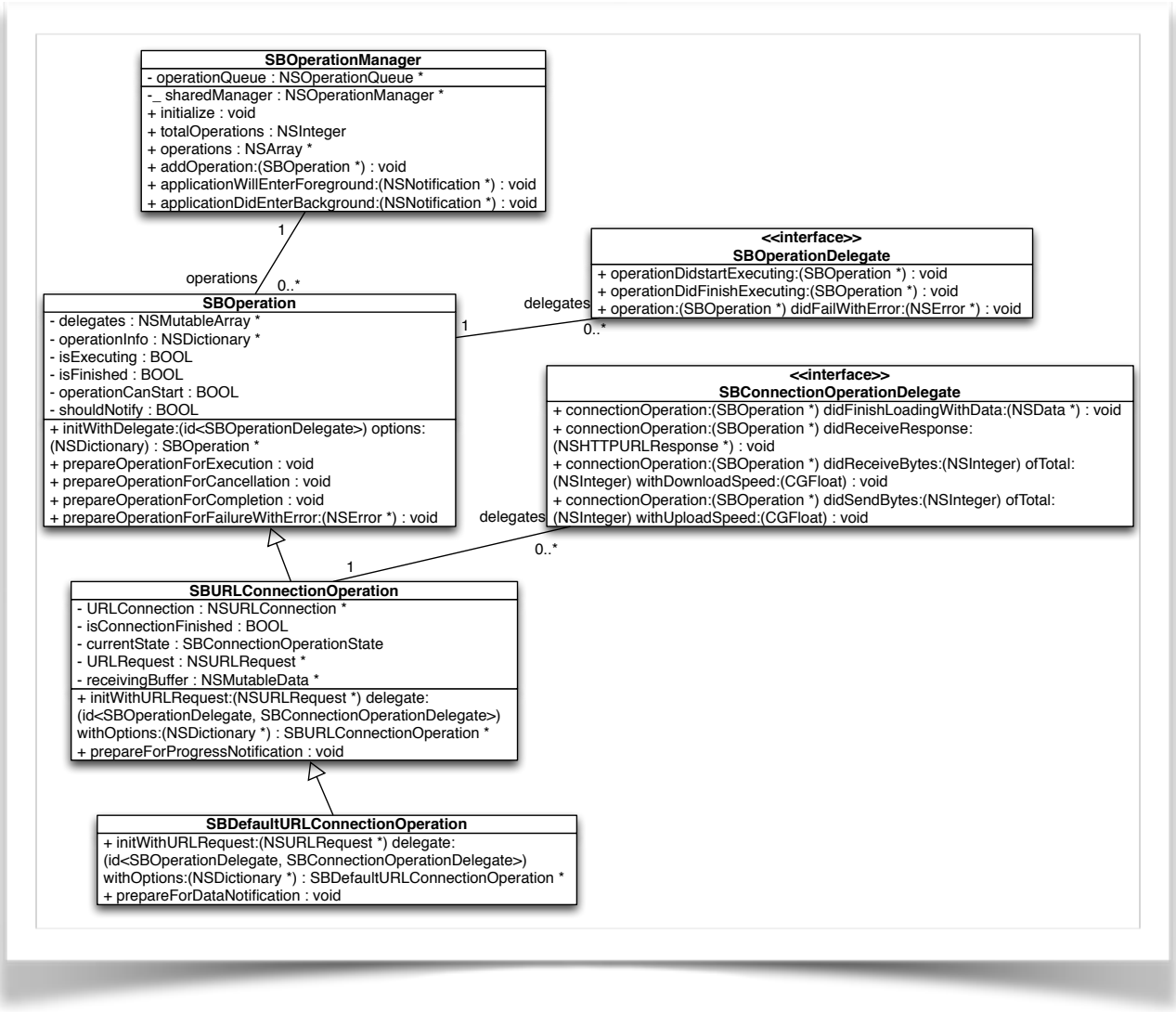
Mobiele applicatie

In dit onderdeel wordt het ontwerpen en ontwikkelen van de mobiele applicatie aan de hand van de Sprint Backlog van de eerste Sprint beschreven. De onderdelen die behandeld worden zijn het

afhandelen van background threads, verder genoemd operaties, en de implementatie van de datum selectie.

Operaties (MultiThreading)

Omdat er in de eerste versie van de mobiele applicatie een mogelijkheid moet zijn om te communiceren met de API is het van belang dat hier een standaard voor wordt geschreven waar in latere stadia op voortgebouwd kan worden. Deze standaard is een threading systeem waarbij zogenaamde operaties aangemaakt kunnen worden die informatie van de server downloaden. In deze Sprint zal bijvoorbeeld geïmplementeerd moeten worden dat een nieuw reparatieverzoek verstuurd kan worden naar de API. In een latere Sprint zal worden nagedacht over de beveiliging en efficiëntie van deze Thread functionaliteit. In figuur 6 (Zie bijlage J) is het eerste ontwerp te vinden, deze zal gebruikt worden in de applicatie.



Figuur 6: Ontwerp Ontvangen / verzenden informatie. (Zie bijlage J).

Het *SBOperationManager* object handelt meerdere *SBOperation* objecten af, dat wil zeggen dat het aan de hand van systeem specificaties en huidige status deze objecten start. In de manager bevindt zich een *NSOperationQueue* object. Dit object is deel van het standaard Framework en houdt rekening met systeemspecificaties zoals eerder beschreven.
Een *NSOperation* object is zo abstract mogelijk gemaakt om er voor te zorgen dat hier subklassen van gemaakt kunnen worden. Zo kunnen objecten worden geïntroduceerd die informatie

verzenden of ontvangen via de API ,of die bestanden wegschrijven op de lokale harde schijf. Door deze keuze is het eenvoudig om in een later stadium nieuwe objecten te introduceren die ook ondersteund worden door het manager object. In de implementatie van de basis klasse bevindt zich een aantal methodes die de manager gebruikt om te beslissen of de operatie gestart kan worden. Bijvoorbeeld de *isReady* methode die door het operatie object teruggegeven kan worden om aan te geven of het object klaar is om uitgevoerd te worden. Daarnaast kunnen subklassen de waarde van *isFinished* en *isExecuting* aanpassen om aan te geven wanneer het object uit het geheugen gehaald kan worden door de manager. Deze eerste methode geeft namelijk aan of de operatie alle acties heeft uitgevoerd die benodigd zijn voor het wel of niet succesvolle afronden.

Wanneer een *SBOperation* object uitgevoerd kan worden wordt de *start* methode aangeroepen vanuit het *SBOperationManager* object. Het operatie object zorgt er zelf voor dat een nieuwe thread wordt aangemaakt die alle acties van de operatie uitvoert. Wanneer dit gebeurd is zorgt de operatie er zelf voor dat de *isFinished* methode “true” retourneert. Aan de hand van deze waarde zal het manager object het object uit de queue plaatsen en verwijderen uit het geheugen. Door middel van Key-Value Coding[9] kunnen deze acties vrijwel direct na aanpassing van de waarde worden toegepast. Deze ontwikkelmethode is in de bijlagen beschreven. Ook de implementatie van de verschillende *SBOperation* objecten is te vinden in de bijlagen (Zie bijlage H voor inhoudsopgave code) met een meer gedetailleerde beschrijving van de werking van de individuele objecten.

Voor communicatie tussen het *SBOperation* object en objecten die geïnteresseerd zijn in het resultaat en voortgang van dit object wordt gebruik gemaakt van delegatie[10]. Per operatie object kan een enkel delegatie object worden opgegeven die een aantal methodes implementeert die vanuit het *SBOperation* object worden aangeroepen. Op deze manier blijft het delegatie object op de hoogte van bijvoorbeeld het ingeladen resultaat van de API.

Datum selectie

In de planning en Sprint Backlog van de eerste Sprint (zie bijlage E) is al omschreven dat de mogelijkheid moet worden ontwikkeld die het voor gebruikers mogelijk maakt om een datum te kiezen waarop het geselecteerde defect gerepareerd zal worden. Dit zal mogelijk gemaakt worden door een tabel te tonen in de mobiele applicatie met een aanpasbaar aantal keuzes die via de API ingeladen worden. Deze selectie moet namelijk bewerkbaar zijn door de opdrachtgever om precies aan te kunnen geven welke datum gebruikers wel of niet kunnen kiezen.

Om de datum informatie in te laden wordt gebruik gemaakt van een *SBOperation* object dat de connectie met de API opent en ophaalt. De API heeft op een vrij eenvoudige wijze een methode geïmplementeerd die “dummy” informatie terugstuurt. Deze informatie zal in een later stadium gekoppeld worden aan de database. Er is voor gekozen om dit op dit moment niet te maken omdat dit veel tijd zou kosten en nog niet een onderdeel is van de Sprint Backlog. De informatie die teruggestuurd wordt is in het formaat JSON. Een andere optie is het XML formaat maar aangezien een JSON structuur over het algemeen minder karakters in beslag neemt is voor dit formaat gekozen. Bij XML is namelijk een afsluitend karakter (vaak inclusief tag) benodigd in tegenstelling tot bij JSON. Daarnaast wordt JSON ook door het gehele SEF[11] systeem gebruikt waardoor het niet meer dan logisch is om dit ook voor deze applicatie te gebruiken. Om ervoor te zorgen dat bij elke response van de API een en dezelfde vorm van informatie wordt teruggestuurd zullen we

---

[9] Bron: https://developer.apple.com/library/ios/documentation/general/conceptual/devpedia-cocoacore/KeyValueCoding.html (Zie bijlage K)

[10] Bron: https://developer.apple.com/library/ios/documentation/general/conceptual/devpedia-cocoacore/Delegation.html (Zie bijlage K)

[11] SEF

gebruik maken van JSend[12]. JSend is een wijze van informatie terugsturen waarbij altijd een status waarde wordt teruggestuurd, daarnaast bepaald de status of het data veld gevuld is en of er een fout is opgetreden. Door gebruik van JSend kunnen we eenvoudig een implementatie maken voor de mobiele applicatie die de informatie kan verwerken. Dit object zal dan altijd een status en andere waarden bevatten die qua vorm en opbouw voor elk verzoek hetzelfde is.

Om JSON te verwerken tot een leesbaar object wordt gebruik gemaakt van een object dat standaard in het Framework zit, namelijk een *NSJSONSerialization* object. Dit object controleert of de informatie de juiste structuur bevat om als JSON object terug te geven. Wanneer dit succesvol is uitgevoerd kan de geladen informatie in de tabel voor de gebruiker getoond worden waarna een keuze gemaakt kan worden voor een afspraak.

Conclusie

Tijdens de Sprint review is geconcludeerd dat alle doelen van deze Sprint zijn behaald. Er is namelijk een Walking Skeleton opgeleverd waarmee een reparatieverzoek kan worden ingediend inclusief beschrijving en gekozen datum. Er werd hierbij wel aangegeven dat het minder vestandig was van de afstudeerder om al zo diep in de code te duiken, dit was op het moment nog niet nodig. Bij de Scrum ontwikkelmethode is het namelijk belangrijk dat eerst functionaliteit opgeleverd wordt, waarna gericht kan worden op de code om dit netjes, uitbreidbaar en efficiënt te maken. Dit onderdeel kan verbeterd worden, daarom is tijdens de Sprint retrospective tijd besteed aan hoe dit probleem opgelost kan worden. Hier kwamen we tot de conclusie dat het handig is om net als de andere ontwikkelaars van Solware gebruik te maken van een intern programma dat allerlei taken (User Stories, functionele eisen) beheerd. Hierbij voeg je taken toe en bepaal je hoeveel uren je hier verwacht mee bezig te zijn. Al deze taken worden uitgeprint en beheerd via een zogenaamd fysiek Scrum bord. Op deze manier weet je precies welke taken je moet maken voor de huidige Sprint en in welke fase deze taak zich bevind. Zo behoud je meer overzicht van je doelen en hoef je dus niet af te dwalen met het maken van onnodige features voor de huidige Sprint. Het *SBOperationManager* systeem had in deze sprint nog niet ontwikkeld hoeven worden.

Uit de eerste Sprint is geleerd dat functionaliteit boven onnodige features staat, dit klinkt al logisch maar is een belangrijk punt waar op gelet moet worden om het project succesvol af te kunnen ronden.

[12] Bron: http://labs.omniti.com/labs/jsend/

2. Synchronisatie & opslag

De tweede Sprint van dit project zal zich vooral richten op het synchroniseren van informatie met de API. Deze synchronisatie bestaat uit het binnenhalen van nieuwe gegevens en deze lokaal op te slaan zodat de informatie niet constant opnieuw opgehaald hoeft te worden.

Sprint Planning Meeting

Voor de tweede Sprint is besloten om de volgende onderdelen in de mobiele applicatie te gaan verwerken, deze onderdelen zijn een deel van de Sprint Backlog:

- Het toevoegen van naam, email en telefoongegevens
- De gebruiker een defect laten kiezen
- Synchronisatie van de defect gegevens
- Het aanvragen van datums die mogelijk zijn om ingepland te kunnen worden en deze aan de gebruiker tonen.

De verwachting van de Sprint is het opleveren van een mobiele applicatie waar de gebruiker zijn gegevens kan invullen en aan de hand van verschillende keuzes een defect en datum kan uitkiezen. Daarnaast zal een onzichtbaar onderdeel toegevoegd worden namelijk de synchronisatie van informatie.

Belangrijk om te weten is dat tijdens deze Sprint gebruik zal worden gemaakt van het interne systeem van Solware voor Scrum projecten. Via dit systeem zullen de functionele eisen ingevoerd worden waardoor per Sprint een Backlog gemaakt wordt met het aantal uren dat geschat is voor elke eis / taak. In totaal moet het aantal uren dat besteed wordt aan deze taken ongeveer gelijk zijn aan 70% van het totaal aantal uren dat mogelijk besteed kan worden in het project voor een bepaalde Sprint. In dit geval is dit 80 uur per Sprint aangezien er fulltime 2 weken per Sprint besteed wordt. Dit komt neer op ongeveer 56 uur, de rest van de tijd kan gestoken worden in niet ontwikkel gerelateerde zaken zoals review of planning. Dit aantal zal in dit geval hoger worden ingeschat dan 70% omdat niet bij alle ontwikkel- gerelateerde onderdelen aangeschoven hoeft te worden. Het aantal uren zal daarom per Sprint rond de 65 uur worden geschat. Om de uren die in het project gestoken wordt precies bij te houden wordt gebruik gemaakt van een uren registratie applicatie genaamd Harvest[13].

[13] Bron: http://www.getharvest.com/

BC-500: [OA] Lijst van Elementen [IOS]	5:00
BC-501: [OA] Lijst van ruimtes [iOS]	5:00
BC-502: [OA] Lijst van gebreken [iOS]	5:00
BC-503: [OA] Tonen van eigen agendapunten in de datumselectie.	8:00
BC-504: [OA] Verstuurde dagdeel en datum controleren p beschikbaarheid op platform.	8:00
BC-505: [OA] Documenteren nieuwe API functies.	3:00
BC-506: [OA] Afstudeerverslag bijwerken	8:00
BC-507: [OA] Opslaan van ontvangen elementen, ruimten en gebreken in CoreData. [iOS]	13:00
BC-508: [OA] Ontwikkelen van een CoreData model	2:00
BC-517: [OA] Volgorde verbeteren van de gebrek selectie flow.	5:00
BC-509: [OA] Updaten van bestaande CoreData objecten.	5:00
BC-519: [OA] Naam, email en telefoonnummers toevoegen aan een defect om te uploaden	5:00
BC-510: [OA] Afstudeerbezoek 18 Juli	4:00

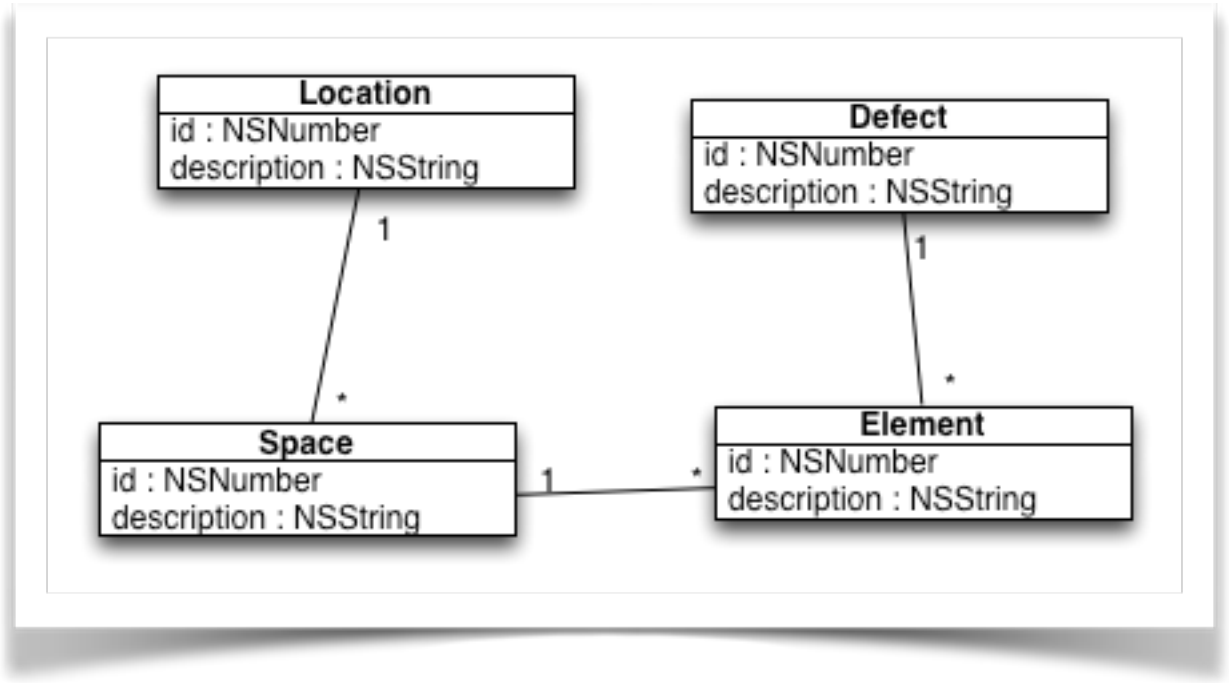
Figuur 7: Planning Sprint 2 / Backlog. (Zie bijlage E).

In figuur 7 (Zie bijlage E) is de volledige planning van deze Sprint beschreven. In totaal zijn 76 uren geschat voor het uitvoeren van deze Sprint. Dit getal is hoger dan de 65 uur die ongeveer per Sprint gerealiseerd kunnen worden omdat er tijdens het ontwikkelen een aantal kleinere tickets bij zijn gekomen, zoals de ticket “Volgorde verbeteren van de gebrek selectie flow”. Deze ticket is aangemaakt na aanleiding van een daily scrum waaruit de conclusie kwam dat de tot nu toe gemaakte keuze schermen niet voldeden aan de eisen van de opdrachtgever. Daarnaast is er bij gekomen dat het afstudeerverslag bijwerken ook een ticket dient te zijn.

CoreData

CoreData[14] stelt de ontwikkelaar in staat om objecten op te slaan op de harde schijf (Flash geheugen) van de mobiele telefoon. Om hier gebruik van te kunnen maken moet een model worden bedacht waarin wordt beschreven hoe de informatie zal worden opgeslagen. Dit is gelijk aan een normaal database ontwerp.

De volgende stap tot het gebruik maken van CoreData is het ontwikkelen van een CoreData model. Dit model beschrijft welke data een object hoort op te slaan en hoe het object in relatie staat tot andere objecten. Van dit model zijn uiteindelijk twee versies gemaakt aangezien de objecten die opgeslagen dienen te worden zijn veranderd na een gesprek met de opdrachtgever. Zo is in de nieuwe versie een extra object toegevoegd. Dit nieuwe object is het *OM_Location* object en staat bovenaan de hiërarchie die het mogelijk maakt om een defect te kiezen. Van elk van deze objecten zal een beschrijving moeten worden opgeslagen en een integer die het uniek maakt. Uiteindelijk is het CoreData ontwerp gemaakt die te zien is in figuur 8 (Zie bijlage L).



Figuur 8: CoreData ontwerp. (Zie bijlage L)

Dit ontwerp wordt in het ontwikkel programma Xcode gebouwd. Hierdoor heb je de mogelijkheid om automatisch klassen aan te maken van het gecreëerde model, wat het implementeren van dit onderdeel vereenvoudigt. Een gedetailleerde beschrijving van de werking van CoreData is terug te vinden in de bijlagen. Ook verwijs ik u voor meer informatie over dit onderdeel naar de documentatie van Apple over CoreData.

[14] Bron: https://developer.apple.com/technologies/mac/data-management.html (Zie bijlage K).

Figuur 9: Voorbeeld overzicht van opgeslagen objecten.

Figuur 9 toont de specifieke *Element* objecten die onder een zojuist gekozen *Space* object gedefinieerd staan. Vier identieke stappen die elk een andere inhoud hebben vormen de keuze tot een uiteindelijk *Defect* object. Dit object wordt gebruikt om het verzoek te versturen en het defect te identificeren.

Voor een gedetailleerde uitleg van de code die bovenstaande afbeelding toont verwijs ik naar de code in bijlage H. In deze bijlage is tevens de volledige synchronisatie code terug te vinden.

Implementatie synchroniseren

Er is gekozen om de synchronisatie door middel van een enkel API verzoek af te handelen. Alle informatie die benodigd is om verzoeken te versturen zal dan worden gesynchroniseerd tussen de mobiele applicatie en API. In de uiteindelijke synchronisatie zal een tijdstip worden meegestuurd.

Dit tijdstip representeert het laatste moment dat de mobiele applicatie succesvol informatie met de API heeft gesynchroniseerd. Als er geen nieuwe informatie beschikbaar is zal de API geen data terugsturen en hoeft er ook geen actie te worden ondernomen. Wanneer er wel nieuwe informatie beschikbaar is wordt alle informatie verstuurd die op dat moment in de database staat. Er is voor gekozen om de gehele CoreData database te legen voordat de nieuwe informatie wordt weggeschreven. Op deze manier is het synchroniseren van informatie vereenvoudigd aangezien er geen bestaande waardes of relaties tussen objecten hoeven te worden aangepast.

Het object dat de synchronisatie afhandelt is het *SBSynchronizeOperation* object. Dit object kan eenvoudig worden toegevoegd aan de *SBOperationManager* door het basis operatie object te overerven. De taken van dit object zijn het binnenhalen van nieuwe informatie en het wegschrijven hiervan. De gehele synchronisatie wordt in de achtergrond uitgevoerd omdat deze taak over een normale internet verbinding soms meer dan 5 seconden kan duren. Het is niet de bedoeling om de gebruiker zo lang te laten wachten voordat de applicatie volledig gebruikt kan worden. Wanneer alle informatie in de achtergrond is opgeslagen zal door middel van een notificatie de applicatie op de hoogte worden gebracht van de zojuist ingeladen objecten. Deze nieuwe objecten kunnen dan getoond worden aan de gebruiker om tot een uiteindelijke defect keuze te komen.

Het tonen van deze objecten gebeurt aan de hand van een *NSFetchedResultsController* die zich bevindt in het standaard Cocoa Touch Framework. Dit object beheert opgeslagen CoreData objecten en verstuurt delegatie berichten naar een luisterend object wanneer nieuwe objecten worden toegevoegd aan de bestaande collectie van objecten. Door middel van dit object kan dus een altijd up to date overzicht worden getoond van opgeslagen CoreData objecten. Dit object zal gebruikt worden voor elke stap die de gebruiker een keus laat maken tussen een *Location*, *Space*, *Element* en *Defect*. Omdat voor elk object een nagenoeg identiek overzicht wordt getoond is dit een mogelijkheid om een generiek object te maken. Dit generieke object wordt aangemaakt met een altijd verschillend type object dat ingelezen moet worden. Zo is de inhoud van het object anders voor een *Location* dan voor een *Defect*. Uiteindelijk is een zogenaamd *SBCollectionController* object geïntroduceerd die dit allemaal voor zijn rekening neemt.

API Database

De locaties die opgeslagen zullen worden in de mobiele applicatie worden via de API opgevraagd. Dit betekent dat de objecten ook in de API database moeten worden opgeslagen. In de vorige Sprint is al rekening gehouden met de toevoeging van deze objecten in de database. Het database ontwerp van de vorige Sprint zal daarom hergebruikt worden, dit ontwerp zal wel uitgebreid moeten worden met de Locatie tabel.

Een tweede uitbreiding die op de API database zal moeten worden uitgevoerd is de ondersteuning van beschikbare datums en het zoeken hierop. Deze uitbreiding is uiteindelijk uitgevoerd door twee nieuwe tabellen te introduceren, namelijk een tabel waar alle dagdelen beschreven zijn en een tabel waar per datum de beschikbare dagdelen beschreven staan. Uiteindelijk wordt een structuur opgeslagen met een tabel waar meerdere keren een datum kan voorkomen, maar het dagdeel en aantal beschikbare medewerkers bij elk van deze datums kan anders zijn. Zo wordt het mogelijk gemaakt keuzes te maken op een datum in de vorm van “21 oktober 2013 - (9:00 - 10:00)”. Op moment van schrijven zijn 6 verschillende dagdelen gedefinieerd in de database. Wanneer een bepaalde datum gekozen wordt bij een reparatieverzoek zal de gekozen datum record gekoppeld worden aan het verzoek. Op deze manier kan rekening worden gehouden met het aantal verzoeken die reeds geplaatst zijn op een datum / dagdeel combinatie.

Datum selectie

Wanneer de gebruiker een datum / dagdeel combinatie wil selecteren moeten eerst alle beschikbare opties vanuit de API worden ingeladen. Een dergelijke combinatie is beschikbaar wanneer er voldoende medewerkers beschikbaar zijn voor het aantal reparatieverzoeken die zijn

geplaatst. Daarnaast moet de datum zich minimaal 3 dagen in de toekomst bevinden. Ook wordt een maximale datum opgegeven welke nog niet gespecificeerd is door de opdrachtgever. Deze minimale en maximale datums zijn in overleg met BouwMeester bepaald waardoor het bedrijf tijd heeft om de opdracht te verwerken. Daarom wordt in dit geval een datum van maximaal 7 dagen in de toekomst gehanteerd. Aan de API kant kan eenvoudig een MySQL query worden uitgevoerd om datums op te halen die voldoen aan bovenstaande zoekcriteria.

Omdat via dezelfde API methode de datum informatie al werd ingeladen vanuit de mobiele applicatie zijn weinig aanpassingen nodig om deze informatie correct te tonen in het overzicht. Een aanpassing die voor deze Sprint wordt toegevoegd is het tonen van agenda punten van de gebruiker in dit overzicht (zie bijlage E). Deze feature maakt het voor de gebruiker gemakkelijker om overlappende agenda punten te zien op bepaalde mogelijke datum / dagdeel combinaties. Dit is in eerste instantie niet besproken tijdens de initiële sprint planning maar als handigheid toegevoegd vanuit de visie van de afstudeerder. In de conclusie wordt beschreven wat voor effect dit heeft gehad op de uiteindelijke sprint review.

Het overzicht van datum / dagdeel combinaties is nu geïmplementeerd zodat een gebruiker twee keer moet tappen voordat een uiteindelijke keuze is gemaakt. Hiermee wordt bedoeld dat de gebruiker eerst een datum kiest waarna het dagdeel op die datum wordt gekozen. Om het de gebruiker makkelijk te maken wordt in eerste instantie geen overvolle lijst getoond maar alleen een lijst met datums die per dag uitgevouwen kan worden voor de dagdelen.

Conclusie

Sprint 2 heeft een applicatie opgeleverd waarmee men kan kiezen tussen verschillende defecten. Dit wordt behaald aan de hand van 4 keuzes, namelijk een locatie, ruimte, element en het uiteindelijke defect om het uiteindelijke onderwerp te kunnen identificeren. De informatie die benodigd is om deze keuze te maken wordt altijd ingeladen wanneer de applicatie opgestart wordt. Tijdens de review is commentaar geleverd op het onderdeel dat agendapunten toont tussen de te selecteren datums. Dit onderdeel was voor het moment niet belangrijk en hoort niet tot een Agile ontwikkelmethode volgens het team van Solware. Features zoals de overlappende agendapunten zouden pas in een latere Sprint mogen worden toegevoegd omdat het niet tot de functionele eisen behoort van de Product Backlog.
Tijdens de Sprint planning is ervoor gekozen om het synchroniseren later toe te passen omdat dit voor nu geen vereiste is aangezien de functionaliteit voor het maken van keuzes er is. Het meer efficiënt maken en versnellen van opstarten zal dus in een latere Sprint opgepakt worden.

Tijdens deze Sprint is gebruik gemaakt van de AgileView applicatie van Solware, door middel van deze applicatie kunnen eenvoudige taken van het project worden bijgehouden en kan de ontwikkelaar ervoor zorgen dat er niet meer uren in een onderdeel wordt gestoken dan nodig. Deze methode is erg handig als er goed gepland is met een reëel aantal uren.

3. Foto’s & REST API

De derde Sprint van het project zal in het teken staan van het verwerken van afbeeldingen voor een reparatieverzoek en het voor een groot deel implementeren van de REST API voor alle informatie voorzieningen. Het doel van deze Sprint is om het mogelijk te maken aan het einde van de Sprint een afbeelding toe te kunnen voegen aan het reparatieverzoek en het juist implementeren van de API door gebruik te maken van SEF.

Sprint Planning Meeting

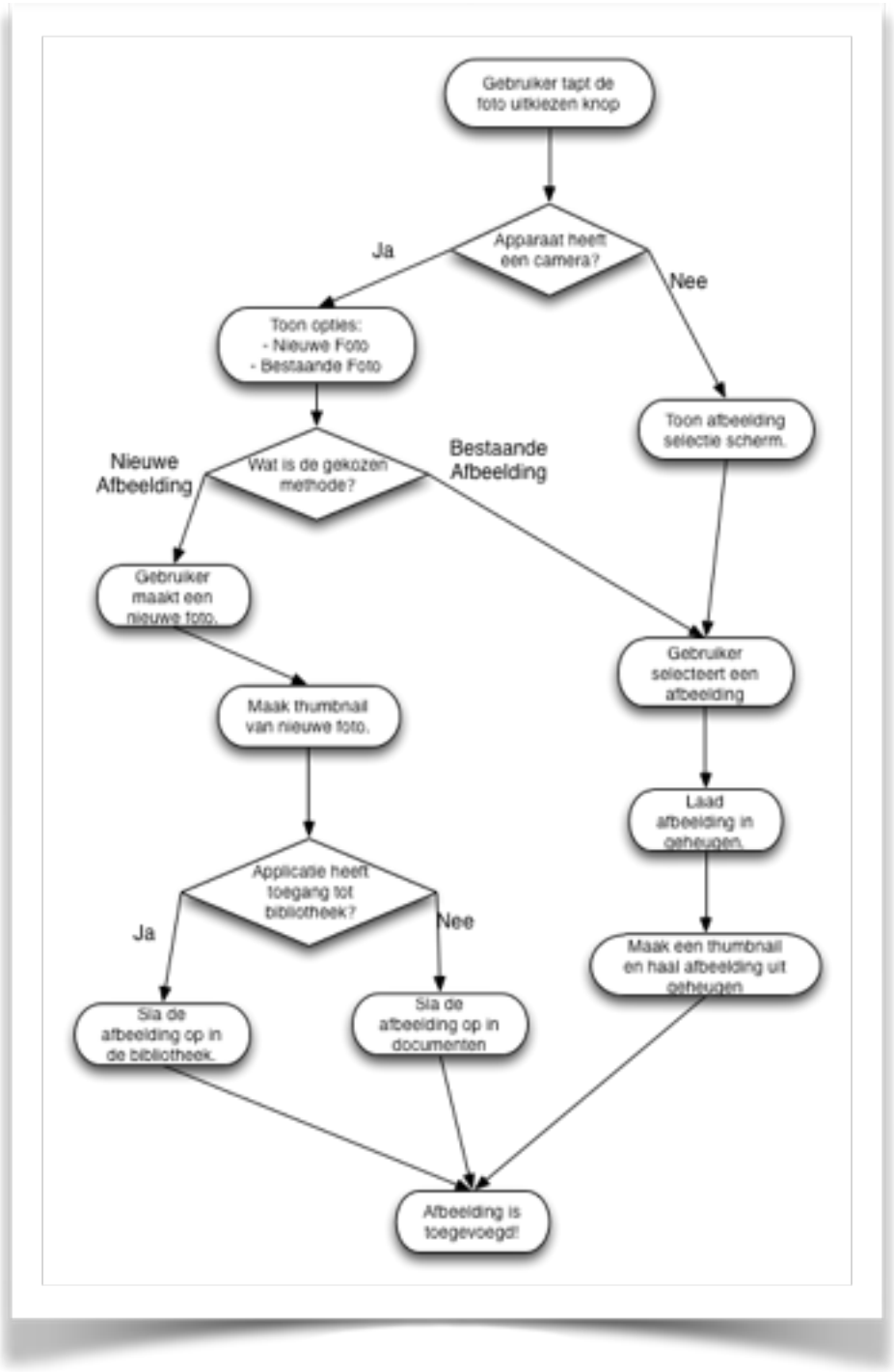
In Sprint 3 is het de bedoeling om de basis functionaliteit van de applicatie af te ronden, dat wil zeggen het toevoegen van onderstaande functies. Met als hoofd functionaliteit het toevoegen van een afbeelding, waar in dit hoofdstuk veel aandacht aan wordt besteedt. Door implementatie van deze functies kan een gebruiker alle benodigde informatie aan een reparatieverzoek toevoegen om zo een defect in te plannen voor reparatie.

- Het toevoegen en uploaden van een gemaakte of gekozen afbeelding.
- Het toevoegen van gebruiker informatie (Naam, Adres, Email etc.)
- Het toevoegen van een woningcorporatie aan een gebruiker.
- De API implementeren op de juiste wijze door gebruik te maken van het REST Framework van SEF.
- Het versturen van een notificatie naar de gebruiker door middel van de Apple Push Notification Service (APNS)
- Het controleren van ingevoerde adresgegevens via de API.

De meeste tijd zal worden gestoken in het gehele traject om een afbeelding te selecteren, te verwerken en te versturen naar de API. In de Sprint Backlog is te zien dat voor deze verschillende onderdelen respectievelijk 5 tot 8 uur is geschat. Voor het versturen van de notificaties naar een gebruiker is in totaal 13 uur gepland aangezien dit ook uit een aantal onderdelen bestaat. Zo moet dit gedeeltelijk op de mobiele applicatie worden geïmplementeerd als in de API. In totaal zal deze Sprint 59 uur in beslag nemen volgens de planning. Dit is minder dan de aangeraden 65 uur maar geeft wel de mogelijkheid voor extra ruimte mochten onderdelen niet in de geplande tijd worden afgerond.

Afbeeldingen

In dit onderdeel wordt aangetoond hoe het proces van afbeelding toevoegen tot verwerking wordt uitgevoerd. Aan de hand van het flow diagram in figuur 10 (Zie bijlage M) zal een eerste indruk worden gegeven over de werking van het afbeelding toevoegen.



Figuur 10: Afbeelding toevoegen flow. (Zie bijlage M).

Afbeeldingen nemen vaak een groot gedeelte van het geheugen in beslag op mobiele telefoons. Zeker in het geval van de moderne smartphones worden afbeeldingen van rond de 8 MegaByte gemaakt. Wanneer deze afbeeldingen in het geheugen worden gehouden totdat ze verstuurd worden kan het in uiterste gevallen gebeuren dat geheugen vol raakt. Aangezien het een belangrijk doel is van het project om een efficiënte applicatie te ontwikkelen moeten we deze afbeelding zo min mogelijk geheugen laten gebruiken. Om twee redenen is gekozen om de afbeelding te verwerken tot een kleinere versie, de opdrachtgever wil afbeeldingen van maximaal 1000 pixels breed of hoog in zijn beheersysteem te zien krijgen. Daarnaast verbruikt een verkleinde afbeelding veel minder geheugen in vergelijking met de originele afbeelding. Wat op zijn beurt voor een versnelde upload connectie zorgt als de gebruiker een reparatieverzoek inclusief afbeelding verstuurt.

Zoals in bovenstaande figuur is te zien kan een gebruiker op twee manieren een afbeelding aan het reparatieverzoek toevoegen. Dat is door middel van een reeds bestaande afbeelding uit de lokale bibliotheek kiezen of door een nieuwe afbeelding te maken. Van deze afbeelding wordt een verkleinde versie gemaakt die tijdelijk in het geheugen gehouden zal worden totdat het verzoek verstuurd is. Daarna wordt de afbeelding verwijderd, dat betekent dat de verkleinde versie opnieuw gemaakt zal moeten worden. Dit is op dit moment een probleem maar zal als feature in de latere versies lokaal worden opgeslagen zodat deze versie van de afbeelding altijd beschikbaar is.

Om een afbeelding te maken of te kiezen dient gebruik gemaakt te worden van het standaard Framework object *UIImagePickerController*. Dit object kan zo geconfigureerd worden om de gebruiker een bestaande of nieuwe afbeelding te laten kiezen. Aan de hand van dit object krijgen we toegang tot de gekozen afbeelding die we op zijn beurt tot een verkleinde versie kunnen bewerken. Deze bewerking wordt uitgevoerd door een subklasse van het *SBOperation* object en wordt dus zoals eerder genoemd ook aan het object toegevoegd die aan de hand van systeem specificaties bepaald of de operatie kan starten. Hiervoor is gekozen omdat het verwerken van een afbeelding van 8 MegaByte relatief lang kan duren waardoor de gebruikers interface vast zou lopen mocht dit op de main thread worden uitgevoerd. Als de operatie klaar is wordt een delegatie bericht naar het luisterende object verstuurd met de verkleinde afbeelding als argument. Deze afbeelding wordt dan aan het reparatieverzoek toegevoegd door middel van het Wrapper object *SBDefectImageDescriptor*.

Versturen van een afbeelding over HTTP(S)

Om de afbeelding te versturen over een HTTP verbinding zal het POST bericht dat verzonden moet worden aangepast worden. De bytes van de afbeelding moeten namelijk aan dit verzoek gekoppeld worden, daarnaast moet ook rekening gehouden worden met andere POST variabelen die aan het verzoek gekoppeld kunnen worden. Volgens het RFC[15] (Request For Comments) van de IETF[16] (Internet Engineering Task Force) moet volgens[17] een dergelijk verzoek een zogenaamd boundary tekst gebruikt worden om waardes af te scheiden. Zo kan een verschil gemaakt worden tussen de beschrijvende tekst en de afbeelding bytes. Om dit te implementeren wordt de MIME type van een reparatieverzoek die via HTTP verstuurd wordt als “multipart-form-data” beschreven. Zo kan per variabele die verstuurd moet worden een zogenaamd “Content-Disposition” header worden meegegeven die exact beschrijft wat voor type waarde verwacht kan worden in de opvolgende bytes. Zo kan je bijvoorbeeld een bestand beschrijving met het MIME-type “image/jpeg”, wat een JPEG afbeelding beschrijft.

Een verzoek ziet er in de volgende vorm uit voor een reparatieverzoek, tussen de { en } wordt een waarde verwacht die door de applicatie wordt ingevuld.

> ***--{BOUNDARY}***
> ***Content-Disposition: form-data; name=”{POST_NAME}”***
>
> ***{POST_VALUE}***
> ***{BOUNDARY}***

Adresgegevens

[15] Bron: http://www.ietf.org/rfc.html

[16] Bron: http://www.ietf.org

[17] Bron: http://www.ietf.org/rfc/rfc2388.txt (Hoofdstuk 4.1)

Om het gebruikers te vereenvoudigen een adres toe te voegen is een onderdeel ingebouwd in de API en mobiele applicatie die automatisch de straatnaam en woonplaats opzoekt van de ingevoerde postcode / huisnummer combinatie. Op deze manier hoeft de gebruiker niet handmatig een straatnaam en woonplaats toe te voegen en zal het invullen van de persoonlijke gegevens versneld worden.

Voor deze feature moet aan de API zijde een koppeling geïmplementeerd worden met de zogenaamde postcode database van een Nederlandse postcode API[18]. In deze database staan alle meest recente postcode combinaties met allerlei extra informatie zoals geo locatie. Deze informatie zou later gebruikt kunnen worden voor extra functionaliteit. Er is voor gekozen om de informatie die van de API geladen worden tijdelijk te cachen zodat deze server niet overbelast raakt met verzoeken. Het is tijdens het testen namelijk een aantal keer voorgekomen dat het verzoek naar deze server meer dan 20 seconden duurde. De oorzaak hiervan is niet duidelijk maar toont aan dat deze API qua snelheid niet altijd even betrouwbaar is.

Apple Push Notification Service (APNS)

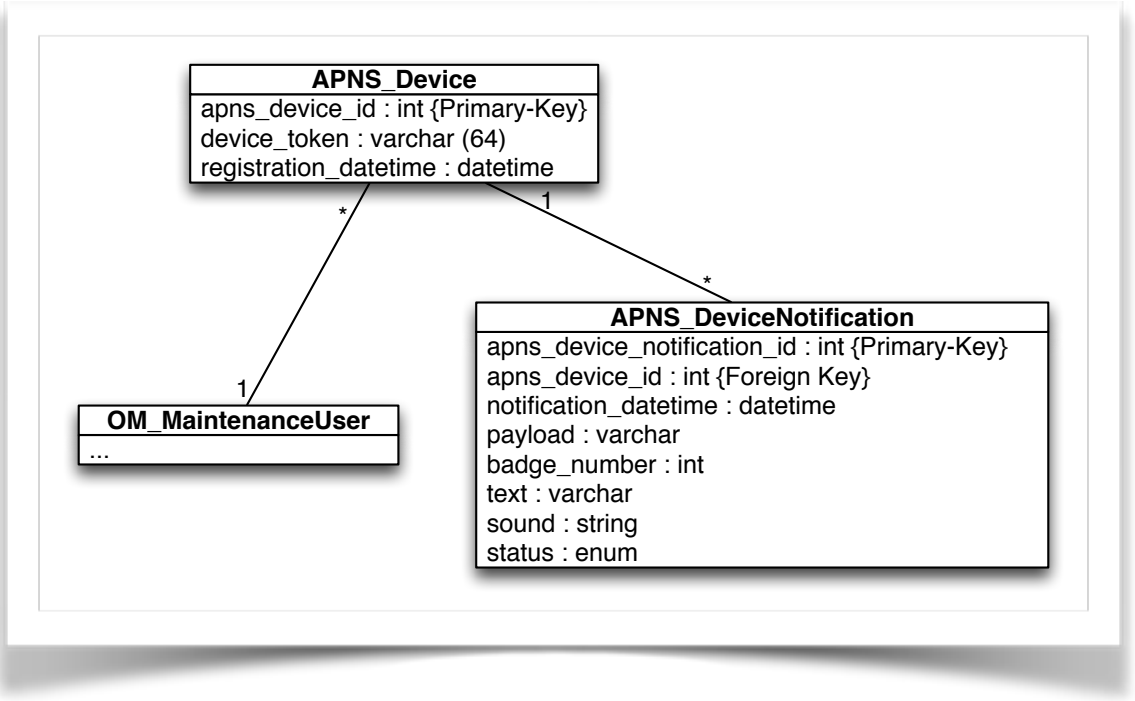
De Apple Push Notification Service[19] (APNS) is een service verzorgd door Apple die het mogelijk maakt notificaties te versturen naar mobiele telefoons die op het iOS platform draaien. Een dergelijk apparaat staat ten alle tijden in verbinding met de Apple servers wanneer mogelijk om deze notificaties te ontvangen. Wat betekent dat er maar een enkele manier is om naar idle mobiele telefoons een bericht te sturen. De notificaties die we willen sturen hebben als doel om de gebruiker op de hoogte te stellen van status veranderingen van het reparatieverzoek.

De APNS kan gebruikt worden door middel van ontwikkelaar certificaten. Met gebruik van deze certificaten kan een connectie worden geopend met de Apple servers om notificaties te versturen. Solware heeft al een dergelijke library geschreven die deze connectie aan de hand van certificaten aanmaakt en verstuurd zo de notificatie objecten naar deze server. De verbinding zal altijd beveiligd worden door SSL om zo een versleutelde overdracht van informatie te garanderen. Om notificaties te versturen is de huidige database uitgebreid met een aantal nieuwe tabellen die de notificaties opslaan. Zo wordt het bericht opgeslagen en de unieke code van een mobiele telefoon waar het bericht naar verzonden moet worden. In figuur 11 zijn de uitbreidende tabellen te zien. Hier wordt het unieke token van een mobiel apparaat opgeslagen, dit verzoek kan verkregen worden wanneer een reparatieverzoek verzonden wordt. Het token is namelijk ingebouwd in dit verzoek om hiervoor gebruikt te worden. Het is echter ook mogelijk dat geen token meegestuurd kan worden omdat de gebruiker dit niet toestaat. In dat geval wordt eenvoudig de beslissing genomen om deze gebruiker geen notificaties te versturen. De gebruiker zal wel e-mail berichten ontvangen met de veranderingen van status.

---

[18] Bron: https://www.postcode.nl

[19] Bron: https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html



Figuur 11: Notificatie tabellen voor de uitbreiding van de database.

Conclusie

Aan het eind van de Sprint zijn we tot de conclusie gekomen dat er meer features in de applicatie zijn verwerkt dan nodig. Deze features zijn bijvoorbeeld het via de API controleren van de adresgegevens en het implementeren van notificaties. Door het implementeren van deze extra taken zijn veel uren gestoken in de taken waar dit in eerste instantie niet voor bedoeld is. Dit is voor dit project niet een heel groot probleem. Echter voor later in het bedrijfsleven is het een belangrijke les dat je je goed aan de planning en de uren moet houden en je tijd niet aan zelfbedachte taken die niet goedgekeurd zijn besteedt.

Bijvoorbeeld het controleren van adresgegevens vooraf zou niet in deze versie ontwikkeld hoeven zijn. Door deze features zijn er ook meer uren gemaakt dan nodig was. Uiteindelijk zijn er 9 uur meer geboekt op deze Sprint dan in eerste instantie gepland was, wat eigenlijk al verwacht is vanaf het begin aangezien er 6 uur minder dan het normale aantal is ingepland.

In ruil voor de uren die te veel zijn uitgevoerd is er nu een versie van de mobiele applicatie die gebruikt kan worden. Alle functionaliteit die verwacht wordt is nu geïmplementeerd en werkt in combinatie met de API. In de volgende Sprint veel focus worden gelegd op de ondersteuning van iOS 6. Aangezien aan het einde van die Sprint de opdrachtgever zijn mening komt geven is het belangrijk dat alle functionaliteit in de applicatie is verwerkt, en dat dat het vereiste iOS 6 ook ondersteund wordt door de applicatie. Dit is namelijk een belangrijk functionele eis beschreven in de Product Backlog (zie bijlage E).

4. Online beheersysteem

De vierde Sprint wordt uitgevoerd om de volgende twee doelen te behalen, dit zijn de ondersteuning van een vorige iPhone OS versie zodat de applicatie op meerdere versies kunnen draaien. Het tweede doel is een online beheersysteem te ontwikkelen dat gebruikt kan worden door de opdrachtgever om informatie voor en van de mobiele applicatie te beheren. Aan het einde van deze Sprint is een afspraak met de opdrachtgever gepland om een demonstratie te geven van de voortgang. Het doel van deze afspraak is om feedback van de opdrachtgever te krijgen zodat er op een goede wijze verder gewerkt kan worden.

Sprint Planning Meeting

De eerder genoemde doelen zullen vertaald worden in verschillende onderdelen met de bijgevoegde tijdsplanning. Een belangrijk onderdeel is het bedenken en implementeren van een ontwikkel methode die ervoor zorgt dat informatie op verschillende versies van iOS juist getoond kan worden. De versies van iOS die voor dit project ondersteund zullen moeten worden zijn iOS 6.0 en 7.0. Ondersteuning voor een oudere versie van iOS is een vereiste van de opdrachtgever aangezien verwacht wordt dat niet iedereen de nieuwe versie van dit operating system zal hebben.

- Online formulier ontwikkelen voor het beheren van beschikbaarheid & reserveringen.
- De opdrachtgever moet een mogelijkheid krijgen om reparatieverzoeken te annuleren of bevestigen.
- Verstuurde POST informatie valideren in de API voor het toegevoegd wordt aan de database.
- Ontwikkelen en implementeren van een methode die ervoor zorgt dat informatie op alle ondersteunde versies van iOS juist wordt getoond.

Het ondersteunen van meerdere versies van iPhone OS is onderverdeeld in een aantal subtaken. Ten eerste moet het ontwerp worden doordacht voor deze implementatie. Daarnaast moet de implementatie taak worden aangemaakt. In totaal zal dit 13 uur duren, omdat elk scherm dat getoond kan worden moet worden doorlopen en gecontroleerd moet worden zodat het de juiste gebruikers interface getoond wordt. Als laatste zal wederom de applicatie worden doorlopen met als doel om alle niet ondersteunde methodes te verwijderen en hier andere oplossingen voor te vinden voor de betreffende iOS versie.

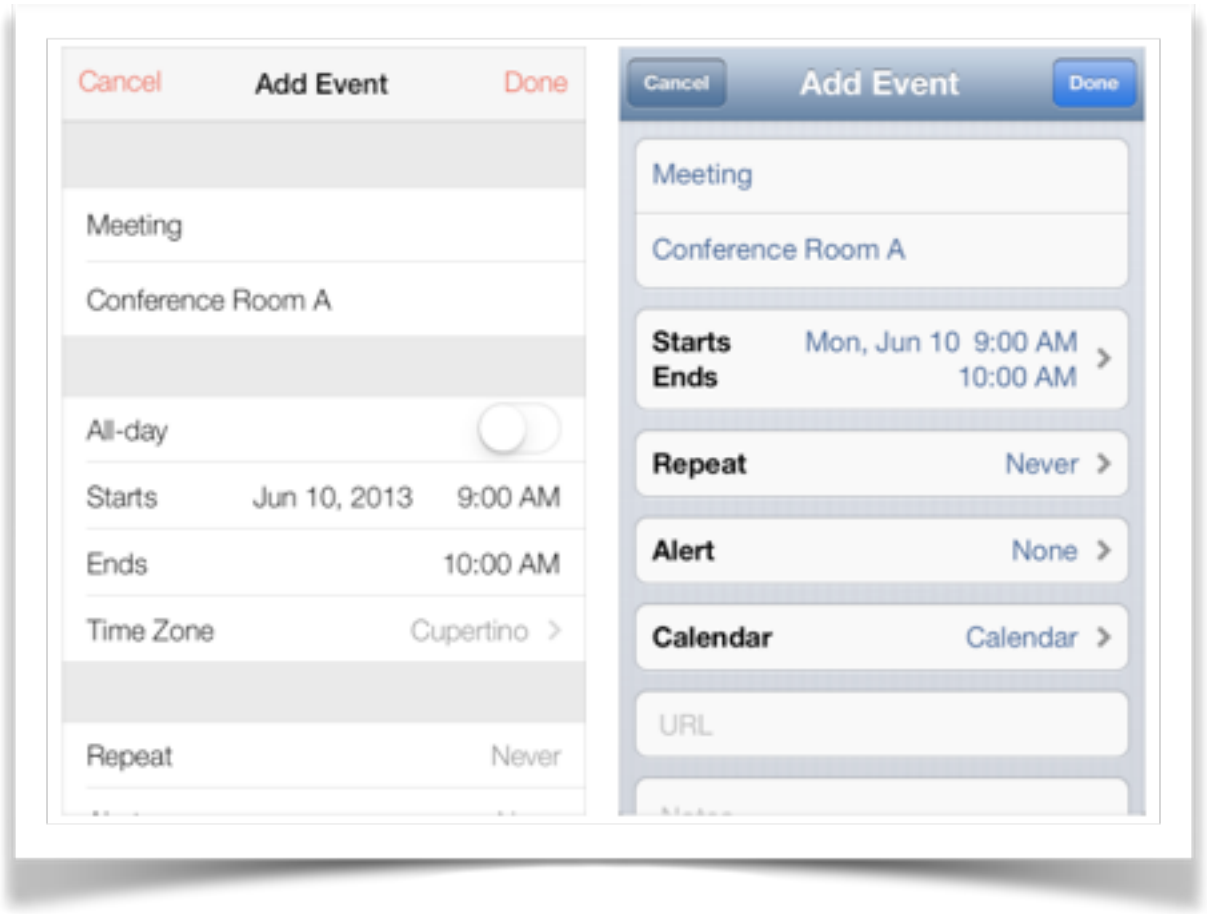
Het tweede belangrijke onderdeel voor deze Sprint is de ontwikkeling van de beheer module voor de opdrachtgever. Met dit systeem zal de opdrachtgever de mogelijkheid krijgen om reparatieverzoeken te beheren, hier hoort het annuleren en goedkeuren bij, maar ook het toevoegen van beschikbare datum & dagdeel combinaties. Dit gehele onderdeel is ook in subtaken opgesplitst en zullen samen in totaal 21 uur in beslag nemen.

Voor een compleet overzicht van de planning voor deze Sprint verwijs ik naar de bijlagen (Zie bijlage E). De nummers van de taken lopen van BC-537 tot en met BC-552.

Versie ondersteuning

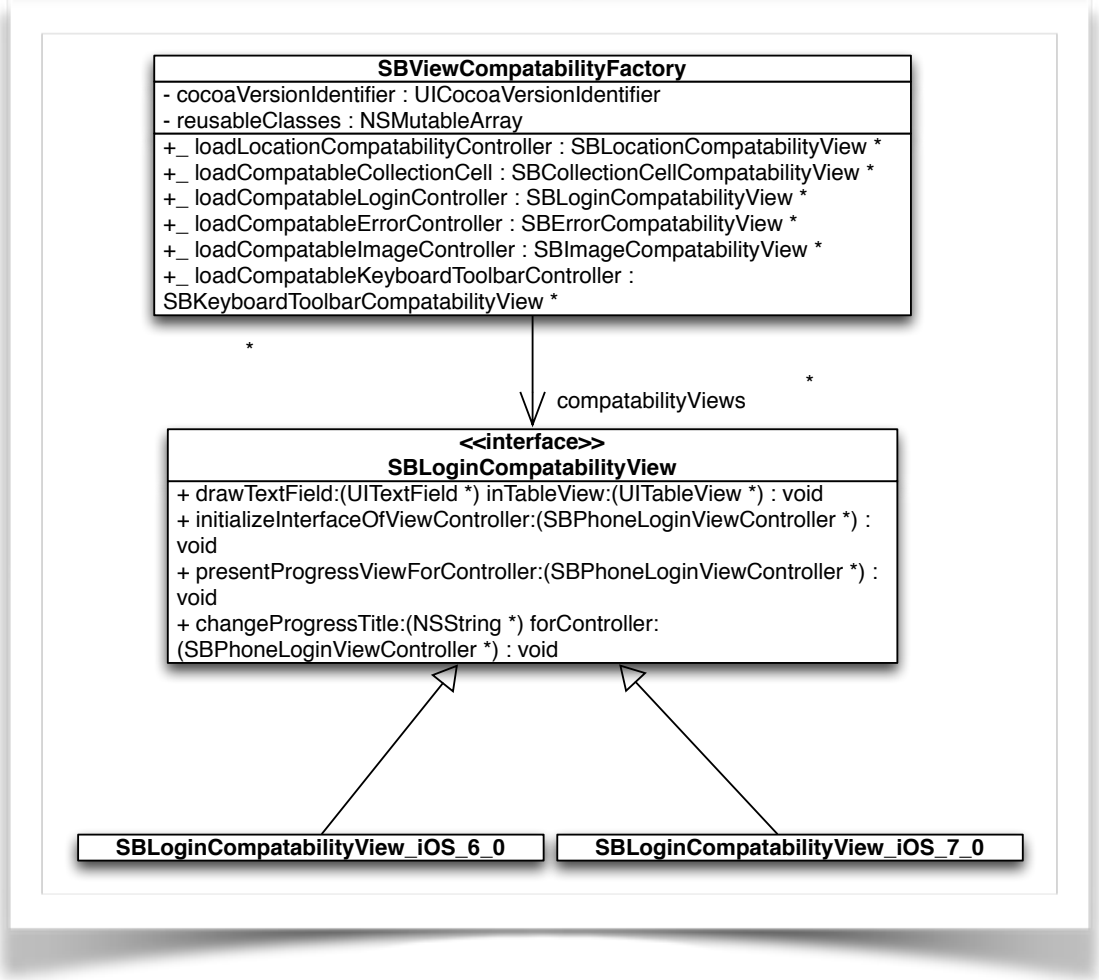
Om een oudere versie van het iPhone OS te kunnen ondersteunen moet rekening gehouden worden met een aantal punten. Een oudere versie heeft namelijk geen beschikbaarheid over methodes die nieuw geïntroduceerd zijn in een nieuwere versie van het OS. Daarnaast heeft een nieuwe versie van het OS soms geen beschikbaarheid over oudere methodes omdat deze “deprecated” kunnen zijn. De methodes zijn dan nog steeds bruikbaar maar de kans is dan groot dat de mobiele applicatie niet wordt goedgekeurd tijdens de validatie door Apple.

Een meer specifiek probleem waar tijdens dit project rekening mee gehouden moet worden is het grote verschil aan gebruiker interface tussen de versies iOS 6.0 & 7.0. Voor het eerst is een grote verandering aan gebruiker interface doorgevoerd wat het ontwikkelaars lastig kan maken om applicaties voor beide versies goed in huisstijl beschikbaar te maken. In figuur 12 is een voorbeeld te zien van de transitie van rechts iOS 6.0 naar links iOS 7.0.



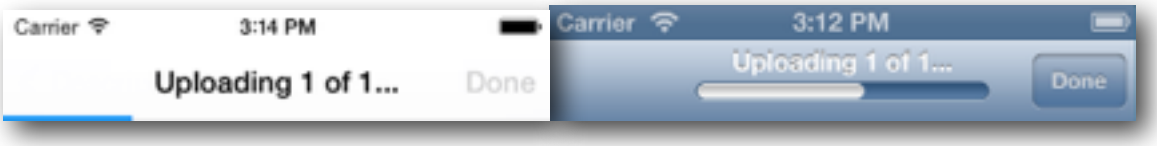
Figuur 12: Interface verschillen, Links: iOS 7. Rechts: iOS 6.

Om beide versies van het iPhone OS te kunnen ondersteunen is een nieuw ontwerp gemaakt van een aantal klassen die dit eenvoudig af kunnen handelen. Er moet voor toekomstige versies rekening gehouden worden dat er wederom een nieuwe implementatie kan worden toegevoegd. Op deze manier kan eenvoudig ondersteuning voor de nieuwere versie van iOS worden toegevoegd. Het ontwerp dat rekening houdt met deze doeleinden is in figuur 13 (Zie bijlage N) weergegeven.



Figuur 13: Ontwerp iOS compatability. (Zie bijlage N)

Het bovenstaande ontwerp wordt herhaald voor elk object dat tekent op het scherm, met de nadruk op de *UIViewController* objecten. Deze objecten vragen aan een factory object een specifiek “Compatability” object aan dat speciaal voor dat object gemaakt is. Zo wordt voor het persoonlijke gegevens invullen het *SBLoginCompatabilityView* aangevraagd. Voor dit object zijn een aantal subklassen gemaakt die specifiek voor een bepaalde iOS versie code implementeert. Een voorbeeld waar dit gebruikt wordt is het tonen van progressie, in beide versies van iOS wordt dit anders getoond door een aangepast interface element (zie figuur 14). In deze situaties komt het bedachte ontwerp goed van pas en kan eventueel uitgebreidt worden voor latere versies van het operating system. De code die de objecten en zijn implementatie aanlevert specifiek voor een operating system is het *SBViewCompatabilityFactory* object.



Figuur 14: Interface verschil tijdens reparatieverzoek versturen.

Een voorbeeld waar deze ontwikkel methode zijn voordelen bewijst is het tonen van de voortgangsbalk wanneer een reparatieverzoek verzonden wordt naar de API. In iOS 7 is de stijl van deze balken veelal over de gehele breedte als standaard beschreven, terwijl dit bij iOS 6 vooral een kortere balk is die anders is vormgegeven. Hoe deze balken er visueel uitzien is terug te vinden in figuur 14.

Online beheersysteem

Een ander gedeelte van het gedistribueerde systeem is het beheer systeem dat gebruikt wordt door de opdrachtgever in de vorm van een web client. Door middel van dit systeem kan de opdrachtgever bepaalde informatie beheren zoals zichtbare informatie in de mobiele applicatie of de status van een door de gebruiker verstuurd reparatieverzoek. Dit betekent dat het beheer systeem een directe verbinding heeft met de database die de mobiele applicatie gebruikt, het enige verschil is dat de mobiele applicatie de API als proxy gebruikt. Belangrijke doelen van dit beheersysteem zijn de volgende:

- Het systeem moet veilig zijn en bestand zijn tegen SQL injectie.
- Het systeem moet lijken op een reeds bestaand systeem dat de opdrachtgever heeft laten ontwikkelen bij Solware.

Dit eerste doel zal behaald worden doordat het standaard Framework van Solware alle gebruikers informatie valideerd en klaarmaakt om verstuurd te worden naar de database. Dat betekent dat deze informatie niet nog eens hoeft te worden gecontroleerd op SQL injectie en andere kwaadaardigheden. Het systeem is op deze wijze al beveiligd op SQL injectie.
Om het systeem te laten lijken op het reeds bestaande systeem zal goed moeten worden gekeken naar de opbouw van dit systeem. Op deze manier is het eenvoudig voor medewerkers die met dit systeem gaan werken om functies te herkennen omdat ze de wijze van beheer structuur herkennen. Schermafbeeldingen of andere voorbeelden van deze bestaande systemen kunnen niet getoond worden vanwege privacy en geheimhouding redenen.

Ontwikkeling

Het ontwikkelen van dit nieuwe beheer systeem zal compleet uitgevoerd moeten worden aan de hand van het SEF Framework. Zoals eerder genoemd is dit het door Solware ontwikkelde Framework, door gebruik hiervan is het eenvoudig om een beheer systeem te maken in een bepaalde structuur die herkenbaar is voor de opdrachtgever. Voor een gedetailleerde uitleg van SEF verwijs ik u naar de bijlagen (Zie bijlage C). In dit onderdeel zal een korte beschrijving worden gegeven over de wijze van implementatie van het beheer systeem.

Door middel van het overerven van het standaard *Formulier* en *ViewController* object is het eenvoudig om specifieke beheer pagina's te maken. Vaak is de standaard implementatie van een formulier voor deze doeleinden voldoende, bijvoorbeeld voor het tonen van een overzicht van reparatieverzoeken. Dit formulier handelt zelf het ophalen van de objecten uit de database af en maakt het ook mogelijk om al deze objecten te wijzigen.

Na de implementatie van dit beheer systeem komt een deel van het gedistribueerde systeem tot stand. Dit systeem bestaat uit een beheersysteem dat via de browser bereikbaar is, een API die bereikbaar is door middel van de mobiele applicatie en eventueel andere methodes die nog niet gedefinieerd zijn. Als laatste bestaat ook een werkende versie van het laatste onderdeel van het systeem, namelijk een mobiele applicatie die door iedereen geïnstalleerd kan worden op een iPhone.

Demonstratie & Review

Figuur 15: Update van het taken bord (User stories) inclusief schermafbeeldingen.

Voor de demonstratie aan de opdrachtgever zijn een aantal voorbereidingen getroffen. Zo is namelijk het taken bord bijgewerkt met nieuwe schermafbeeldingen. De schermafbeeldingen maken de huidige werking en flow van de mobiele applicatie duidelijker voor de opdrachtgever. Tijdens de demonstratie zal de gehele mobiele applicatie worden doorgelopen om het beeld te verduidelijken. Tevens zal de eerste versie van het beheersysteem aan de opdrachtgever worden getoond om feedback te geven.
De demonstratie is een goed moment om voor onbeantwoorde vragen en vage functionele eisen oplossingen te zoeken met behulp van de opdrachtgever.

Review

De demonstratie die voor de opdrachtgever gegeven wordt is voorbereid tijdens de interne review binnen Solware. Hier wordt feedback over geleverd om ervoor te zorgen dat de demonstratie goed zal verlopen. Daarnaast is tijdens de review ook de normale feedback gegeven op de taken die geïmplementeerd zijn tijdens de Sprint. Een belangrijk feedback punt is het feit dat het beheersysteem nog niet voldoet aan de eisen die Solware aan zijn eigen beheersystemen geeft. De wijze hoe een beheerder door verschillende stappen moet lopen om acties te kunnen uitvoeren zoals het verwijderen van een reparatieverzoek. Het team waarschuwde daarom al dat dit waarschijnlijk een groot feedback punt van de opdrachtgever zal zijn en het daarom belangrijk is om extra op voor te bereiden. Daarentegen zag de mobiele applicatie er mooi uit en werkte naar behoren. Op dit punt verwachtte het team juist veel positieve feedback. Enkele benamingen die gebruikt worden in de mobiele applicatie zijn vaag maar zouden nog voor de demonstratie gewijzigd kunnen worden waardoor de applicatie tekstueel duidelijker is. Er wordt verwacht dat op

de mobiele applicatie alleen functionele feedback wordt gegeven voor bijvoorbeeld de wijze van opslag etc.

Demonstratie

Tijdens de demonstratie zijn alle onderdelen die bij de interne review getoond zijn ook getoond aan de opdrachtgever. Zoals al tijdens de review is besproken is erg veel feedback geleverd op het beheer gedeelte van het systeem. Volgens de opdrachtgever was dit nog niet logisch ingedeeld en vergt nog veel aandacht voordat dit als goed wordt beschouwd. De focus zal vooral gelegd moeten worden op het beheren van een individueel reparatieverzoek. Daarnaast was de feedback op de mobiele applicatie inderdaad zoals verwachting minder dan op het beheersysteem. De opdrachtgever was vooral te spreken over de ondersteuning mogelijkheden van zowel iOS 6 als iOS 7. De gehele wijze van reparatieverzoek toevoegen was volgens de opdrachtgever goed, echter miste de applicatie nog een overzicht pagina van het aangemaakte verzoek.

Tijdens deze demonstratie zijn veel feedback punten opgeschreven. Deze punten zullen verwerkt worden tot nieuwe doelen voor de volgende Sprint. De volgende Sprint zal vooral in het teken staan van het verbeteren van het beheersysteem.

Conclusie

Het doel van deze Sprint om ondersteuning te hebben van meerdere iPhone OS versies is behaald. Tijdens de demonstratie kwam al naar boven dat deze feature erg werd gewaardeerd door de opdrachtgever wat een goed gevoel geeft voor de uiteindelijke realisatie van de mobiele applicatie. Daarnaast is een eerste versie van het beheersysteem opgeleverd waar minder positieve feedback over is geleverd. Het doel om dit beheersysteem goed te maken is daarom nog niet in deze Sprint behaald. In de volgende Sprint zullen taken worden aangemaakt die ertoe leiden dat dit doel wel bereikt zal worden.

Concluderend kan gezegd worden dat in deze Sprint niet alle doeleinden behaald zijn, maar er is wel een redelijk positieve indruk achtergelaten bij de opdrachtgever. Dit is een goed teken voor de relatie tussen de klant en Solware.

5. Reparatieverzoek overzicht & Excel

De doelen van de vijfde Sprint zijn het verbeteren van het online beheer systeem voor de opdrachtgever. Dit moet gebeuren aan de hand van de verkregen feedback die in de vorige Sprint is verzameld tijdens de demonstratie. Daarnaast zal de mobiele applicatie worden uitgebreid met de mogelijkheid om verzonden reparatieverzoeken te annuleren en van de telefoon te verwijderen, inclusief meerdere defecten aan een enkel verzoek te koppelen. Door deze implementatie zijn alle acties die een gebruiker zou moeten kunnen uitvoeren toegevoegd aan de applicatie.

Sprint Planning

Bij de Sprint planning is onderscheid gemaakt tussen de de volgende onderdelen die ontwikkeld zullen worden. Deze onderdelen zijn tot stand gekomen aan de hand van de verkregen feedback tijdens de demonstratie aan de opdrachtgever.

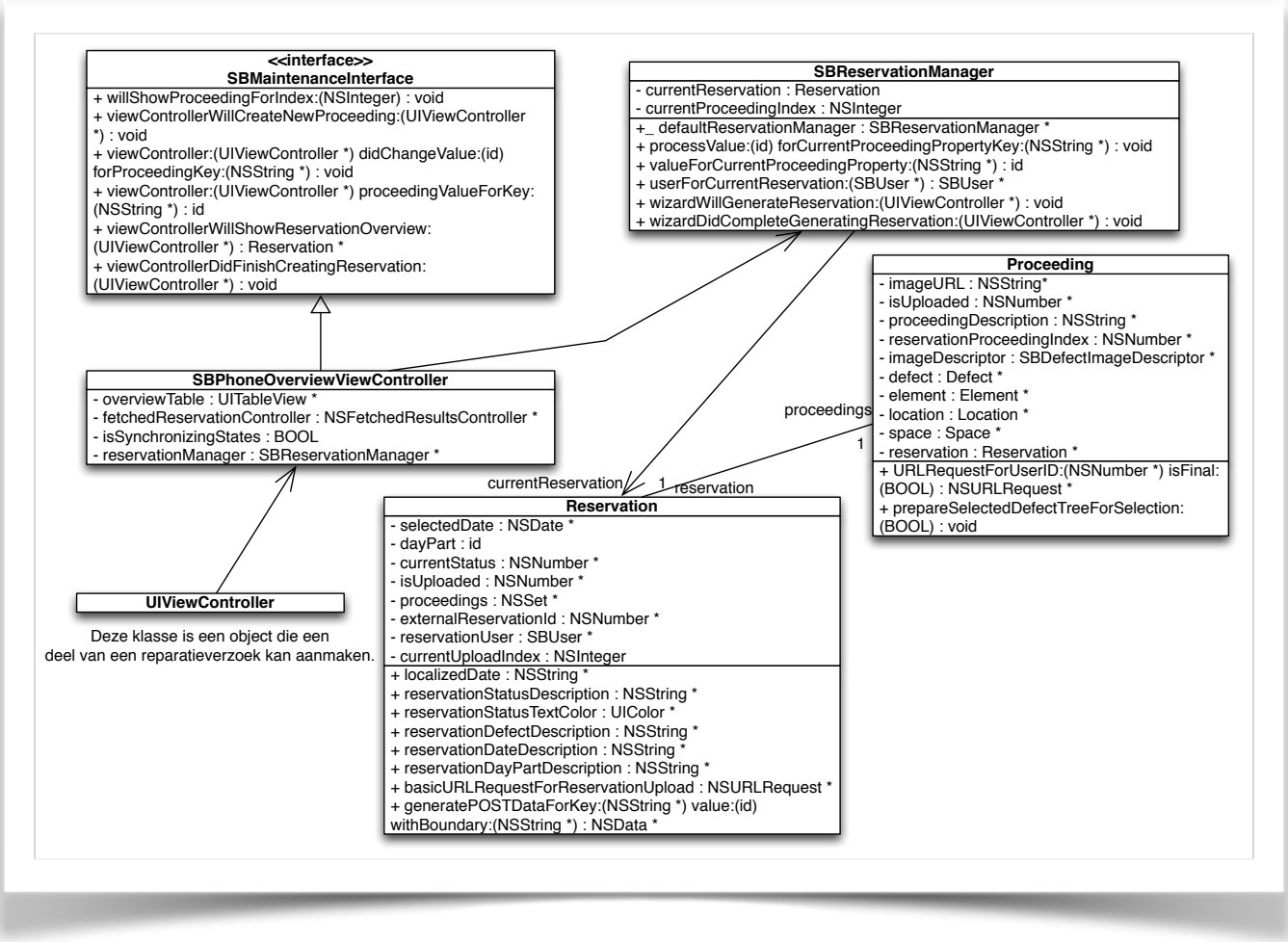
- Een Excel importeer & exporteer methode maken om beschikbaarheid aan te geven in het beheer systeem.
- Het overzicht van reserveringen verbeteren en overzichtelijker maken.
- Op de mobiele applicatie een scherm tonen die de status van alle verzonden reparatieverzoeken toont.
- De mogelijkheid toevoegen aan de mobiele applicatie om reeds bestaande verzoeken te annuleren.

De opdrachtgever zou graag zien dat een Excel importeer & exporteer methode wordt toegevoegd aan het beheersysteem om zo eenvoudig nieuwe datum / dagdeel combinaties beschikbaar te maken voor gebruikers. Naast het onderdeel dat in grote lijnen beschrijft dat het beheer systeem verbetert moet worden neemt naar schatting 20 uur in beslag. In combinatie met alle andere taken die uitgevoerd zullen worden in deze Sprint is er 73 uur geschat aan tijd die nodig is om al deze onderdelen succesvol te kunnen implementeren. Als referentie naar de Sprint Backlog zijn de onderdelen BC-553 tot en met BC-568 te vinden (Zie bijlage E).

Reparatieverzoeken overzicht (iOS)

Een onderdeel dat in de mobiele applicatie geïmplementeerd moet worden is een overzicht van reparatieverzoeken dat de gebruiker kan inzien. In dit overzicht bevinden zich alle reparatieverzoeken die zijn aangemaakt en in welke status ze zich bevinden. Zo is eenvoudig te zien of een bepaald verzoek al goedgekeurd is door de opdrachtgever. Om dit te realiseren zijn een aantal wijzigingen nodig voor de mobiele applicatie zoals hij op het moment geïmplementeerd is. Reparatieverzoeken moeten ook lokaal op de telefoon worden opgeslagen om ervoor te zorgen dat informatie altijd beschikbaar is zonder het op te halen via de API. Dit betekent dat het CoreData model uitgebreid moet worden met reparatieverzoek objecten en meer, hoe dit is gerealiseerd kunt u later in dit hoofdstuk lezen. Er is voor gekozen om extra objecten aan te maken die communicatie over reparatieverzoeken naar een enkele plaats toe leiden. Op deze manier is er een enkel aanspreek punt over informatie van reparatieverzoeken. Hierdoor zou de code beter onderhoudbaar worden en duidelijker te lezen zijn voor onafhankelijke ontwikkelaars omdat alle code die te maken heeft met reparatieverzoeken is samengevoegd in een enkel object.

Ontwerp

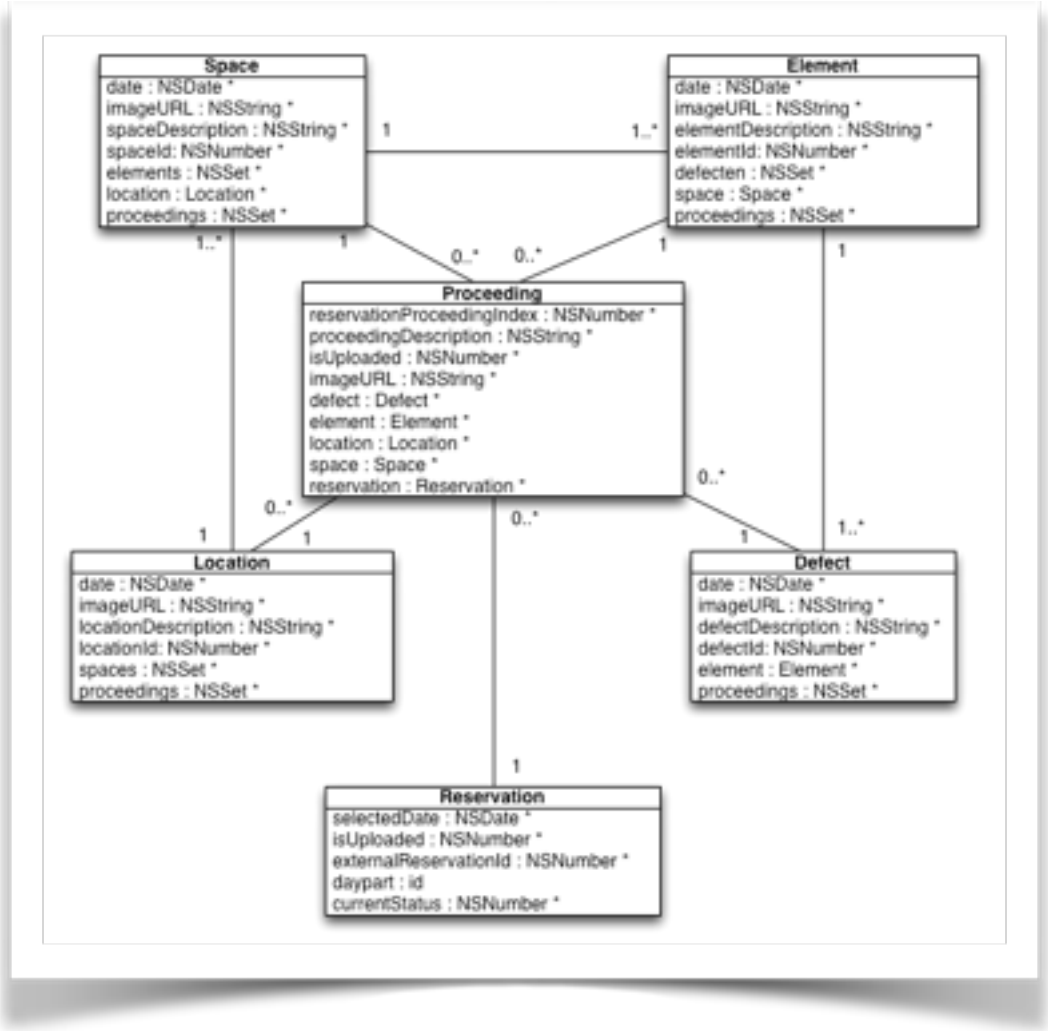


Figuur 16: Diagram van het overzicht en verzoek interne communicatie. (Zie bijlage O).

Bovenstaand diagram toont het nieuwe ontwerp dat geïmplementeerd zal worden voor het overzicht van reparatieverzoeken. Dit ontwerp heeft als extra doel het vereenvoudigen van reparatieverzoeken indienen door alle communicatie naar het *SBReservationManager* object te sturen. Zo zorgt dit object ervoor dat nieuwe objecten worden aangemaakt in de database en dat bestaande waarden worden toegevoegd of gewijzigd. De objecten die vanaf dit moment naar het CoreData model zijn verplaatst zijn de *Reservation* en *Proceeding* objecten. Het object dat alle informatie van reparatieverzoeken verzamelt en doorstuurt naar het manager object is het *SBPhoneOverviewViewController* object. Dit object krijgt notificaties van bijvoorbeeld de stap waar de gebruiker opmerkingen toevoegt met als informatie de toegevoegde tekst. Deze tekst wordt doorgestuurd naar de *SBOperationManager* die het in het juiste object opslaat.

CoreData

In vorige Sprints is gebruik gemaakt van de objecten *SBReservation* en *SBProceeding* om informatie over een reparatieverzoek op te slaan. In de huidige Sprints zullen deze objecten gemigreerd worden naar de CoreData database. De objecten moeten dan klaar worden gemaakt om als subklasse te dienen van een *NSManagedObject*. Deze CoreData objecten kunnen in het model gekoppeld worden aan een *Defect*, dit object is al sinds de tweede Sprint geïmplementeerd in CoreData en hoeft niet te worden aangepast om voor deze doeleinden te werken.



Figuur 17: CoreData database design, tweede versie. (Zie bijlage P).

Een opmerking over bovenstaand ontwerp is dat het misschien een fout ontwerp lijkt door de relaties tussen een *Proceeding* object en het *Defect*, *Element*, *Space* en *Location* object. Aangezien voor het reparatieverzoek alleen het *Defect* object belangrijk is lijkt het niet nodig om de andere objecten ook als referentie in een *Proceeding* op te slaan. Hier is echter wel voor gekozen omdat op deze manier een eerdere keuze wel te herkennen wanneer nog niet een specifiek *Defect* object is uitgekozen. Dit kan gebeuren wanneer een gebruiker op een bepaald moment niet verder komt dan de *Space*. De twee onderliggende objecten zijn nog niet bekend maar tot deze stap kan wel informatie worden ingevuld omdat de vorige stappen wel beschikbaar zijn. Concluderend zou het qua ontwerp mooier zijn om een enkele relatie naar een *Defect* te implementeren, maar in dit geval is het voor de werking van de applicatie en de ervaring van de gebruiker niet de beste oplossing.

Excel importeren & exporteren

Tijdens de demonstratie is aan bod gekomen dat de opdrachtgever graag een eenvoudige methode in het beheersysteem wil zien die het mogelijk maakt om datum / dagdeel combinaties te importeren of exporteren. Op deze manier hoeft een medewerker niet veel moeite te doen om alle beschikbaarheid voor een bepaalde dag in te voeren wat het proces versnelt. Het proces wordt uitgevoerd door eerst een exporteer methode aan te roepen die een standaard template download met een aantal dagen reeds ingevuld. De werknemer kan deze informatie eventueel wijzigen voordat het template weer geïmporteerd wordt waarna de waarden toegevoegd worden aan de database.

Bovenstaande werkwijze is geïmplementeerd door gebruik te maken van een open source Framework genaamd PHPExcel. Door gebruik te maken van dit Framework kan programmatisch eenvoudig een nieuw excel bestand aangemaakt worden. Ook voor het verwerken en uitlezen van deze bestanden is ondersteuning wat voor ons ideaal is. Voor de implementatie en gedetailleerde beschrijving van de code die deze acties afhandelt verwijs ik u naar de bijlagen (Zie bijlage H voor de inhoudsopgave van code).

Uitbreiden van de API

Omdat er vanuit de mobiele applicatie twee extra acties door de gebruiker uitgevoerd kunnen worden is het noodzakelijk dat er nieuwe API methodes worden geïntroduceerd. Dit is respectievelijk een methode die het mogelijk maakt om een reparatieverzoek te annuleren en een methode die de status van reparatieverzoeken kan opvragen. Voor deze uitbreiding van de API is een document opgesteld die alle API methoden beschrijft. In dit document is bijvoorbeeld te vinden welke POST parameters verwacht worden voor een succesvol resultaat (Zie bijlage Q).

Een belangrijke opmerking die op dit moment gemaakt moet worden is de veiligheid en privacy bescherming van reparatieverzoeken. Via de API wordt namelijk geen controle uitgevoerd of de aanvrager van de gegevens wel daadwerkelijk toegang heeft tot de informatie. Dat wil zeggen dat door het aanpassen van een ID die gebruikt wordt om een reparatieverzoek te identificeren het mogelijk is om verzoeken die niet van jou zijn te annuleren. Dit probleem zal in een later stadium worden opgelost wanneer alle API methodes tegelijk beter worden beveiligd tegen onder meer dit genoemde probleem.

Sprint review

Tijdens de Sprint review is vooral de aandacht besteed aan het beheer systeem, dit was ook het een van de hoofddoelen van de Sprint. Tijdens de review is elke mogelijke stap doorlopen die mogelijk is in het beheer systeem. Bij een aantal schermen is de conclusie getrokken dat vooral de naamgeving opnieuw zal moeten worden bekeken. Voor de opdrachtgever zou het namelijk niet duidelijk zijn onder welke opties bepaalde functies zitten. Dit is geen grondige wijziging aangezien alle tekstuele aanpassingen in een taalbestand moeten worden aangepast. Als tip werd gegeven om zoveel mogelijk te kijken naar hoe het beheer werkt in het reeds bestaande systeem en het hier zo veel mogelijk op te laten lijken. Op deze manier zullen de medewerkers van de opdrachtgever weinig moeite hebben met de introductie van een nieuw systeem omdat het nagenoeg hetzelfde werkt. Het is helaas onmogelijk om alle onderdelen op precies dezelfde methode uit te laten voeren. Zo is het beheer scherm voor nieuwe reparatieverzoeken ontwikkeld met een checkbox voor elke rij zodat sneller goedkeuren van verzoeken mogelijk wordt gemaakt. Dit zie je nergens anders in het bestaande systeem. Voor schermafbeeldingen van het huidige systeem zoals dit op het einde van deze Sprint is geïmplementeerd verwijs ik u naar de bijlagen.

Tijdens de review van de mobiele applicatie kwam naar voren dat sommige functies die nu zijn toegevoegd niet nodig zijn. Zoals het toevoegen van meerdere defecten in een enkel verzoek, aangezien naar verwachting niet veel bewoners gebruik zouden gaan maken van deze functie. Deze functie zal echter wel in de mobiele applicatie geïmplementeerd blijven aangezien dit de wens was van de opdrachtgever. Verder is de mobiele applicatie zo goed als af, waardoor meer aandacht besteed kan worden aan het design. In de volgende Sprint zal daarom getracht worden een traject te starten om een design te laten maken door een bedrijf dat hierin gespecialiseerd is.

Conclusie

Na afronding van de vijfde Sprint is een mobiele applicatie opgeleverd die zo goed als gereed is om een design voor te laten maken. Daarnaast kan de applicatie ook onderzocht worden op de onderzoeksvragen die voor het afstudeerproject opgesteld zijn, met name de beveiliging en efficiëntie aspecten van de mobiele applicatie. Het grootste deel van de minimale functionaliteit is op dit moment geïmplementeerd. Bij de Sprint retrospective is al besloten dat bij het plannen van de volgende Sprint een aantal taken worden gemaakt die te maken hebben met de beveiliging van de applicatie en een belangrijke toevoeging aan dit verslag.

6. Internet beveiliging

Voor de zesde Sprint is als doel gesteld de applicatie te voorzien van beveiligde communicatie met de API server. Daarnaast zal een manier van communicatie moeten worden bedacht zodat de API kan bepalen welk mobiel apparaat de connectie probeert te openen. Door dit te bepalen kunnen eventueel apparaten worden afgesloten van communicatie met de API.

Sprint Planning

Om bovenstaande onderdelen in te kunnen plannen zullen de taken bepaald moeten worden voor elk onderdeel. Het onderzoek naar een wijze om aan te kunnen tonen dat een bepaalde telefoon probeert te communiceren is opgesplitst in drie onderdelen. Het eerste onderdeel is het onderzoeken wat de beste methode is om dit aan te pakken, de andere twee onderdelen zijn respectievelijk het implementeren van deze methode op de API en op de mobiele applicatie. Tijdens het plannen is in totaal voor deze onderdelen 13 uur ingepland.
Het volgende onderdeel dat als hoofddoel van de Sprint zal worden uitgevoerd is het invoeren van een beveiligde connectie. Dit zal door middel van Secure Sockets Layer (nader genoemd SSL) geïmplementeerd worden. In totaal is voor dit onderdeel 8 uur ingepland, dit aantal is gekozen omdat de API server al beschikt over een SSL certificaat en communicatie en dit dus niet door de ontwikkelaar geregeld hoeft te worden. Aan de API kant zal dus niet veel gewijzigd hoeven worden. Voor de mobiele applicatie zal echter wel een bepaalde verificatie methode moeten worden geïmplementeerd die kan controleren of het certificaat dat de beveiligde server gebruikt geldig is.

Als laatste onderdeel zullen de defect keuze schermen worden uitgebreid met een afbeelding van het defect. Deze afbeeldingen worden getoond in de mobiele applicatie om de gebruiker duidelijker aan te tonen wat het defect inhoud. Voor dit onderdeel is 13 uur ingepland aangezien er waarschijnlijk veel tijd gestoken moet worden in een caching methode die ervoor zorgt dat de applicatie niet onnodig afbeeldingen gaat ophalen.

Geïdentificeerde connectie methoden

Een geïdentificeerde connectie met de API server is benodigd om zo ontoegankelijke connecties tegen te houden. Zo zou het in dit geval alleen mogelijk moeten zijn om met de API te communiceren wanneer je daadwerkelijk vanuit de legale versie van de mobiele applicatie de connectie opent. Op deze manier wordt alleen informatie opgehaald door de daartoe toegewezen applicatie.

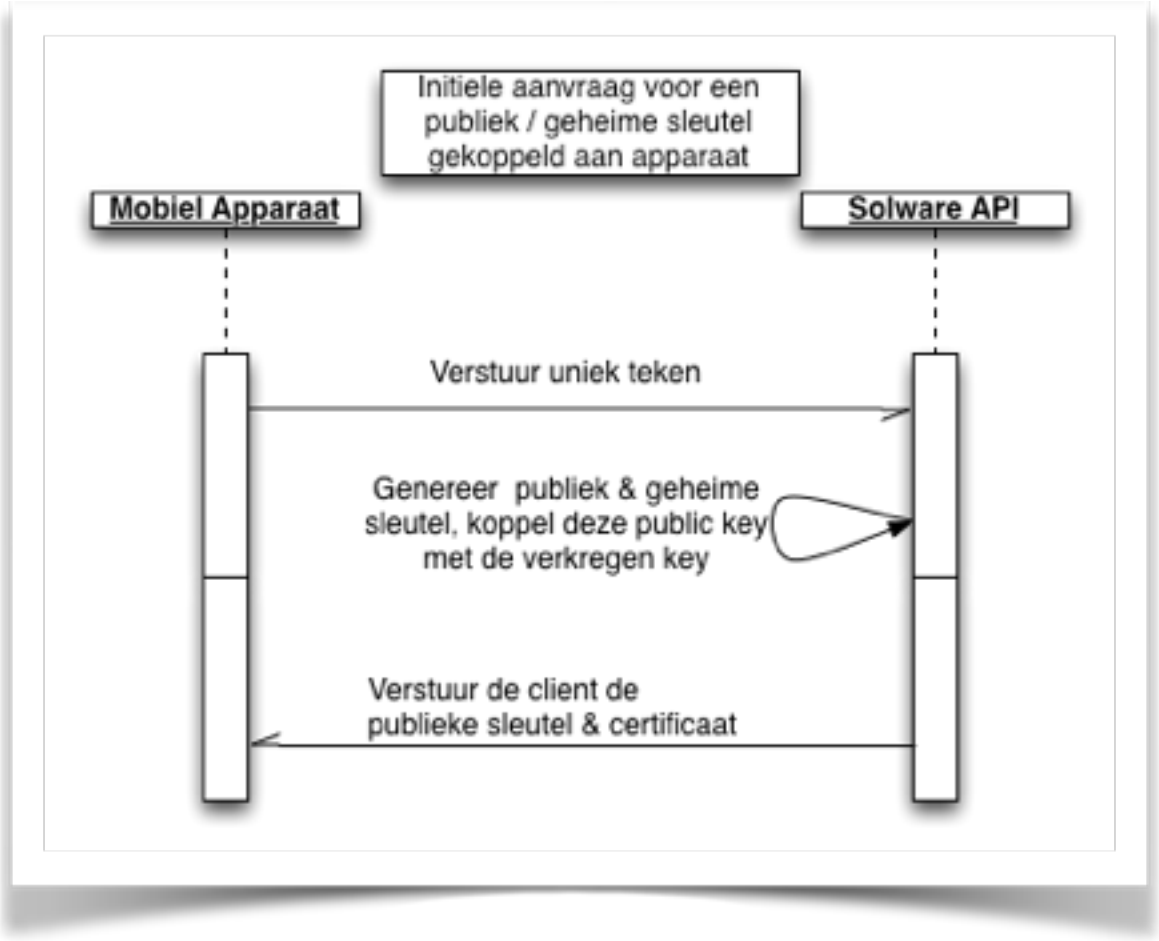
Om de juiste methode te kunnen kiezen die geïmplementeerd zal worden om deze geïdentificeerde connectie te realiseren zal eerst moeten worden onderzocht wat de mogelijkheden hiervoor zijn. Voor het onderzoek zullen 4 verschillende methodes met elkaar vergeleken worden, uit deze methoden zal gekozen worden wat de beste manier is om een geïdentificeerde connectie te gebruiken. De volgende methodes zullen vergeleken worden:

- Client- server certificaten door middel van een Certificate Signing Request (CSR).
- Communicatie door middel van een uniek device token als username.
- Communicatie door middel van het aanvragen van een API token, zonder identieke device tokens.
- Communicatie door middel van SMS verificatie.

Client-server certificaten & Certificate Signing Request

Deze methode maakt gebruik van een publiek en geheim sleutel paar dat aangemaakt wordt door een Certificate Authority. Wat in dit geval de Solware API server zal zijn. De Certificate Authority maakt een publieke sleutel aan aan de hand van het door het apparaat meegegeven identificatie teken. Deze zal verder gebruikt worden om het apparaat te identificeren wanneer het deze informatie aanbiedt die versleuteld is door middel van deze publieke sleutel. Echter kan de server alleen door middel van de gegenereerde geheime sleutel deze versleutelde informatie ontcijferen en lezen. Op deze manier heb je een beveiligde connectie met de server, alle informatie die verstuurd wordt is versleuteld en onleesbaar voor iedereen die niet over de private sleutel beschikt. Daarnaast is het ook mogelijk om aan de hand van het gebruikte publieke sleutel aan te tonen welk apparaat de connectie probeert te maken. Voor een verduidelijkt overzicht hoe de communicatie zou gaan werken wanneer deze methode geïmplementeerd zou worden zie figuur 18 (Zie bijlage R).

Door gebruik van een Certificate Authority en CSR maak je het dus mogelijk om weliswaar door middel van een zelf-ondertekend certificaat versleutelde communicatie te ondersteunen met daarnaast de mogelijkheid om apparaten te identificeren.



Figuur 18: Aanvraag van een nieuwe publieke sleutel voor communicatie met de API. (Zie bijlage R).

Aan de hand van de verkregen publieke sleutel kan eventueel informatie extra versleuteld worden verstuurd, aangezien er ook al gebruik gemaakt zal worden van de standaard SSL functionaliteit. Per nieuw API verzoek wordt het unieke nummer van het mobiele apparaat verstuurd. Het unieke

nummer dat wordt meegestuurd wordt door middel van een cryptografische hashfunctie opgeslagen in de database. Op deze manier zou wanneer kwaadwillende toegang krijgen tot de database de unieke nummers van mobiele apparaten niet uitlezen. Aan de hand van dit nummer kan worden opgezocht wat de benodigde geheime sleutel is om eventueel versleutelde informatie uit te kunnen lezen. Als tweede weet de API nu ook precies welk apparaat de connectie heeft geopend en kan hier mogelijk op reageren door de connectie niet te accepteren wanneer het apparaat te veel spam heeft verstuurd.

Communicatie door middel van uniek teken

Eerder werd al omschreven dat gebruik gemaakt zal worden van een zogenaamd uniek apparaat teken. Dit teken zou identiek moeten zijn en niet aangepast kunnen worden door de gebruiker. Wanneer de applicatie opnieuw geïnstalleerd zou worden mag dit unieke teken ook niet aangepast worden om er zo voor te kunnen zorgen dat een en hetzelfde apparaat altijd herkend wordt door de server.
Deze methode maakt ook gebruik van dit teken maar dan zonder een Certificate Signing Request. Dat wil zeggen dat per verzoek naar de API het unieke nummer verstuurd wordt zodat er geen mogelijkheid wordt gecreëerd om eventueel informatie extra te versleutelen. Deze methode is daarom makkelijker te implementeren aangezien er geen handelingen hoeven te worden uitgevoerd met de OpenSSL code om certificaten, publieke en geheime sleutels te genereren. Op de server wordt wederom het unieke nummer door middel van een cryptografische hashfunctie opgeslagen om zo misbruik te voorkomen. Aan de hand van dit unieke nummer zou een apparaat herkend kunnen worden.

Om een uniek teken te genereren voor een apparaat kan een methode worden aangeroepen van de *UIDevice* klasse, namelijk de *uniqueIdentifier* methode. Deze methode stuurt een string terug met daarin het unieke nummer aan de hand van diverse hardware details. Dit nummer is door de gebruiker niet aan te passen en blijft hetzelfde tijdens de gehele levensduur van het mobiele apparaat. Helaas is deze methode sinds de invoer van iOS 5 afgeschaft, dat wil zeggen er andere methodes verzonnen moeten worden om een uniek nummer te genereren. Alleen is het de vraag of de applicatie zal worden goedgekeurd door Apple wanneer je hardware details gaat uitlezen door middel van geheime API methodes. Het uitlezen van hardware zal door die reden niet direct gebruikt worden tenzij dit echt noodzakelijk is. Apple stelt andere methodes beschikbaar om te gebruiken, echter blijven deze waardes niet uniek nadat de applicatie opnieuw geïnstalleerd wordt. Op deze manier is het voor kwaadwillende mogelijk om een nieuw uniek nummer aan te vragen en opnieuw te communiceren met de API server totdat deze wordt geblokkeerd, waarna de applicatie verwijderd kan worden en de gehele cirkel zich herhaald. Echter zal deze wijze van misbruik weinig voorkomen, meer dan 99% van de gebruikers zal de applicatie normaal gebruiken. Dus zal een zelf gegenereerd teken die identiek is per installatie volstaan. Ook voor de Certificate Authority methode zal een zelf gegenereerd uniek teken voldoen.

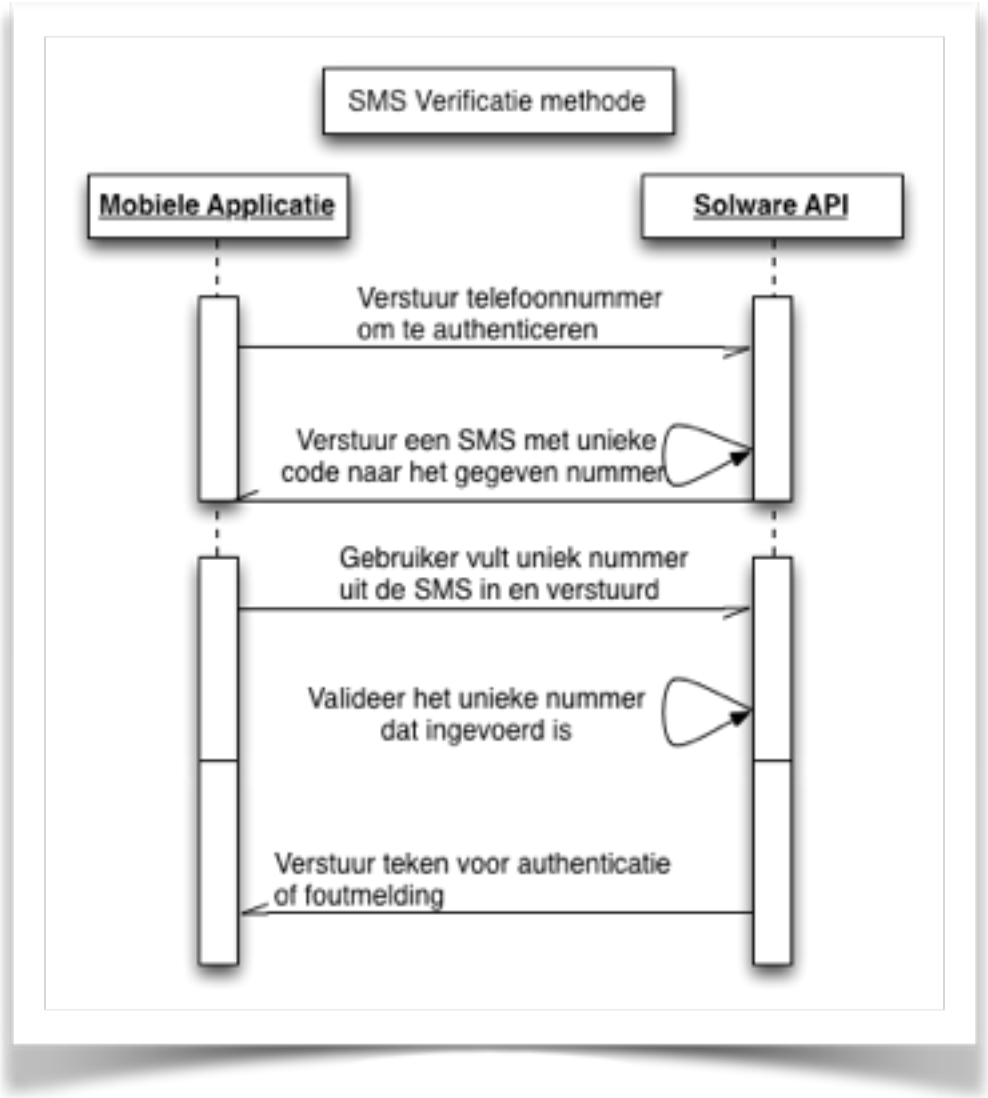
Communicatie door middel van een gegenereerd API token

Deze methode lijkt in grote lijnen op de methode die authenticeerd aan de hand van een uniek apparaat teken. Alleen zal in eerste instantie een token gegenereerd moeten worden door de API server die gebruikt moet worden om informatie van de API binnen te halen. Op deze manier hoeft het unieke nummer van de telefoon niet in de database te worden opgeslagen die gebruikt wordt door de API. Wat een voordeel is voor de privacy van de eindgebruiker.

SMS verificatie

Door gebruik van SMS verificatie wordt gevalideerd dat de eind gebruiker toegang heeft tot de telefoon waar het opgegeven telefoonnummer toe behoort. SMS verificatie kan in de applicatie worden ingebouwd door de gebruiker zijn telefoonnummer in te laten voeren, dit is nodig zodat je als programmeur geen toegang hebt tot het nummer van het apparaat waar de applicatie op draait.

De gebruiker zal dit nummer dus zelf in moeten voeren. Naar dit nummer wordt vanaf de server een SMS bericht verzonden met een unieke code die gebruikt kan worden om communicatie met de API te activeren. Echter geeft dit niet de garantie dat de communicatie ook daadwerkelijk van de geactiveerde telefoon komt. Op deze manier limiteer je wel het aantal geactiveerde accounts dat de gebruiker kan gebruiken om connecties te activeren met de API. Aangezien gebruikers vaak niet over een groot aantal telefoons beschikken is dit geen probleem. In figuur 19 (Zie bijlage S) is een afbeelding te zien die de wijze van authenticatie door middel van een SMS beschreven.



Figuur 19: Verificatie van een mobiel apparaat door middel van SMS. (Zie bijlage S).

Wanneer het unieke nummer dat verstuurd wordt om aan te tonen dat de gebruiker het SMS bericht heeft gehad gelijk is aan het nummer dat verstuurd is zal de gebruiker toegang krijgen tot communicatie met de API.

Vergelijking

Alle beschreven methodes garanderen niet dat er geen mogelijkheid is om een connectie te maken voor personen die hier niet bevoegd voor zijn. Het is namelijk voor andere apparaten ook mogelijk om een CSR te versturen om zo een publieke sleutel te krijgen. Deze connecties kunnen later handmatig worden afgesloten door een medewerker. Wat je met deze methode wel garandeert is extra beveiligde overdracht en de identificatie van een systeem. Bij het gebruik van SMS verificatie garandeer je dat de eind gebruiker toegang heeft tot een telefoon met een geldig

telefoonnummer, na de verificatie kan echter ook gebruik worden gemaakt van de API service vanuit elk systeem.
Door gebruik van tokens verwijder je de moeilijkheden die mee worden gebracht met de zojuist beschreven methodes. Het is nog steeds mogelijk om een systeem te identificeren, alleen is het mogelijk om de applicatie opnieuw te installeren waardoor je de authenticiteit ververst. Dit geldt ook voor de methode waarbij een uniek nummer wordt gebruikt.

De beste manier om een systeem te identificeren zou zijn wanneer Apple een download afschrift gebruikt om zo aan te kunnen tonen dat een applicatie op de telefoon geïnstalleerd is. Zo kun je via de API server het afschrift controleren bij Apple waarna je weet of het systeem de applicatie via de legale App Store heeft gedownload. Deze methode is helaas nog niet beschikbaar en het is onbekend of dat deze ooit beschikbaar zal worden gesteld.

Conclusie

De uiteindelijke keuze is gevallen op de methode die tokens genereert aan de API kant. Deze tokens moeten door client systemen gebruikt worden om verder informatie van de API te kunnen verschaffen. Deze methode is eenvoudig te implementeren en biedt evenveel mogelijkheden als bij de andere methodes. Het is namelijk niet geheel mogelijk om een systeem altijd hetzelfde unieke nummer te laten genereren. SMS verificatie is voor de gebruiker een last, aangezien het de gebruiker relatief veel tijd en moeite kost om de telefoon te authenticeren terwijl we het de gebruiker zo makkelijk mogelijk willen maken om een verzoek te sturen.

Het authenticeren door middel van een token zal geïmplementeerd worden in combinatie met de HTTP basic authentication methode. Hierbij gebruikt de applicatie een door de server gegenereerde gebruikersnaam en wachtwoord. Hierbij is het wachtwoord de verkregen token.

Secure Socket Layers (SSL)

Om ervoor te zorgen dat de mobiele applicatie altijd beveiligd communiceert met de buitenwereld en daarnaast ook alleen communiceert met de juiste servers is het noodzakelijk dat Secure Socket Layers geïmplementeerd wordt. Door dit te implementeren wordt alle informatie die verstuurd wordt versleuteld verstuurd door middel van een publieke sleutel, deze informatie is alleen door de eigenaar van de geheime sleutel te ontcijferen. De eigenaar van deze sleutel is in dit geval de Solware API.

Een kwetsbaarheid die voorkomen kan worden is de zogenaamde “Man-in-the-Middle” aanval. Deze aanval introduceert een tussenpersoon die als communicatie medium dient tussen de mobiele applicatie en de Solware API. Omdat de tussenpersoon van de server de publieke sleutel heeft weten te krijgen kan andere informatie worden aangemaakt en doorgestuurd worden naar de API, waardoor de juiste informatie van de mobiele applicatie wordt weggegooid. Andersom gebeurd hetzelfde, de informatie die door de server wordt verstuurd kan ook aangepast worden door de tussenpersoon waardoor de mobiele applicatie andere informatie ontvangt dan de bedoeling is. Door gebruik te maken van SSL wordt dit probleem niet direct opgelost. Om dit wel op te lossen dient op de mobiele applicatie het verkregen certificaat van de server (of tussenpersoon) gevalideerd te worden. Dit gebeurt door te controleren of het certificaat niet zelf-gesigneerd is. Als dit het geval is is de kans groot dat een kwaadwillend persoon de connectie probeert over te nemen. Wanneer dit het geval is zal de mobiele applicatie moeten beslissen of de connectie gemaakt wordt.

Andere manieren om te controleren of het certificaat geldig is, is door te controleren of het certificaat is uitgegeven door een vertrouwd Root Certificate Authority. Hierdoor weet je zeker dat het bedrijf achter het certificaat vertrouwd is. In het geval van de mobiele applicatie kan een lijst van vertrouwde Certificate Authorities geïnstalleerd worden die valideren of de uitgever van het certificaat geldig is.

Een beveiligingsrisico dat voor kan komen is het feit dat wanneer de geheime sleutel (certificaat) gestolen is van de API dat een tussenpersoon zich precies kan voordoen als de juiste server. Deze kans is echter klein en is door middel van deze opdracht niet veel tegen in te brengen. Daarom zal de door middel van certificaat verificatie methode geïmplementeerd worden aangezien dit voor voldoende beveiliging zal zorgen.

Beveiliging implementatie

Door de implementatie van zowel een token authenticatie systeem als gebruik van SSL met certificaat verificatie kunnen we minimale beveiliging garanderen. De server weet welk mobiel apparaat de connectie probeert aan te maken en kan hierdoor beslissen welke informatie wel of niet getoond mag worden. Daarnaast weet de mobiele applicatie wanneer een connectie wordt aangemaakt met onbetrouwbare bronnen door middel van het valideren van het certificaat aan de hand van Root Certificate Authorities. Aan de API kant hoeven geen aanpassingen doorgevoerd te worden aangezien SSL al standaard ondersteund wordt door Solware. Voor de implementatie van de beveiliging methode en een gedetailleerde uitleg van de code verwijs ik naar de bijlagen (Zie bijlage H voor de inhoudsopgave van de code).

Wanneer de authenticatie niet is geannuleerd aan de hand van het certificaat zal de server vragen om basic HTTP authentication gegevens. Deze gegevens zijn nodig om een mobiel apparaat te identificeren zodat we kunnen bepalen of dit apparaat toegang heeft tot informatie die opgevraagd wordt. Als deze authenticatie niet zou plaatsvinden is het mogelijk om door middel van het aanpassen van een identifier in het HTTP verzoek informatie van andere gebruikers op te vragen. Door in de achterliggende database alle informatie te koppelen aan een gebruiker is het mogelijk om te controleren of een apparaat dat informatie wil downloaden ook daadwerkelijk toegang heeft. De gebruiker kan aan deze login gegevens komen door een bepaalde API functie aan te roepen met een unieke tekenreeks. De API maakt voor deze tekenreeks nieuwe login gegevens aan die door het mobiele apparaat gebruikt kan worden bij alle andere API verzoeken.

Conclusie

Door de geïmplementeerde bevindingen van beveiliging is de mobiele applicatie nu op een minimale methode beveiligd. Een man in the middle aanval wordt nu zo goed als onmogelijk gemaakt omdat de certificaten gecontroleerd worden. Daarnaast wordt alle belangrijke informatie versleuteld verzonden door middel van SSL om er zo voor te zorgen dat kwaadwillenden de informatie niet kunnen lezen. Dit is vooral belangrijk voor de adres en contactgegevens van de bewoner omdat dit gevoelige informatie is.

Tijdens de review is niet veel van dit onderdeel behandeld aangezien het niet een visueel onderdeel is van de applicatie. Naast de beveiliging is het onderdeel behandeld die afbeeldingen bij defecten download en toont aan de bewoner. Feedback op dit onderdeel is de tijd dat een afbeelding in de cache geldig blijft, in eerste instantie is dit op een week ingesteld maar omdat naar alle waarschijnlijkheid de afbeeldingen toch niet veranderen zal deze tijd langer worden ingesteld.

De 6e Sprint is daarmee succesvol afgerond. Alle onderdelen die gedaan moesten worden zijn uitgevoerd waardoor zonder extra onderdelen gestart kan worden aan de volgende Sprint.

7. Thread management

De zevende Sprint zal zich vooral bezighouden met het verbeteren van kleine onderdelen in de mobiele applicatie en beheer systeem om zo het gehele project klaar te kunnen maken voor een design. De belangrijkste activiteiten die in de Sprint uitgevoerd zullen worden zijn de volgende:

- Compleet herschrijven van de background threading klassen, vooral lettend op foutafhandeling en internet beschikbaarheid afwisseling.
- Onderzoeken wat de beste instellingen zijn om ervoor te zorgen dat background threads die communiceren over het internet sneller worden afgehandeld. Dit behandelt gelijk de doelstelling hoe de het beste MultiThreaded systemen geïmplementeerd kunnen worden op het iOS platform.

Verder zullen in de product backlog een aantal kleinere onderdelen worden gedefinieerd die te maken hebben met verfijnen en perfectioneren van functionaliteit. Aan het eind van de Sprint zal dus het doel bereikt worden wanneer de applicatie klaar is om een design op te zetten waarna het als beta in productie kan worden gezet.

Sprint planning

Deze Sprint zal vooral rekening worden gehouden met kleine aanpassingen die gedaan moeten worden aan bepaalde functionaliteiten. Daarnaast zal de applicatie voorbereid worden op de vormgeving die de volgende Sprint zal worden geïmplementeerd. Als laatste zal een Proof of Concept worden opgeleverd die beschrijft wat de beste en meest efficiënte methode is om de afhandeling van internet communicatie uit te voeren.

De kleine aanpassing die uitgevoerd zal worden is bijvoorbeeld een afbeelding van het gekozen defect tonen in het verzoeken overzicht. Daarnaast zal een extra navigatie balk worden toegevoegd bij het invoeren van tekst om zo sneller te kunnen navigeren tussen de verschillende invoervelden. Deze genoemde functionaliteit zal deel zijn van de visuele aanpassingen van de mobiele applicatie.
Een niet visuele aanpassing die uitgevoerd zal worden is het reageren op de zogenaamde “Memory warnings”, wat dit precies inhoud zal later in dit hoofdstuk worden beschreven. Ook zal binnen de REST API beveiliging moeten worden toegepast om ervoor te zorgen dat onbevoegde gebruikers geen toegang hebben tot bepaalde informatie.

Bovengenoemde functionaliteit zal eerst worden geïmplementeerd voordat het onderzoek naar performance wordt uitgevoerd. Hier is voor gekozen omdat het belangrijk is eerst de applicatie zo werkend mogelijk af te maken voordat een Proof of Concept wordt gemaakt. Op deze manier kan bij de review de nieuwe versie worden getoond zonder dat er veel onderdelen nog niet afgemaakt zijn door tijd gebrek of andere problemen.

Hoe de verschillende onderdelen zijn ingepland is te zien in onderstaand figuur 20 (Zie bijlage E), in deze afbeelding is te zien dat eerst tijd zal worden gestoken in de onderdelen die van belang zijn voor Solware. Nadat deze onderdelen zijn uitgevoerd zal het onderzoek worden gestart naar de performance van de applicatie.

Figuur 20: Sprintlog voor de 7de Sprint (Zie bijlage E).

Als laatste beschrijving voor de Sprintlog zijn de verschillende onderdelen van het onderzoek. In bovenstaande afbeelding zie je een onderdeel met nummer BC-633. Onder dit onderdeel bevinden zich de onderstaande sub-onderdelen. Deze onderdelen zullen individueel worden behandeld in dit hoofdstuk bij het onderzoek gedeelte.

- Object aanmaken dat de internet beschikbaarheid bijhoudt en notificaties doorstuurt. (3 uur)
- Zorgen dat operaties reageren op wanneer internet beschikbaarheid wordt aangepast. (5 uur)
- Alle operatie objecten geschikt maken voor het nieuwe ontwerp. (5 uur)
- Ontwerpen van een model voor het verwerken van background threads. (3 uur)
- Testen van de operatie klassen, hierbij vooral richten op foutafhandeling binnen de applicatie. (5 uur)

Memory management

Memory management in een iPhone applicatie is een belangrijk onderdeel om rekening mee te houden. Op dit platform zijn een aantal methodes die je kunnen helpen dit zo efficiënt mogelijk aan te pakken. Zo heb je bijvoorbeeld de mogelijkheid om het project te starten met het zogenaamde Automatic Reference Counting[20] (nader genoemd ARC), dit zorgt ervoor dat je niet zelf objecten uit het geheugen hoeft te halen. Dit zal de garbage collector voor je doen. Als hier geen gebruik van wordt maakt zal per object dat je niet meer nodig hebt de methode *release* of *autorelease* moeten worden aangeroepen.

Tijdens de start van het project is ervoor gekozen om standaard gebruik te maken van ARC, op deze manier kan veel tijd worden bespaard met het goed beheren van de aangemaakte objecten. Ook zal tijd worden bespaard met problemen die veroorzaakt worden door fouten met verwijderen van objecten uit het geheugen. Deze problemen zijn voorspeld aan de hand van ervaring met eerdere projecten op het iPhone platform.

[20] Bron: http://clang.llvm.org/docs/AutomaticReferenceCounting.html

Een tweede methode om ervoor te zorgen dat de mobiele applicatie zo min mogelijk geheugen gebruikt is door te reageren op zogenaamde “Memory Warnings”. Wanneer een dergelijke waarschuwing plaatsvindt worden notificaties aan applicaties verstuurd om te proberen meer geheugen vrij te maken. Op het moment van de notificatie is namelijk niet genoeg geheugen vrij in het systeem van de mobiele telefoon. Reageren op deze notificatie is vrij eenvoudig en kan door middel van een regel code worden uitgevoerd op een bepaald object. Je registreert namelijk een methode van dit object die aangeroepen wordt wanneer een notificatie is verstuurd. In ons project zijn op dit moment alle *UIViewController* objecten klaargemaakt om te reageren op deze notificaties.

Wanneer een *UIViewController* object deze notificatie krijgt wordt gecontroleerd of het *UIView* object van de controller niet wordt getoond op het scherm. Wanneer dit wel zo is betekent dit dat er geen geheugen vrijgemaakt hoeft te worden omdat de gebruiker op dat moment naar dat scherm kijkt, je wilt namelijk voorkomen dat de gebruiker een leeg scherm krijgt vanuit het niets door view objecten te verwijderen uit het geheugen. In andere gevallen wanneer het *UIView* object niet wordt getoond is het mogelijk om objecten uit het geheugen te halen zoals de getoonde tabel of afbeelding. In het geval van een afbeelding kan dit zorgen voor een geheugen winst ten opzichte van een moment eerder. In de andere gevallen is het ook vaak zo dat de applicatie een kleine hoeveelheid geheugen vrij kan maken aangezien er vaak meer dan 5 *UIViewController* objecten in het geheugen staan, maar op dat moment niet op het scherm worden getoond.

Voorbeelden van de code die dit afhandelt zijn te vinden in de bijlagen (Zie bijgale H voor de inhoudsopgave van code). In de meeste *UIViewController* objecten, in de methode *didReceiveMemoryWarning* wordt deze actie uitgevoerd. Kort beschreven wordt eerst gecontroleerd of de view deel uitmaakt van het *UIWindow* object, als het view object hier geen deel van uitgemaakt worden verschillende objecten uit het geheugen verwijdert.

Scheduling onderzoek

In dit onderdeel zal een belangrijke deelvraag van de afstudeeropdracht worden afgehandeld. Dit is namelijk de vraag op welke manier het beste een thread scheduling methode kan worden gebruikt. Uiteindelijk zal de keuze gemaakt moeten worden tussen de standaard beschikbare *NSOperationQueue* en een eigengemaakte implementatie hiervan. Beide methodes hebben zo hun eigen voor- en nadelen die in dit onderdeel duidelijk gemaakt zullen worden.

In dit onderdeel zal eerst de huidige standaard worden beschreven, waarna het ontwerp van de nieuwe implementatie wordt voorgesteld en geïmplementeerd. Na de implementatie zal de methode getest en onderzocht worden voor gebruik.

Tegelijkertijd met dit onderzoek zullen alle operatie objecten herschreven worden voor ondersteuning van bijvoorbeeld annulering en het tijdelijk stoppen van een bepaalde operatie. Dit is namelijk nog niet geïmplementeerd in de huidige versie. Dit kan je zien als een herontwerp waar goed over nagedacht is in plaats van de snelle implementatie keuze aan het begin van het project. De referentie naar de taak van de backlog die hiervoor is aangemaakt is BC-633 tot en met BC-637.

Standaard & problemen

De standaard die gebruikt wordt in de meeste mobiele applicaties is de zogenaamde *NSOperationQueue*. Dit object is standaard inbegrepen in de SDK en wordt gebruikt in combinatie met *NSOperation* objecten. Over deze objecten is al eerder geschreven in de eerste Sprint, dit wordt namelijk al sinds het begin van het project gebruikt in de applicatie en heeft altijd naar behoren gewerkt. Het object dat ervoor zorgt dat operaties worden gestart kijkt ten alle tijden naar de systeem specificaties en bezigheid. Aan de hand hiervan wordt een operatie die wacht gestart.

Een nadeel aan dit object is dat het niet automatisch operaties annuleert of pauzeert wanneer dit aan de hand van systeem prestaties op dat moment wel nodig is.
Een ander nadeel is ook dat dit object niet het aantal actieve connecties limiteert, het kan dus mogelijk (in het geval) zijn dat er tegelijk 100 connecties openstaan. De prestaties zouden verbeterd kunnen worden door bijvoorbeeld 10 connecties tegelijkertijd open te houden. Op deze manier worden operaties sneller afgerond in de plaats van lang te wachten op een enkele operatie.

Concluderend kunnen we zeggen dat we met de nieuwe scheduling methode de volgende problemen kunnen oplossen. Deze problemen hebben vooral te maken met de connectie prestaties.

- Een maximaal aantal openstaande connecties.
- Het afhandelen van wegvallend internet en bijvoorbeeld de overgang van een snelle naar een langzame verbinding (3G naar GPRS of Edge).
- Het versnellen van afhandeling van openstaande connecties.

Nieuw ontwerp

Voor het nieuwe ontwerp (Zie bijlage T) van de scheduling methode zullen een aantal objecten worden geïntroduceerd om bijvoorbeeld de processor activiteit en internet veranderingen bij te houden. Deze objecten worden gebruikt in het scheduling package. Daarnaast zullen twee andere packages worden geïntroduceerd, namelijk de Operation en State package. Door gebruik te maken van packages is het mogelijk om de code tot een framework te bouwen en dynamische libraries van aan te maken zodat niet alle code opnieuw hoeft te worden gecompileerd.

Scheduling
Het belangrijkste object in deze package is het *SBOperationManager* object, dit object zorgt ervoor dat operaties die worden toegevoegd uitgevoerd worden en gecontroleerd worden of bepaalde objecten ook daadwerkelijk kunnen starten. Zo moet er bijvoorbeeld een internet connectie aanwezig zijn om een connectie operation uit te voeren. Of operaties mogen niet worden gestart wanneer de processor activiteit hoger is dan maximaal toegestaan. Dit kan worden geregeld door gebruik te maken van de *SBInternetCenter* & *SBProcessorManager* object, deze objecten versturen notificaties naar de manager om er zo voor te zorgen dat deze manager juist handelt wanneer het internet wegvalt. De actie hiervoor is namelijk alle operaties die op dat moment internet gebruiken te laten wachten tot een connectie weer beschikbaar komt. In de tussentijd is het mogelijk om niet-internet afhankelijke operatie objecten uit te voeren.

Operation
In deze package worden alle delegatie interfaces en objecten die benodigd zijn voor operaties gedefinieerd. Het abstracte *SBOperation* object bevat een aantal methodes waar de manager aan kan zien wat de huidige status van dit object is. Zo kan worden bepaald of dit object aan het uitvoeren of klaar is, maar ook of het object internet zal gaan gebruiken of harde schijf acties. Subklassen van dit object zijn de*SBURLOperation* & *SBFileOperation* klassen. Deze objecten zijn standaard verwerkers van internet en bestands acties. Zo kan het *SBURLOperation* object een *SBAPIRequest* afhandelen en het resultaat terugsturen naar een meegegeven object. Er is rekening gehouden met de mogelijkheid om subklassen van deze objecten aan te maken.

State
Deze package zal alleen gebruikt worden in het Proof Of Concept en niet in de daadwerkelijke applicatie. Hier is voor gekozen omdat deze extra functionaliteit niet nodig is voor de werkende applicatie. Als Proof of Concept en uitbreiding is besloten dit wel in te bouwen aangezien dit later als library gebruikt kan gaan worden.
In deze package worden alle verschillende statussen van een operatie beschreven, zo kan een operatie zich bijvoorbeeld in de “wachten” of “geannuleerd” status bevinden. Aan de hand van deze objecten is het mogelijk om extra acties te ondernemen per transactie tussen statussen. Dit

kan vooral van pas komen wanneer een operatie zich moet voorbereiden op “wachten” wanneer het internet is weggevallen.

Implementatie

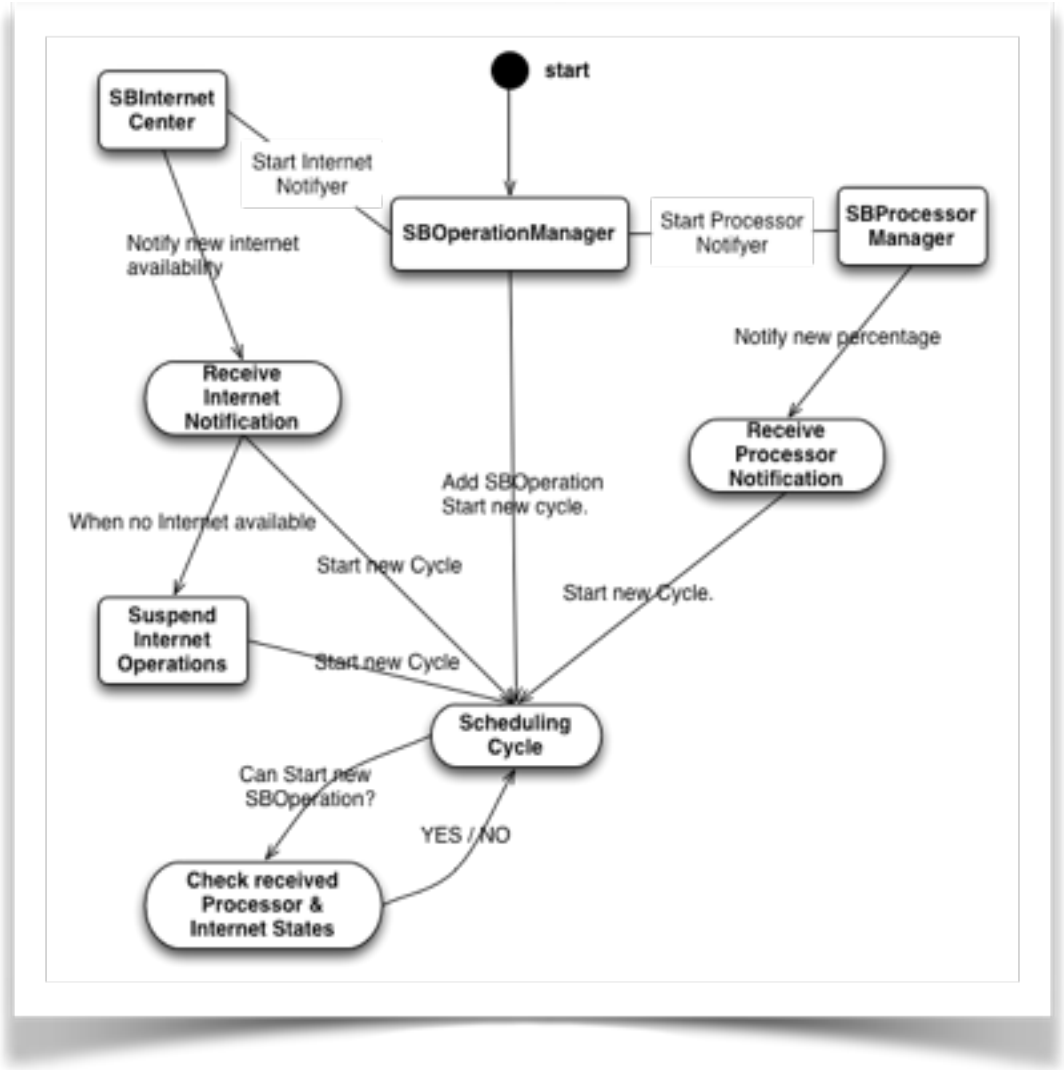
Bij de implementatie van de nieuwe scheduling methode kwamen de vorige versies van de operaties van pas, aangezien deze met een aantal kleine aanpassingen herschreven kon worden naar een nieuwe klasse. Tegelijk met het omschrijven konden een aantal optimalisaties worden uitgevoerd, bijvoorbeeld de synchronisatie klasse die meer acties dan nodig uitvoerde. Deze acties waren in een eerdere versie van de applicatie benodigd maar kunnen op dit moment weggelaten worden.
Een belangrijke methode die voor de werking van de statussen zorgt is de *changeToState: (SBOperationState *) : void* methode. Deze methode wordt namelijk aangeroepen door het vorige status object om ervoor te zorgen dat het object juist geconfigureerd wordt voor de nieuwe status. Voor of na deze status verandering zorgt het *SBOperationState* object ervoor dat de juiste methode op de *SBOperation* wordt aangeroepen om zo door te kunnen gaan.

De *SBProcessorManager* klasse kan per uitzonderlijke core de activiteit uitlezen, van deze informatie kan een totaal gemiddelde worden berekend om zo aan te kunnen tonen of het systeem veel taken heeft. Ook op een systeem met HyperThreading kan het juiste aantal cores worden uitgelezen. Deze actie wordt alleen uitgevoerd wanneer scheduling benodigd is, dat wil zeggen dat *SBOperation* objecten beschikbaar is in de wachtrij of bezig zijn met uitvoeren. Een timer zorgt ervoor dat elke halve seconde een controle wordt uitgevoerd. Voor meer informatie en uitleg van de code die deze informatie berekent verwijs ik naar de bijlage H voor de inhoudsopgave.

Het *SBInternetCenter* object kan lezen of het systeem een connectie heeft via de WiFi interface, daarnaast kan op de mobiele applicatie worden gekeken of een 3G, LTE of EDGE verbinding aanwezig is. Dit laatste kan helaas alleen worden opgevraagd in iOS 7 of hoger, dat betekent dat op eerdere versies van het iPhone platform deze code niet naar behoren zal werken. Om ervoor te zorgen dat het altijd zeker is dat er een WiFi connectie aanwezig is wordt om de halve seconde een controle uitgevoerd of de interface nog aanwezig is. Waar nog geen rekening mee wordt gehouden is of er ook daadwerkelijk gecommuniceerd kan worden met de API wanneer er een interface beschikbaar is. Het kan namelijk zo zijn dat via de WiFi interface het internet niet bereikbaar is. Er is voor gekozen om geen rekening te houden met deze status voor de Proof Of Concept versie. Als de operatie om deze reden fout wordt uitgevoerd zal een foutmelding worden gegenereerd en het object wordt niet opnieuw in de queue gezet.

Communicatie tussen manager, processor en internet objecten gebeurt vooral via delegatie methodes. Hierdoor kan de manager snel reageren op veranderingen in beschikbaarheid en activiteit omdat bijna gelijk na de verandering de manager op de hoogte wordt gebracht. Ook de communicatie tussen een *SBOperation* object en het object dat status veranderingen wil weten zal gebeuren via delegatie methoden.
De communicatie tussen de manager en een operatie zal plaatsvinden aan de hand van Key-Value Observing. Zo kan bijvoorbeeld een operatie laten weten aan de manager wanneer het object klaar is om uitgevoerd te worden door de “isReady” waarde aan te passen. De manager krijgt op deze manier informatie over het object over de volgende statussen / acties (Engelse benamingen) en kan hier zonodig op reageren.

- *isExecuting : BOOL*
- *isCancelled : BOOL*
- *isFinished : BOOL*
- *isReady : BOOL*
- *isUsingInternet : BOOL*
- *isUsingDisk : BOOL*



Figuur 21: Scheduling methode state (Zie bijlage U).

Figuur 21 (Zie bijlage U) toont hoe de *SBOperationManager* operaties start en eventueel stopt. De scheduling methode zorgt ervoor dat de operaties uitgevoerd worden zodra de queue of de uitvoerende array een operatie object bevat. Wanneer de *SBOperationManager* dan van een idle stand naar uitvoerende stand wordt gebracht betekent dit dat operaties uitgevoerd worden, tegelijkertijd activeert de manager het processor en internet management object zodat notificaties worden verstuurd naar de *SBOperationManager*. Een belangrijke methode in de manager is de *startSchedulingCycle* methode, deze methode kijkt naar beschikbare operatie objecten en bepaalt of deze objecten uitgevoerd kunnen worden. Daarnaast controleert het object ook of er “suspended” objecten in de queue aanwezig zijn en herstart deze wanneer dit mogelijk is. De methode wordt aangeroepen wanneer een nieuw operatie object is uitgevoerd, toegevoegd of wanneer de processor of internet manager een notificatie stuurt. Voor een gedetailleerde uitleg van deze methode verwijs ik naar de bijlagen.

Een lastig te implementeren onderdeel van de nieuwe operaties is het feit dat een connectie geannuleerd moet worden wanneer internet wegvalt. Het is namelijk zo dat de connectie eerder doorheeft dat het internet is weggevallen dan wanneer de internet klasse de kans heeft gekregen om een notificatie door te sturen. Hierdoor wordt de operatie niet “suspended” maar zal door middel van een error volledig klaar zijn en niet meer in de queue terugkomen. Terwijl dit wel het geval zal moeten zijn wil het Proof Of Concept naar behoren werken. Op dit moment is dat probleem opgelost door ervoor te zorgen dat het connectie object geen delegatie methoden meer

kan aanroepen. Het probleem dat de connectie eerder is afgelopen dan de notificatie is ontvangen blijft nu nog wel bestaan. Een oplossing hiervoor zal later nog geïmplementeerd moeten worden.

Testen & onderzoek

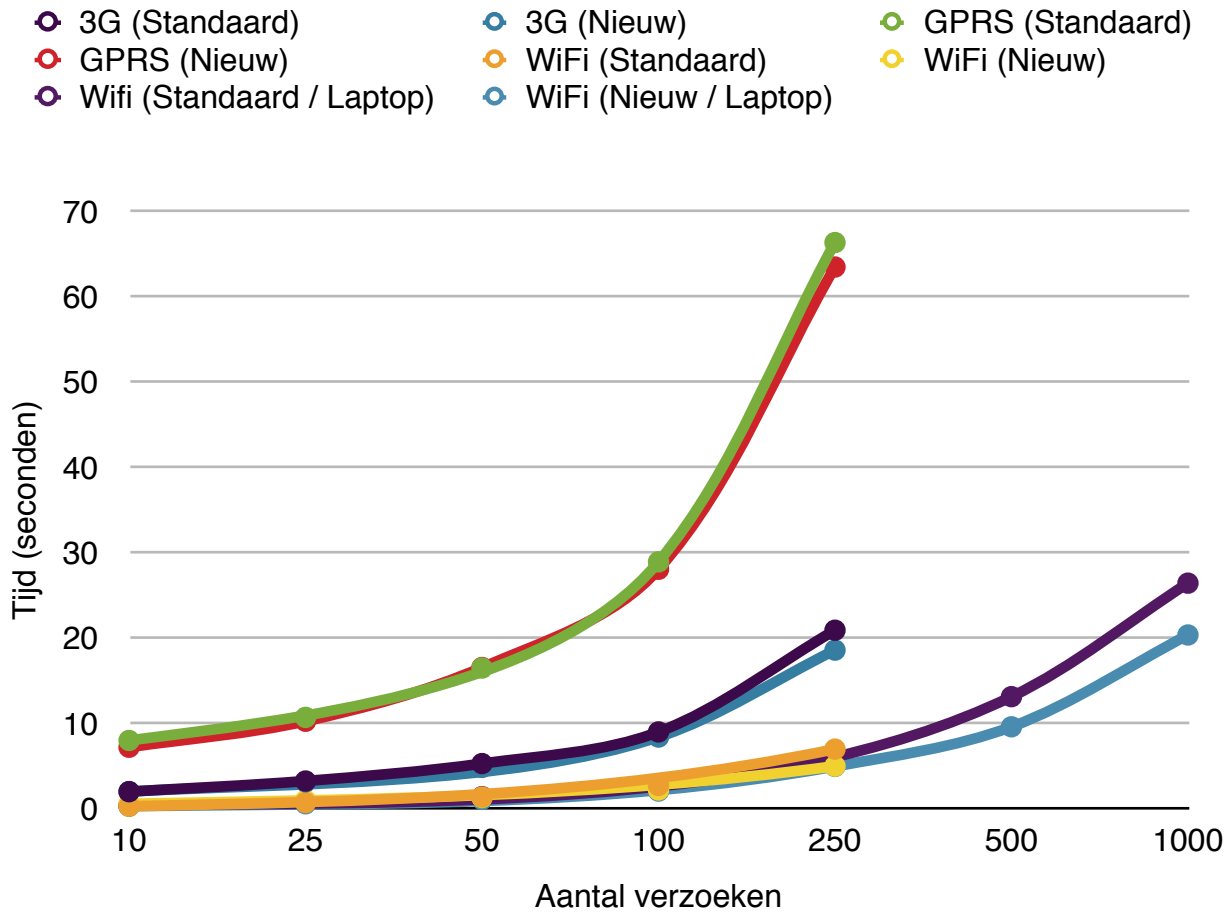
In dit onderdeel zal de onderzoeksvraag beantwoord worden over de prestaties van de scheduling methode en multithreading. Daarnaast zal een voorstel worden gedaan voor wanneer je de standaard methodes van Apple zou moeten gebruiken en wanneer hier een systeem voor geschreven moet worden.

Om aan te kunnen tonen welk van de twee scheduling algoritmes sneller is is een testsuite opgericht die hier antwoord op zal geven. Deze tests zullen bestaan uit een aantal identieke verzoeken aan de REST API die tegelijkertijd in de queue worden gezet. Aan de hand hiervan wordt een tijd registratie gestart die op de milliseconde nauwkeurig zal meten hoe lang het duurt om elke connectie succesvol af te ronden. Deze test zal zowel worden uitgevoerd door middel van de standaard methode als met de nieuwe scheduling methode.

De test zal worden uitgevoerd onder gesimuleerde omstandigheden, namelijk een WiFi, 3G of Edge connectie. LTE (4G) is achterwegen gelaten omdat dit qua snelheid vaak vergelijkbaar is met WiFi en alleen maar voor overbodige testresultaten zou zorgen. Per gesimuleerde connectie zullen tests met 10, 25, 50, 100, 250 & eventueel 500 & 1000 verzoeken worden uitgevoerd. Voor deze resultaten wordt de totale tijd opgeschreven en het gemiddelde van 10 tests genomen. Van dit gemiddelde wordt eerst de hoogste en laagste waarde weggestreept om er zo voor te zorgen dat uitschieters niet worden meegeteld, deze uitschieters kunnen aanwezig zijn omdat een internet connectie niet altijd even betrouwbaar is qua snelheid. Op deze manier wordt ervoor gezorgd dat de testresultaten betrouwbaarder zijn. In dit document zal een voorbeeld van deze testresultaten worden getoond in combinatie met een grafiek met alle test resulaten van de verschillende testcases. Voor alle test resultaten van beide methodes zie bijlage V.

#	10	25	50	100	250
1	2,497	2,954	5,263	8,841	20,998
2	2,177	3,121	5,251	8,59	19,912
3	1,959	3,516	5,406	8,483	21,433
4	1,91	3,265	4,903	8,931	20,363
5	1,918	3,294	5,09	9,255	20,963
6	1,962	3,286	5,352	8,914	19,735
7	1,962	3,214	5,216	9,006	21,444
8	1,917	3,345	5,51	9,251	22,294
9	2,091	3,298	5,509	9,295	21,193
10	2,093	3,296	5,28	10,109	21,045
Gemiddelde	**2,01**	**3,256**	**5,296**	**9,006**	**20,92**

Figuur 22: Voorbeeld tabel testresultaten (3G Standaard scheduling). (Zie bijlage V)



Figuur 23: Testresultaten performance onderzoek. (Zie bijlage V)

In figuur 23 is te zien dat de resultaten dicht bij elkaar liggen. In bijlage V met alle individuele grafieken is beter het verschil tussen de resultaten te zien. In de grafiek is het grootste verschil te zien bij de testcase “WiFi (Laptop)”, voor deze testcase zijn namelijk de meeste verzoeken gestuurd omdat dit het snelst van allemaal werd afgehandeld. Het verschil tussen de twee testen op 1000 verzoeken is rond de 6 seconden, dit is een aanzienlijk verschil dat je helaas alleen kan zien bij het uitvoeren van een groot aantal verzoeken. Bij alle andere aantallen is het tijdsverschil nagenoeg verwaarloosbaar. Uit deze testresultaten kunnen we concluderen dat het niet uitvoerbaar is om een eigen scheduling systeem te ontwikkelen tot een bepaalde hoogte. Als je namelijk weet dat je niet meer dan 10 verzoeken maximaal in de queue hebt staan zijn de snelheidswinsten minimaal of zelfs slechter ten opzichte van het standaard scheduling. Maar als je verder gaat naar meer dan 100 verzoeken beginnen de tijdsverschillen aanzienlijk te worden om de moeite waard te zijn. Alleen zijn er weinig applicatie voorbeelden die zoveel verzoeken tegelijk uitgevoerd moeten hebben. De keuze om de nieuwe scheduling niet te implementeren in de mobiele applicatie is dus uiteindelijk de juiste keuze geweest.

Het voordeel van deze testresultaten is het feit dat er tegelijkertijd ook een unit test is uitgevoerd op beide scheduling methodes. Door tegelijkertijd veel verzoeken in de queue te plaatsen van de standaard scheduling kan gelijk worden getest of deze queue deze allemaal goed afhandelt. Deze tests zijn uitgevoerd onder constante verbinding die niet weg viel waardoor er geen foutmeldingen zijn getoond. Dat betekent dat in de normale omgeving met een constante verbinding operaties betrouwbaar uitgevoerd worden.

Connecties stoppen & opnieuw proberen

Om connecties te kunnen stoppen wanneer tussentijds de verbinding wegvalt wordt gebruik gemaakt van de *suspend* methode. Deze methode zorgt ervoor dat de connectie afgebroken wordt

en eventuele progressie wordt opgeslagen wanneer nodig. De operatie bevindt zich dan in een status die wacht op een volgend signaal van de manager. Dit signaal wordt aangeroepen wanneer er weer een actieve verbinding beschikbaar is. Door deze functionaliteit is het mogelijk om internet afhankelijke operaties in de queue te houden totdat de connectie succesvol is afgesloten. Deze functionaliteit bevindt zich niet in de standaard scheduling methode en brengt een groot voordeel met zich mee omdat geen extra acties hoeven te worden ondernomen wanneer de verbinding wegvalt.

Een probleem dat zich voordoet met de connecties “suspenden” is het feit dat de delegatie methodes nog steeds worden aangeroepen terwijl we deze niet meer verwachten. Dit is eerder ook al beschreven in dit hoofdstuk en is op dit moment nog een bug in de nieuwe scheduling methode. Wanneer deze bug opgelost kan worden wordt ervoor gezorgd dat er een betrouwbaar stuk code wordt neergezet.

Deze implementatie zou in een later stadium nog kunnen worden toegevoegd aan de huidige methode van scheduling. Hierdoor worden minder acties verwacht wanneer een connectie mislukt om uit te voeren en zal de gebruiker minder gestoord worden met notificaties over een mislukte connectie.

De code is getest door middel van unit testing, de case die werd uitgevoerd door een aantal verzoeken in de queue te plaatsen waarna na bepaalde tijd de verbinding geforceerd wordt uitgezet. Door deze tests kwam bovengenoemd probleem aan het licht waardoor niet alle operaties succesvol zijn uitgevoerd. Gemiddeld 1 op de 4 keer dat de test is uitgevoerd mislukte het doordat een aantal connecties in een infinite loop bleven staan bij het weg vallen van de verbinding. Op deze manier stuurde de connectie een foutmelding en werd de connectie gemarkeerd als klaar.

Conclusie

Concluderend kiezen we er om de nieuwe scheduling methode niet in te voeren met als reden dat dit te veel moeite is n afweging met de te behalen winst in snelheid. Wel zullen een aantal features verwerkt worden in de huidige methode zoals het reageren op veranderingen in internet beschikbaarheid. In de meest gevallen is er voor het kleine aantal operaties dat uitgevoerd gaat worden geen tijdswinst waardoor deze scheduling methode niet noodzakelijk zal zijn. Dit beantwoordt een deel van de onderzoeksvraag hoe het beste een MultiThreaded systeem kan worden opgezet op het iOS platform. De geschreven methode zou gebruikt kunnen worden voor applicaties die extreem veel connecties en andere threads aanmaken, aangezien niet veel applicaties zoveel threads nodig hebben is het niet winstgevend om dit toe te passen. Daarom wordt aangeraden om gebruik te maken van de standaard *NSOperationQueue* in combinatie met zelfgeschreven *NSOperation* subklasse objecten. Een mooie feature van de nieuwe scheduling methode is dat dit zowel gebruikt kan worden voor desktop applicaties op het OSX platform als op het iOS platform. Applicaties waar dit in zou kunnen worden geïmplementeerd zullen vooral server applicaties zijn.

Review & Retrospective

Tijdens de Sprint review zijn de visuele aanpassingen getoond aan het ontwikkel team van Solware. Aangezien deze aanpassingen niet veel inhielden was de feedback van de Sprint niet groot. Aan bod kwam wel de vormgeving van de applicatie. Tijdens de Sprint was een afspraak met het vormgevende bedrijf ingepland om de applicatie te bespreken zodat ze een aantal schetsen en ideeën konden uitdenken voor de uiteindelijke vormgeving. Tijdens de review werden deze schetsen gereviewd om feedback te geven voordat de daadwerkelijke vormgeving gemaakt zou worden. Een feedback punt was bijvoorbeeld een historie knop die een compleet overzicht geeft van alle verstuurde verzoeken. Het team was van mening dat deze knop voor de gebruiker niet belangrijk zal zijn en dus ook niet gebruikt zal worden.

Tijdens de Sprint Retrospective kwam naar voren dat de planning in deze Sprint niet goed is afgehandeld aangezien de meeste onderdelen onder een bepaalde tijd zijn uitgevoerd, Dit is geen probleem als het maar niet te vaak gebeurt. Daarnaast was voor een ticket 5 uur ingepland en 25 uur gerealiseerd. Dat is een verschil van 20 uur op het ticket met nummer BC-637. Er zal beter gekeken moeten worden naar het totaal aantal uren dat op een taak zijn geregistreerd. Wanneer er wordt verwacht dat de maximale tijd wordt overstreken dient er ook echt gestopt te worden met ontwikkelen wanneer er ook andere onderdelen uitgevoerd dienden te worden.

Conclusie

Aan het einde van de zevende Sprint is een manier om efficiënt de verschillende achtergrondtaken af te handelen geïmplementeerd. Gekozen is voor de standaard methode omdat de nieuwe methode niet de snelheid winst geeft voor de doeleinden van onze mobiele applicatie. Deze methode heeft als nadeel dat er geen functie is geïmplementeerd die rekening houdt met het beschikbaar worden en wegvallen van verbindingen zonder de gebruiker lastig te vallen. Deze feature kan nog worden toegevoegd aan de applicatie in een later stadium.

Een onderzoeksvraag voor het afstudeerproject is op dit moment ook beantwoord door te onderzoeken wat de snelste en meest efficiënte methode is om achtergrond taken uit te voeren. Het gebruik van de nieuwe methode zal alleen effectief zijn voor programma’s die uitzonderlijk veel operaties (Threads) uit te voeren hebben. Voor normale applicaties zal daardoor een te verwaarlozen of geen snelheidswinst bereikt worden.

8. Validatie en Bugs

In eerste instantie zou de 8e Sprint in het teken staan van vormgeving van de mobiele applicatie. Maar tijdens de uitvoering hiervan bleek dat het veel tijd zou gaan kosten voor het bedrijf dat deze vormgeving zou gaan bouwen. Om geen tijd te verspillen is er daarom tijdens de Sprint voor gekozen om de Sprint meer te richten op testen van de mobiele applicatie en API. Dit deel van het verslag kan in combinatie gelezen worden met bijlage W, in deze bijlagen zijn alle resultaten en onderzoeks onderdelen van de Unit Tests terug te vinden.

Sprint Planning Meeting

Zoals u kunt zien in de onderstaande doelstellingen voor deze Sprint is het belangrijk dat de mobiele applicatie wordt vormgegeven. De doelstellingen die gedurende de Sprint zijn toegevoegd zijn schuin gemarkeerd. De taken die zijn aangemaakt voor vormgeving zullen naar de volgende Sprint worden verplaatst.

- Communicatie met het vormgeving bedrijf om feedback over voorbeelden te sturen.
- De verkregen vormgeving in de applicatie toepassen.
- *Bedenken van Unit test onderwerpen voor zowel de mobiele applicatie als de online API.*
- *Het uitvoeren van de bedachte Unit test en documenteren van de resultaten.*

De referentie naar de bovenstaande onderdelen en een aantal niet relevante onderdelen voor dit verslag zijn te vinden in de Product Backlog (Zie bijlage E). De nummers die hiertoe behoren zijn BC-647 tot en met BC-673. Zoals in de introductie te lezen was is de reden dat een aantal van deze taken tijdens de ontwikkeling zijn toegevoegd dat het ontvangen van de definitieve vormgeving langer duurde dan verwacht. Om geen kostbare tijd te verspillen is daarom tijd gestoken in het Unit Testen van bepaalde functies van de mobiele applicatie.

Unit Testing

Unit Tests zullen worden gebruikt om te controleren of bepaalde functionaliteit van een programma naar behoren werkt. Een voordeel van unit tests is dat door de automatische uitvoering hiervan snel kan worden bepaald of een onderdeel na enige aanpassing blijft werken. In dit geval is het bijvoorbeeld mogelijk dat een reguliere expressie tussentijds wordt gewijzigd omdat er een fout in is ontdekt. Door deze aanpassing is het mogelijk dat andere fouten optreden. Daarom is het belangrijk dat na vooral grote aanpassingen de test opnieuw wordt uitgevoerd om er zeker van te zijn dat de belangrijkste onderdelen nog steeds werken.

Meestal worden unit tests ontwikkeld tijdens de ontwikkel fase van een project, dit project bevindt zich op dit moment aan het einde van de ontwikkel fase. Een grote fase die nog doorgevoerd moet worden is het vormgeven van de mobiele applicatie, dit zal echter geen grote aanpassingen met zich meebrengen aan de functionaliteit van deze applicatie. Er is tijdens de ontwikkeling van dit project in eerste instantie nog niet nagedacht over het uitvoeren van unit tests. Code is tijdens het ontwikkelen vaak doorlopen door de ontwikkelaar waardoor een zekere vorm van juiste werking wordt gegarandeerd. Met een aantal onderdelen is dit echter niet uit te voeren, daarom zullen voor deze onderdelen unit tests ontwikkeld worden.

Een aspect dat vaak bij Scrum wordt toegepast is het feit dat in de eerste Sprints een aantal unit tests geschreven worden. Aan het einde van het project of een bepaalde Sprint moet de code

worden opgeleverd die aan de hand van deze tests goedgekeurd is. Voor dit project pakken we het andersom aan omdat deze tests pas geschreven zijn nadat alle functionele eisen van de mobiele applicatie zijn opgeleverd. Hier is in eerste instantie niet voor gekozen omdat er geen ervaring is met deze wijze van ontwikkelen, en aangezien het ontwikkelteam van Solware ook niet op deze wijze ontwikkelt was er ook geen mogelijkheid om dit project op weg te helpen op deze wijze.

Concluderend wordt hier aangetoond dat unit testen lastig te implementeren is en veel tijd kost. Maar wanneer de programmeur hier genoeg tijd voor neemt wordt een herbruikbare test suite ontwikkeld die gebruikt kan worden om aan te tonen of een aangepast stuk software nog steeds naar behoren werkt. De unit tests die hier zullen worden uitgevoerd zullen veruit niet alle onderdelen van het gedistribueerde systeem aanpakken, dit is voor een enkele ontwikkelaar niet in een tijdsvak van rond de 10 werkdagen uit te voeren op een goede wijze. De meest belangrijke onderdelen waar gebruikers mee te maken hebben zullen goed getest worden zoals elk ander onderdeel zou moeten worden getest.

Testen schrijven

Als voorbereiding op het schrijven van unit tests is eerst onderzocht welke onderdelen belangrijk zijn om getest te worden. Dit wordt in twee onderdelen gesplitst, namelijk het gedeelte dat voor de mobiele applicatie geschreven wordt en het gedeelte dat voor de online API wordt geschreven. Zo zal voor de mobiele applicatie vooral gericht getest worden op de volgende onderdelen.

- Validatie door middel van reguliere expressies (Naam, E-mail, Telefoonnummer etc.).
- Het verwerken van afbeeldingen van verschillende formaten.
- Het communiceren met de API, d.w.z. controleren of de juiste URL's worden gebruikt en de status 200 wordt teruggestuurd.
- Het genereren van authenticatie strings door middel van base 64 encodering.

Het schrijven van de unit tests voor de mobiele applicatie wordt met behulp van de ontwikkelsuite Xcode gedaan. Deze suite heeft een standaard Framework die ontwikkelaars helpt om makkelijker unit tests te schrijven. Een enkele test kan worden aangemaakt door een methode in de klasse te schrijven die start met *test* gevolgd door de naam van de test. Zo heb je bijvoorbeeld een *testEmailValidation* methode die de implementatie bevat om het valideren van email adressen te testen. Een test mislukt wanneer een aanname fout wordt uitgevoerd, hier worden Framework methodes voor gebruikt die de vorm *XCTAssert* hebben. Wanneer deze methodes een foute aanname detecteren wordt een foutmelding gegenereerd en getoond in een test lijst. Daarnaast kan door middel van de *XCTFail* methode een generieke foutmeldingen worden geactiveerd met de daarbij opgegeven beschrijving. Voor een compleet inzicht in de code die de geschreven tests uitvoeren verwijs ik naar de bijlage H. Daarnaast is een document geschreven dat een gedetailleerd resultaat en beschrijving van de verschillende tests die uitgevoerd zijn geeft (Zie bijlage W).

Elke unit test is op een wijze geschreven zodat de test uitgevoerd kan worden ook al wordt de implementatie van een methode die getest wordt aangepast. Dat wil zeggen dat de aanroep van de test altijd gelijk blijft nadat de test is geschreven. Op deze manier wordt geen tijd gestoken in het aanpassen van de unit tests implementatie. Deze wijze van testen wordt ook wel Test Driven Development (TDD) genoemd. Dit is echter niet de methode die wij gebruiken aangezien bij TDD eerst tests worden geschreven waarna de implementatie van de applicatie wordt geschreven. Er wordt getracht van tests naar werkende code toe te werken. Bij dit project is er niet gekozen om deze wijze van ontwikkeling te gebruiken omdat hier in eerste instantie niet over na is gedacht, en omdat het lastig is om vanaf het begin te weten welke tests je wilt maken. Hierdoor kan de ontwikkelaar te veel tijd in het testen steken waardoor de werking van de applicatie achter kan lopen op de voortgang die verwacht wordt per uitgevoerde Sprint.

Bugs

Tijdens het testen zijn een aantal bugs gevonden die anders tot vervelende gebruikers ervaringen kan zorgen in de mobiele applicatie. Een voorbeeld hiervan is een geheugen afhandeling bug die ertoe leidt dat een operatie manager in een status bleef waardoor geen nieuwe achtergrond taken werden uitgevoerd. Wat betekent dat de applicatie volledig nutteloos wordt. Deze bug is alleen tijdens een unit test nagebootst omdat een uiterste situatie is uitgebeeld, deze situatie zou niet mogelijk zijn in de mobiele applicatie om na te bootsen. Toch is het belangrijk om deze bug op te lossen mocht de code namelijk hergebruikt worden in een andere applicatie.

Een andere belangrijke bug die opgelost is is een authenticatie bug. Tijdens authenticatie met de API was het mogelijk om informatie te lezen van andere gebruikers door verkeerd geïmplementeerde code. Dit was echter eenvoudig op te lossen door een bepaalde waarde om te zetten. Hierdoor kunnen onbevoegden niet bij informatie van andere gebruikers van de applicatie. Naast deze bugs zijn een aantal kleinere problemen opgelost die niet tijdens de ontwikkeling van de applicatie zijn opgevallen.

Conclusie

Unit tests zijn een belangrijk onderdeel bij het ontwikkelen van een dergelijke applicatie zoals binnen dit project. Vooral het gedeelte van het systeem waarbij de applicatie met de API kan praten is een onderdeel dat feilloos moet werken omdat dit de bron is van informatie voor en van de mobiele applicatie. Daarom zijn de tests vooral gericht op het simuleren van alle mogelijke API aanroepen met juiste en foute parameters. Ook de validatie van informatie is uitbundig getest omdat de informatie voorziening een belangrijk onderdeel is voor het gehele gedistribueerde systeem.
Het vormgeven van de mobiele applicatie is verplaatst naar de volgende Sprint omdat de definitieve versie van deze versie langer op zich liet wachten dan gepland. Om geen tijd te verspillen is ervoor gekozen om deze Sprint in het teken van testen te laten staan.

Algemene Conclusie

Tot nu toe is er in het afstudeerdossier nog geen beschrijving gegeven van het vormgevende proces voor de mobiele applicatie. Dit proces zal uitgevoerd worden in de laatste Sprint. Op het moment is de mobiele applicatie zover dat hij bijna uit gegeven zal worden, dit hangt samen met het naar behoren functioneren van de afstudeerder. De deadline voor de mobiele applicatie is geplaatst op 8 november 2013. Voor verdere informatie betreffende de vormgeving verwijs ik u naar bijlage Y.

Het onderzoek en Proof Of Concept voor het MultiThreaded systeem op de mobiele applicatie is succesvol afgerond en heeft positieve resultaten opgeleverd. Het is namelijk zo dat in extreme gevallen dit Proof Of Concept beter presteert dan de standaard beschikbare versie. Er is echter voor gekozen deze niet te gebruiken in de mobiele applicatie die opgeleverd gaat worden omdat er enkele onstabiliteit bestaat. Deze instabiliteit kan echter nog opgelost worden. Op dit moment zal als vervanging de normale versie worden opgeleverd.

Ook de implementatie van veiligheid is succesvol geïmplementeerd. Er is onderzocht wat de beste wijze van authenticatie is tussen de API en de mobiele applicatie zodat de applicatie herkent zou worden. De applicatie communiceert door middel van SSL en token authenticatie met de API om er zo voor te zorgen dat geen tussenpersoon in de connectie mogelijk is. Ook door middel van token authenticatie is het niet mogelijk om andermans informatie op te vragen, wijzigen of verwijderen.

Voor het bewijs van het behalen van competenties verwijs ik u naar bijlage Z. In deze bijlage wordt per competentie beschreven hoe en waarom de afstudeerder denkt dat deze competentie op een voldoende niveau is uitgevoerd.

Tijdens het doorlopen van de Sprints is veel tijd besteedt aan de communicatie met de opdrachtgever en intern via de Proxy Product Owner. Hierdoor is het ontwikkelen van het eindproduct in goed overleg gegaan met de opdrachtgever aan de hand van presentaties. Door deze communicatie met de opdrachgever zal het opgeleverde product aan de behoefte van de opdrachtgever voldoen. De laatste afspraak met de opdrachtgever om het eindresultaat te bespreken moet nog plaatsvinden. Aan de hand van deze handelingen en communicatie kunnen we aannemen dat de afstudeerder voldoet aan de competentie voor het achterhalen van de behoeften van de belanghebbenden. Ook is er veel overleg geweest met de eigenaar van Solware ook deze belanghebbende heeft voldoende bijdrage en feedback kunnen geven tijdens de ontwikkeling van het product.

Het realiseren van het gedistribueerde systeem is in 10 grote stappen (Sprints) uitgevoerd. Waarbij bij elke stap een nieuwe functionaliteit werd toegevoegd. Deze wijze van ontwerpen en ontwikkelen is vrij verfijnd in combinatie met de feedback die van het ontwikkelteam van Solware is verkregen. Aan de hand van deze feedback kan worden gediscussieerd over bepaalde ontwerpen en ontwikkelmethodes waardoor vaak een beter ontwerp kan worden bedacht. Zo is het vaak aan bod gekomen dat er te veel onnodige onderdelen in een vroeg stadium toegevoegd zouden gaan worden aan de mobiele applicatie en de REST API. Zo is bijvoorbeeld de adresgegevens opzoek methode in een vroeg stadium toegevoegd terwijl dit niet nodig was. Dit is na de betreffende Sprint Review opgemerkt waarna beter werd gelet op het efficiënt realiseren van het gedistribueerde systeem. Door eerst de verreisde functionaliteit te realiseren kan sneller een werkend systeem worden opgezet zonder extra functionaliteit waar de opdrachtgever in eerste instantie niet naar heeft gevraagd. Hierbij zijn de stappen die de afstudeerder doorliep aangepast aan de behoeften van de opdrachtgever.

Nawoord

Ik heb veel geleerd tijdens het gehele afstudeertraject en ben daarom alle collega's van Solware erg dankbaar voor het aanbieden van de opdracht. Ik heb het gevoel dat ik nu veel verstand heb van het ontwikkelen van software binnen een bedrijf waar het van belang is om op uren en functionele eisen te waken. Mijn begeleider binnen Solware heeft het evaluatieformulier ingevuld (zie bijlage AB) voor meer informatie over mijn integratie binnen het Solware team en mijn algehele inzet.

Verder ben ik blij met de opgeleverde applicaties en ben op dit moment erg benieuwd hoe er wordt gereageerd door de gebruikers wanneer beide applicaties in productie worden gezet.

Bronnen

BouwMeester, (1964). BouwMeester website. Verkregen op 14 oktober, 2013 van http://www.bouwmeester.org

Solware, Solware website, Verkregen op 14 oktober, 2013 van http://www.solware.nl

Apple Inc. (1976), Cocoa Touch, Verkregen op 14 oktober, 2013 van https://developer.apple.com/technologies/ios/cocoa-touch.html

Schwaber, K., & Beedle, M. (2001). *Agile Software Development with Scrum.* Prentica Hall

Apache, Apache Subversion, Verkregen op 14 oktober 2013 van http://subversion.apache.org

Oracle, (2013), MySQL MyISAM engine, Verkregen op 14 oktober 2013 van http://dev.mysql.com/doc/refman/5.0/en/myisam-storage-engine.html

Oracle, (2013), MySQL InnoDB engine, Verkregen op 14 oktober 2013 van http://dev.mysql.com/doc/refman/5.0/en/innodb-storage-engine.html

Apple Inc., (2013), iOS ontwikkelaar bibliotheek. Verkregen op 14 oktober, 2013 van https://developer.apple.com/library/ios/navigation/

Apple Inc., (2013), Key-Value Coding referentie. Verkregen op 14 oktober 2013 van https://developer.apple.com/library/ios/documentation/general/conceptual/devpedia-cocoacore/KeyValueCoding.html

Apple Inc., (2013), Delegatie referentie. Verkregen op 14 oktober, 2013 van https://developer.apple.com/library/ios/documentation/general/conceptual/devpedia-cocoacore/Delegation.html

OmniTE Labs, (2007). JSend informatie. Verkregen op 14 oktober, 2013 van http://labs.omniti.com/labs/jsend/

Harvest, (2006). Harvest tijd registratie website. Verkregen op 14 oktober, 2013 van http://www.getharvest.com/

Apple Inc., (2013) CoreData referentie. Verkregen op 14 oktober, 2013 van https://developer.apple.com/technologies/mac/data-management.html

IETF., (2013) IETF RFC pagina. Verkregen op 14 oktober, 2013 van http://www.ietf.org/rfc.html

IETF., (2013) IETF startpagina. Verkregen op 14 oktober, 2013 van http://www.ietf.org/

IETF., (2013) RFC-2388 referentie pagina. Verkregen op 14 oktober, 2013 van http://www.ietf.org/rfc/rfc2388.txt

postcode.nl B.V., (2013) Postcode API database website. Verkregen op 14 oktober, 2013 van https://www.postcode.nl/

Apple Inc., (2013) Apple Push Notification Service referentie. Verkregen op 14 oktober, 2013 van https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html

The Clang Team, (2007) Clang Automatic Reference Counting referentie. Verkregen op 14 oktober, 2013 van http://clang.llvm.org/docs/AutomaticReferenceCounting.html

Bijlagen

Bijlage A - *Organisatie grafiek van Solware.*

Bijlage B - *Solware Afstudeeropdracht.*

Bijlage C - *Solware Enterprise Framework, SEF*.

Bijlage D - *Systeem Architectuur*.

Bijlage E - *Product & Sprint Backlogs.*

Bijlage F - *Plan van Aanpak.*

Bijlage G - *User Stories.*

Bijlage H - *Inhoudsopgave code.*

Bijlage I - *Database ontwerp (Sprint 1).*

Bijlage J - *Operatie management ontwerp (Sprint 1).*

Bijlage K - *Standaard Objective-C guide.*

Bijlage L - *CoreData ontwerp (Sprint 2).*

Bijlage M - *Flow ontwerp afbeeldingen toevoegen (Sprint 3).*

Bijlage N - *Ontwerp ondersteuning meerdere iOS versies. (Sprint 4).*

Bijlage O - *Ontwerp interne communicatie reparatieverzoeken (Sprint 5).*

Bijlage P - *CoreData ontwerp tweede versie (Sprint 5).*

Bijlage Q - *Documentatie API methoden (Sprint 5).*

Bijlage R - *Sequentie diagram CSR & SSL (Sprint 6)*.

Bijlage S - *Sequentie diagram SMS verificatie (Sprint 6).*

Bijlage T - *Ontwerp zelfgeschreven scheduling methode (Sprint 7).*

Bijlage U - *Flow diagram van de scheduling methode (Sprint 7).*

Bijlage V - *Test resultaten voor Operatie scheduling (Sprint 7).*

Bijlage W - *Onderzoek en resultaten voor Unit Testing (Sprint 8).*

Bijlage X - *Afstudeerplan.*

Bijlage Y - *Screenshots mobiele applicatie.*

Bijlage Z - *Wijze van behalen van Competenties*

Bijlage AA - *Use Cases*

Bijlage AB - *Evaluatieformulier*