Cross-platform mobiele-app authenticatie voor een bestaand urenregistratie systeem

Afstudeerverslag

Auteur	Allard Soeters
Studentnummer	11114762
Studie	Informatica
School	Haagse Hogeschool
Eerste examinator:	G.A. Mijnarends
Tweede examinator:	H.G.J. Bechet
Opdrachtgever:	R. Flohil
Afstudeerbegeleidster:	P. Hijl
Technisch Begeleider:	E. van Alebeek
Business Unit manager:	H. Brands
Plaats	Den Haag
Datum	05-10-2015
Bedrijf	Info Support B.V.

Referaat

Soeters, Allard, Het bouwen van een cross-platform applicatie met de focus op security, Veendendaal, Info Support, 2015

Afstudeerverslag van Allard Soeters, geschreven in het kader van het afstuderen bij de opleiding Informatica aan de academie van ICT & Media aan de Haagse Hogeschool in Den Haag.

Het verslag behandelt het proces dat doorlopen is tijdens de afstudeerperiode van Allard Soeters bij Info Support te Veenendaal. De opdracht stond in het teken van het selecteren van een goed authenticatie protocol voor een cross-platform mobiele applicatie voor het huidige urenregistratiesysteem, het ontwerpen en bouwen hiervan.

Descriptoren:

- Scrum
- C# (.Net)
- REST
- Webservice
- Xamarin forms
- Web api 2
- MVC
- SRP
- Security
- KPMG-pakketselectie
- Encryptie
- Webapplicatie

Voorwoord

Dit verslag is geschreven door Allard Soeters, student Informatica aan de Haagse Hogeschool te Den Haag. Dit verslag is geschreven naar aanleiding van het afstuderen van Allard Soeters. Dit verslag wordt ter beoordeling voorgelegd aan Gerard Mijnarends, Rianne Bechet en een extern gecommitteerde om inzicht te geven in de uitgevoerde werkzaamheden tijdens de afstudeerperiode.

In dit voorwoord wil ik een aantal mensen danken die mij tijdens deze periode begeleid hebben. Dit zijn Henk Brands, Rutger Flohil, Erma van Alebeek en Pascalle Hijl. Deze begeleiders vanuit Info Support zijn van grote steun voor mij geweest. Daarnaast hebben een aantal medewerkers vanuit Info Support (Martijn de Vries, Rob te Broekhorst, Léon Bouquiet en Gert Jan Timmerman) mij ondersteund met hun technische expertises op verschillende vlakken.

Vanuit de Haagse Hogeschool heb ik van Gerard Mijnarends en Rianne Bechet op verschillende momenten tijdens de afstudeerperiode goed bruikbare kritische feedback ontvangen. Daarnaast wil ik Willem van Vliet bedanken voor het helpen definiëren, het kaderen, van de opdracht.

Tot slot wil ik Bart Bijl, Hanjo de Wit, Pascalle Hijl, Tom Keim, Maikel Hofman, Maarten Koene en Ruben Zorgman bedanken voor de prettige werksfeer op de werkvloer in Zoetermeer en/of Veenendaal.

Allard Soeters
Veenendaal, 05-10-2015

Inhoudsopgave

1. Inleiding

Dit afstudeerverslag beschrijft het proces van het uitvoeren van de afstudeeropdracht. De opdracht is uitgevoerd voor Info Support. Het doel van dit verslag is om aan de examinatoren en de extern gecommitteerde te tonen dat ik als afstuderende de werkzaamheden tijdens de afstudeerperiode (20 weken) op niveau heb kunnen uitvoeren. Zodanig dat zij een oordeel kunnen geven of ik voor de afstudeeropdracht geslaagd ben.

In dit verslag komt een aantal onderwerpen aan bod: informatie over het bedrijf Info Support, wat zij doen en waar zij voor staan. In hoofdstuk 3 wordt de opdracht beschreven. Hierin wordt beschreven wat het probleem en het uiteindelijke doel is van deze opdracht. Vervolgens wordt de huidige situatie beschreven en een mogelijke oplossing weergegeven. In hoofdstuk 5 is de aanpak van het project geschetst en welke methodiek is er gebruikt. In hoofdstukken 6 tot en met 9 worden alle sprints beschreven. Per sprint worden een aantal onderdelen van de applicatie behandeld. Ook worden de belangrijke beslissingen benoemd en uitgewerkt.
In hoofdstuk 10 wordt het project afgerond. Daarin worden de producten en proces geëvalueerd. Ook is het advies wat verder nog gedaan moet worden voor de applicatie beschreven. In hoofdstuk 11 is iedere beroepstaak die gehaald moest worden uitgelegd en aangegeven of deze beroepstaak wel of niet gehaald is.

2. Organisatie

Info Support is een softwarebedrijf met meer dan 400 medewerkers. Het is in 1986 opgericht door Toon Jansen. Het hoofdkantoor van Info Support heeft zich in Veenendaal gevestigd daarnaast heeft Info Support kenniscentra in Utrecht en Veenendaal. Hier worden trainingen voor interne maar ook externe IT'ers geven. Info Support biedt daarnaast twee plekken om af te studeren, in Zoetermeer en Tilburg. Sinds enkele jaren heeft Info Support ook een kantoor in België.

Info Support maakt gebruik van de Java en Microsofttechnieken. Denk aan C# en .Net. Info support heeft een eigen ontwikkelstraat ontwikkeld genaamd "Endeavour".

Info Support heeft vier kenwaarden: Soliditeit, Integriteit, vakmanschap en Passie. Deze kernwaarden staan voor het bedrijf. Tijdens de afstudeerperiode ben ik deze kernwaarden tegengekomen in verschillende afspraken en manier van werken.
Hieronder zijn deze kernwaarden afgebeeld. Deze afbeeldingen komen uit de hal van het hoofdkantoor in Veenendaal. Zodra je hier binnenkomt vallen deze afbeeldingen op waardoor meteen duidelijk is waar het bedrijf waarde aan hecht.

Figuur 1 Integriteit

Figuur 2 Soliditeit

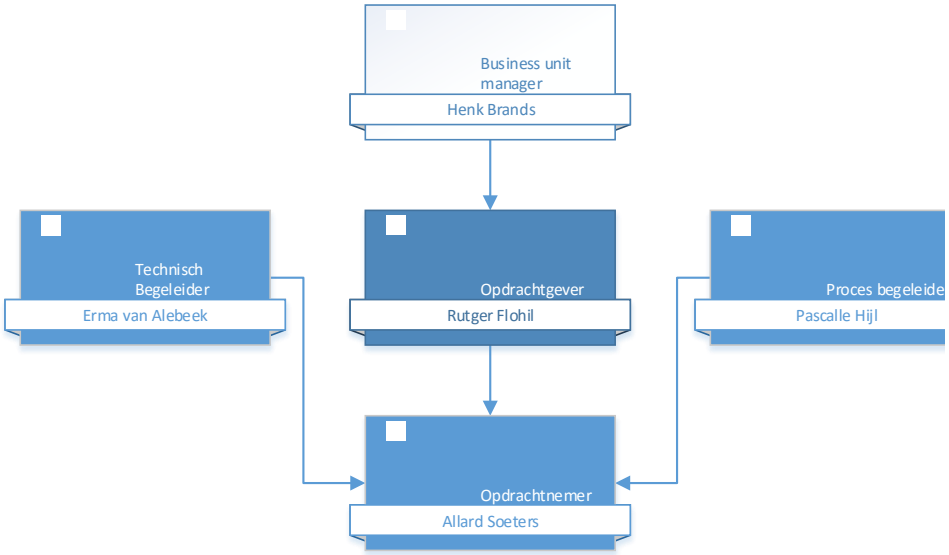
Figuur 4 passie

Figuur 3 vakmanschap

Info Support heeft veel klanten. Deze bevinden zich in meerdere sectoren waaronder de overheid, financiën, handel & industrie en zorg & verzekeringen. Hieronder is een afbeelding met een selectie van de klanten.

Figuur 5 Klantenprofiel

Tijdens het afstuderen ben ik onderdeel van de tak data solutions & mobile development. Het hoofd van deze unit is Henk Brands. Hieronder is een organogram voor de verdeling van dit project. In Bijlage E – Plan van aanpak worden de taken uitgelegd.



Figuur 6 Organogram

Naam	Functie	Omschrijving
Henk Brands	Business unit manager	Business unit manager zorgt ervoor dat de afdeling goed werkt
Rutger Flohil	Opdrachtgever	Voor de opdrachtgever wordt het project uitgevoerd
Erma van Alebeek	Technisch begeleider	De technisch begeleider biedt technische ondersteuning aan de opdrachtnemer
Pascalle Hijl	Procesbegeleider	De procesbegeleider zorgt ervoor dat de opdrachtnemer
Allard Soeters	Opdrachtnemer	De opdrachtnemer voert de opdracht uit

3. Opdrachtomschrijving

Iedere week op vrijdagmiddag moeten de medewerkers van Info Support de uren die zij gewerkt hebben registreren via de online omgeving Coress. De opdrachtgever hoort steeds meer medewerkers klagen over Coress. De afstudeeropdracht is de urenregistratie gebruikersvriendelijk en efficiënter te maken.

3.1. Huidige situatie

Coress is een product, ontwikkeld door Info Support, om projecten in bij te houden. Per project moeten de medewerkers van Info Support aangeven hoeveel zij gewerkt hebben aan dat project. Met het aantal uur dat gewerkt is aan een project kunnen de klanten gefactureerd worden. Coress is via het intranet van Info Support beschikbaar waardoor niemand buiten het netwerk van Info Support bij het systeem kan. Omdat veel medewerkers consultants zijn en bij klanten werken, kunnen deze medewerkers niet direct bij Coress. Om dit probleem op te lossen wordt via de firewall toegang gegeven tot Coress. De medewerkers moeten via de firewall inloggen.

3.2. Probleem

Het huidige systeem is niet gemaakt om uren te registreren via een smartphone. Veel medewerkers zijn veel onderweg en hebben dan niet de mogelijkheid om hun uren in te vullen via de laptop. Op dit moment moet de medewerker te veel handelingen verrichten voordat hij zijn uren in kan vullen. Dit wordt als vervelend ervaren en hierdoor wordt deze vereiste handeling vaak uitgesteld. Als de uren niet ingevuld worden kan er geen factuur naar de opdrachtgever gestuurd worden. Om deze medewerkers op tijd hun uren in te laten vullen kost Human Resources onnodig veel tijd. Zij moeten steeds iedere medewerker controleren en contact met de medewerker opnemen als de uren niet ingevuld zijn.

3.3. Resultaat

Het resultaat zal een gebruiksvriendelijker Coress zijn dat gebruikt kan worden via de smartphones van alle medewerkers van Info Support. Hierdoor zal het invullen van de uren voor de medewerkers makkelijker worden en zal Human Resources minder tijd kwijt zijn aan het bellen en mailen van de medewerkers die hun uren niet hebben ingevuld.

Daarnaast zal het resultaat zijn dat nader onderzoek naar de mogelijkheden en risico's naar een beveiligde verbinding met het interne netwerk van Info support nodig zijn. Als Systeembeheer deze mogelijkheden en risico's accepteert zal de applicatie in productie genomen worden.

3.4. Effect

Omdat de medewerkers minder tijd kwijt zijn aan het invullen van de uren die zij gemaakt hebben, kunnen zij meer tijd aan de klant besteden waardoor geld bespaard kan worden. Ook zal Human Resources efficiënter kunnen werken.

Deze applicatie kan eventueel als blauwdruk voor toekomstige interne applicaties gebruikt worden.

3.5. Afbakening

De focus ligt op de manier hoe de mobiele applicatie gemaakt wordt waarbij vooral gelet wordt op de beveiliging van de mobiele applicatie. Beveiliging is een ruim begrip dus is in de opstartfase is een afbakening bepaald met systeembeheer. Systeembeheer moet de applicatie goedkeuren voordat deze in productie genomen kan worden. De volgende aspecten van beveiliging worden behandeld:

- Authenticatie
- Transport beveiliging

De manier van authenticatie en transport zal als een proof of concept in de applicatie zelf gebouwd worden. De applicatie zal modulair opgebouwd worden waardoor later de applicatie makkelijk omgebouwd kan worden als er voor andere manieren van transport en authenticatie gekozen worden.

3.6. Vooraf besproken mogelijkheden

Om Coress gebruiksvriendelijker te maken voor smartphones, platform onafhankelijk, kan dit bereikt worden op de volgende manieren:

1. Een nieuwe mobiel vriendelijke userinterface (html5 & css3)
2. Native mobiele applicaties maken voor IOS, Android en Windows Phone apart
3. Een native cross-platform mobiele applicatie maken met behulp van Xamarin

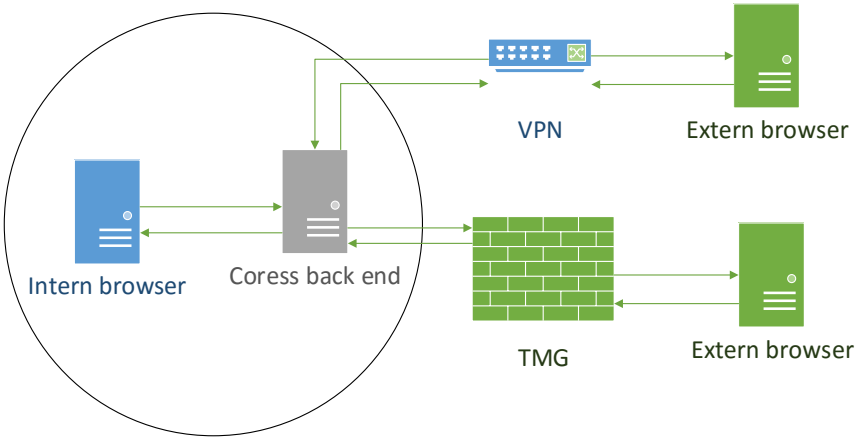
De opdrachtgever vindt het belangrijk om de medewerkers van Info Support zo min mogelijk stappen hoeven te nemen om hun uren in te vullen. Eén van de stappen, wat tevens de grootste is, is het inloggen via de firewall van Info Support threat management gateway(TMG) van Microsoft. Als een mobiel vriendelijke userinterface gemaakt wordt, zal er alsnog ingelogd moeten worden via de TMG. De eerste manier viel hierdoor meteen al af. Een native applicatie zal beter passen bij de oplossing van het probleem omdat de medewerkers vaak onderweg zijn en niet altijd internet heeft. Met een native applicatie hoeft de medewerker niet online te zijn om zijn uren in te vullen. De uren kunnen zodra hij online is gesynchroniseerd worden. Ook reageert een native applicatie sneller omdat de afbeeldingen en stijl niet bij ieder request opgehaald hoeven te worden, uit de cache of van de server.

De keuze tussen de tweede en derde manier heeft te maken met de hoeveelheid werk dat gedaan moet worden om de oplossing te realiseren. Als voor de drie platformen allemaal een aparte applicatie gemaakt moet worden gemaakt kost dit meer tijd dan één cross-platform mobiele applicatie. Omdat deze opdracht naast het maken van een gebruiksvriendelijke Coress ook een focus heeft op de beveiliging van de oplossing is vooraf besloten om een cross-platform mobiele applicatie te maken met behulp van Xamarin.

De reden waarom dit project de focus op security heeft gelegd komt doordat de beheerder van het interne netwerk van Info Support nog geen mobiele applicaties accepteert die gebruik maken van het interne netwerk. Hierdoor is een nieuwe soort van beveiliging voor Info Support nodig. Deze beveiliging is nodig omdat er vertrouwelijke gegevens vanuit de interne omgeving moeten komen. Ook zullen vertrouwelijke gegevens naar de interne omgeving verstuurd moeten worden. De afdeling systeembeheer staat dergelijke communicatie alleen maar toe als de veiligheid gewaarborgd blijft. Coress bevat vertrouwelijke gegevens, alle projecten en klanten zijn in Coress opgeslagen. Het is niet de bedoeling dat deze gegevens van klanten op straat belanden. De mobiele applicatie moet hier rekening mee houden. Om deze verbinding zo veilig mogelijk te maken is een belangrijk nevendoel van dit project.

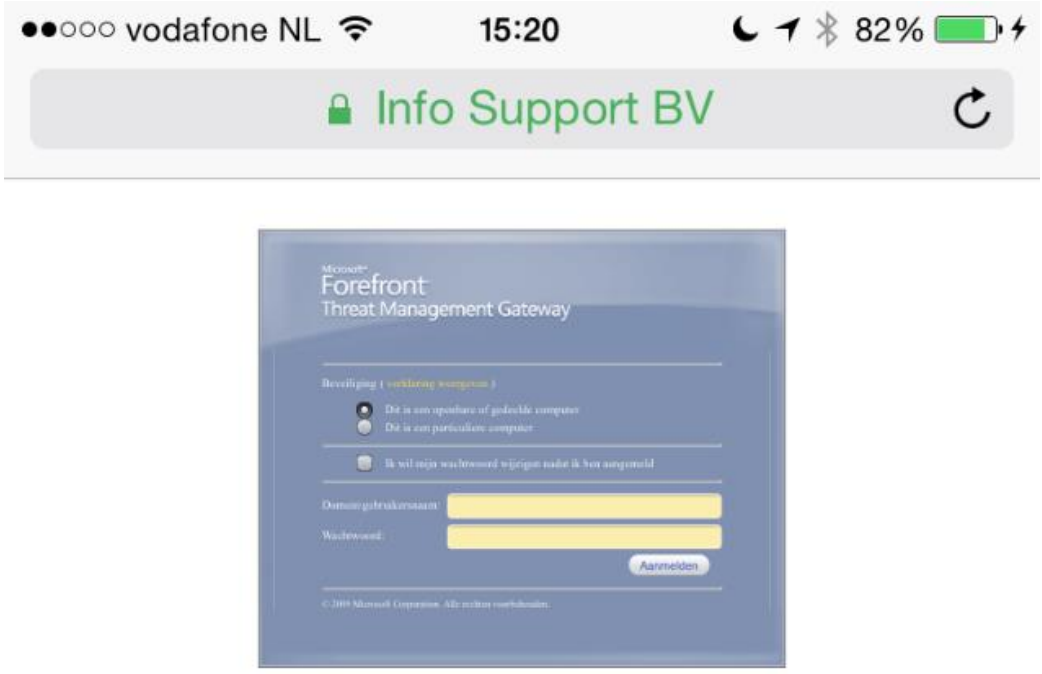
4. Huidige situatie

De Coress webapplicatie staat in het interne netwerk. Dit zorgt ervoor dat de medewerkers die op het netwerk van de klant werken geen gebruik kunnen maken van Coress. De medewerkers van Info Support kunnen via een vpn verbinding op het netwerk van Coress komen. Dan kunnen zij wel van Coress gebruik maken. De medewerkers vinden het omslachtig om iedere keer via de VPN hun uren in te invullen. Smartphones kunnen geen gebruik maken van de VPN dus is er een uitzondering voor Coress gemaakt. Als de medewerker ingelogd is via Threat Management Gateway(TMG), de firewall van Info Support, mag hij Coress gewoon gebruiken.
In Figuur 7 Huidig netwerk wordt duidelijk gemaakt wat de huidige situatie m.b.t. het netwerk is.

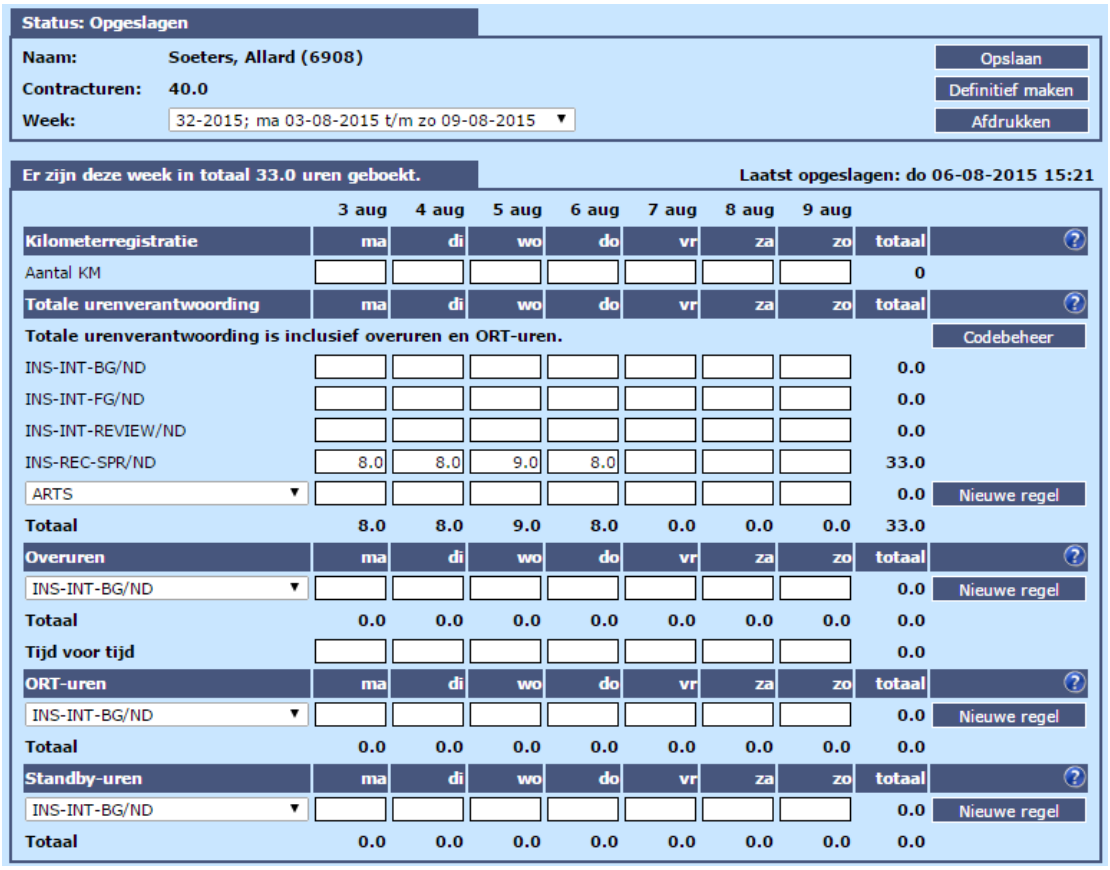


Figuur 7 netwerk diagram huidig netwerk

Om Coress te kunnen gebruiken moet ingelogd worden op de TMG. TMG heeft geen mobiel vriendelijke inlog pagina. Dit is te zien in Figuur 8 Inlogscherm Coress. De invoervelden zijn te ver uitgezoomd waardoor het moeilijk is om op de velden te klikken om de gegevens in te vullen.



Figuur 8 niet mobiel vriendelijke inlogscherm TMG



Figuur 9 huidig Coress via de browser benaderd

Zodra ingelogd is, wordt de huidige versie van Coress getoond. In “Figuur 9 Huidig Coress” is dit te zien.
Hierin kunnen de medewerkers per week zijn uren invullen. In dit scherm staan er vier verschillende soorten uren:

1. Gewerkte uren
2. Overuren
3. Standby uren
4. ORT-uren

Al deze soorten uren zijn onderverdeeld in uur codes. Per dag kunnen meerdere uren aan verschillende uur codes gekoppeld worden. Deze uur codes zijn verbonden aan projecten. Hierdoor kan de afdeling Sales facturen opmaken. Daarnaast kunnen er ook diverse codes aan de week gekoppeld worden. Dit is nodig als er feestdag geweest is of als de medewerker ziek is.

Als de medewerker nog niet alle uren ingevuld heeft kan hij tussentijds via de knop ‘opslaan’ zijn reeds ingevulde gegevens opslaan. Zodra hij alles van die week ingevuld heeft en hij wil dat zijn manager dit gecontroleerd en goedkeurt, drukt hij op de knop definitief maken.

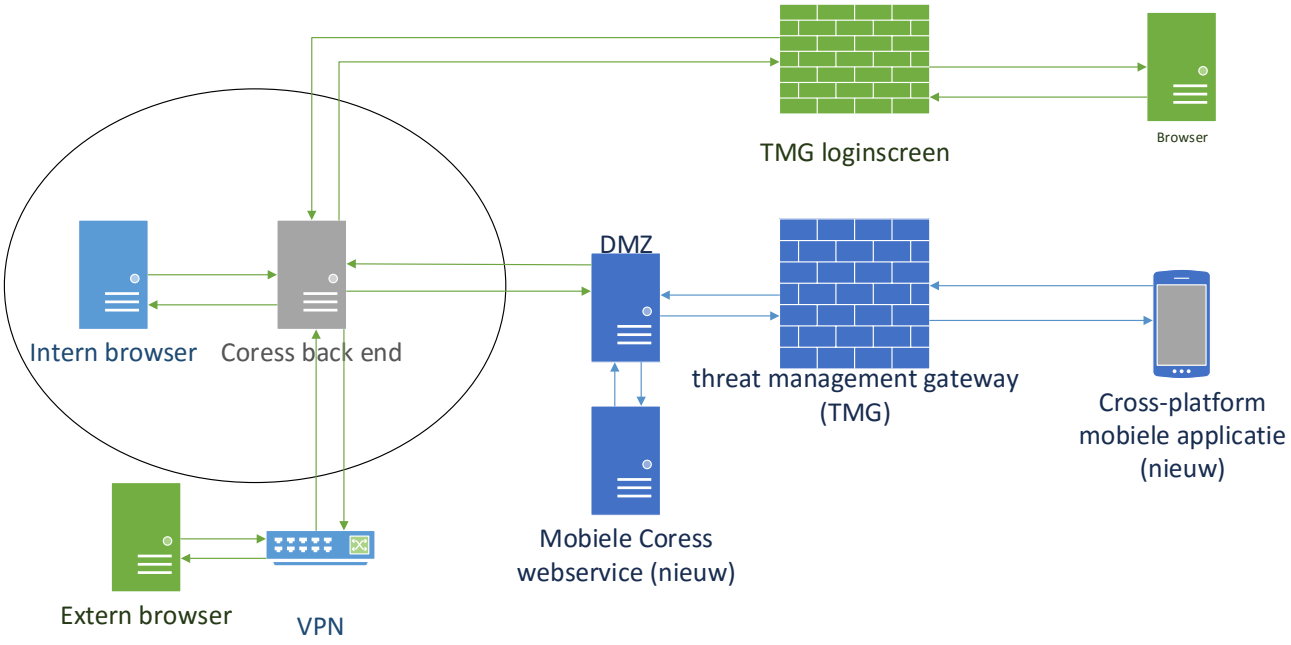
Via een pc is dit scherm makkelijk en overzichtelijk maar voor een mobiele telefoon zijn de vakjes te klein waardoor ingezoomd moet worden om het juiste vakje aan te klikken om er een getal in te vullen.

4.1. Mogelijke oplossing

Zoals in hoofdstuk 3.6 beschreven is gekozen om een cross-platform native mobiele applicatie te maken. Omdat deze applicatie op Android, iOS en Windows Phone moet draaien, is gekozen voor Xamarin. Met Xamarin kan voor deze drie devices één applicatie gemaakt worden in C#. Xamarin zorgt er dan voor dat de code in de native taal gecompileerd wordt. Hierdoor wordt op het device zelf native code uitgevoerd.

Coress heeft een eigen webservice. Deze webservice zorgt voor de business logica en de databaseconnectie. Deze webservice is ideaal als backend voor de mobiele applicatie omdat dan niet alle business logica in de applicatie zelf gebouwd moet worden. Het probleem is dat deze webservice in het interne netwerk zit. De medewerkers die gebruik willen maken van de applicatie zijn meestal niet verbonden met het interne netwerk. Om zulke problemen op te lossen heeft Info Support een demilitarized zone, DMZ, ingericht. In deze zone kan wel verkeer van buiten het interne netwerk komen. De applicaties die in deze zone staan kunnen wel met de applicaties die intern staan communiceren. De applicaties in de DMZ zijn een soort tussenpersonen.

In onderstaand diagram, uitwerking op figuur 7, is een netwerk diagram gemaakt van de mogelijke oplossing.



Figuur 10 netwerk diagram mogelijke oplossing

5. Project aanpak

Aan de hand van een gekozen ontwikkelmethode wordt uitgelegd hoe de opdracht uitgevoerd zal worden en hoe het project aangepakt is.

5.1. Ontwikkelmethode

Voor dit project is een aangepaste vorm van Scrum (nadere uitleg onder 5.2) gekozen. Scrum is een agile ontwikkeltechniek waarmee producten iteratief ontwikkeld kunnen worden. Bij iedere iteratie moet een deelproduct opgeleverd worden. Bij Scrum bestaan verschillende rollen: een product owner, een scrum master en development teamleden. De product owner is de opdrachtgever. De rest van de taken wordt uitgevoerd door mij als afstudeerder.

Scrum is een handige en veelgebruikte methode binnen softwareontwikkeling omdat het iteratief werkt en aan het eind van iedere sprint(iteratie) een product oplevert. Dit product moet werken en dus ook getest zijn. Bij scrum wordt de product owner nauw bij het project betrokken. Dit is erg handig bij dit project omdat de cross-platform applicatie de gebruiksvriendelijkheid moet verbeteren. Er zullen beslissingen genomen moeten worden tussen de beveiliging en de gebruikersvriendelijkheid. Direct contact met de opdrachtgever is daarbij essentieel.

Doordat er voor scrum gekozen is, kunnen tijdens het project aanpassingen worden gedaan aan de initiële opzet. De product back log kan na iedere sprint wijzigen zonder daar veel voor te hoeven doen.

5.2. Scrum

Scrum is een ontwikkelmethode waarin normaal gesproken meerdere personen het development team vormen. Omdat deze opdracht alleen uitgevoerd wordt, zal een aantal veranderingen gemaakt moeten worden aan de wijze van werken. Wel zal de product owner als enig ander teamlid bij het team horen.

De daily standup zal niet worden gehouden omdat dit met één persoon niet nodig is.
Wel zal er iedere dag de volgende drie vragen worden beantwoord:

- Wat is er gisteren gedaan?
- Wat wordt vandaag gedaan?
- Zijn er zaken waardoor de sprint in gevaarloopt?

Deze daily standup zal opgeschreven worden en iedere dag wordt deze naar de opdrachtgever gestuurd. De opdrachtgever heeft dit aangevraagd omdat hij graag betrokken wil blijven. Bij scrum mag de opdrachtgever bij iedere daily standup zijn om meer betrokken bij het project te zijn maar ook om eerder problemen te ontdekken. In de praktijk gebeurt het niet vaak dat de product owner aanwezig is bij de daily scrum omdat de opdrachtgever vaak bij een ander bedrijf werkt of het team op een andere locatie werkt. De daily scrum is wel echt bedoeld ter ondersteuning van het team bij het realiseren van het commitment en niet het op de hoogte houden van de opdrachtgever.

5.3. Sprints

Er zullen vijf sprints worden gehouden die omringd worden door een opstartfase en een afrondfase. In de opstartfase wordt het plan van aanpak en de initiële product backlog gemaakt. Deze product backlog zal samen met de opdrachtgever/product owner gemaakt worden. De product backlog zal tijdens het project aangevuld worden.

Aan het begin van iedere sprint zal er een definition of done geschreven worden. In de definition of done wordt bepaald wanneer het product af is. Na iedere sprint zal er gekeken worden door de product owner of er aan de definition of done gehouden is en of de sprint dus goed is afgerond.

Iedere sprint duurt drie weken omdat er rekening gehouden moet worden om het proces zo goed mogelijk te documenteren zodat dit verslag zo goed mogelijk het proces beschrijft. Daarnaast zijn er meerdere systeemdelen, de webservice en cross-platform applicatie die wijzigingen moeten ondergaan.

Na iedere sprint volgt een sprint review. In deze review zal samen met de product owner gekeken worden naar het product dat opgeleverd is.

Het allerlaatste item in een sprint zal een retrospective zijn. Een retrospective is een meeting waarin meestal het volgende besproken wordt:

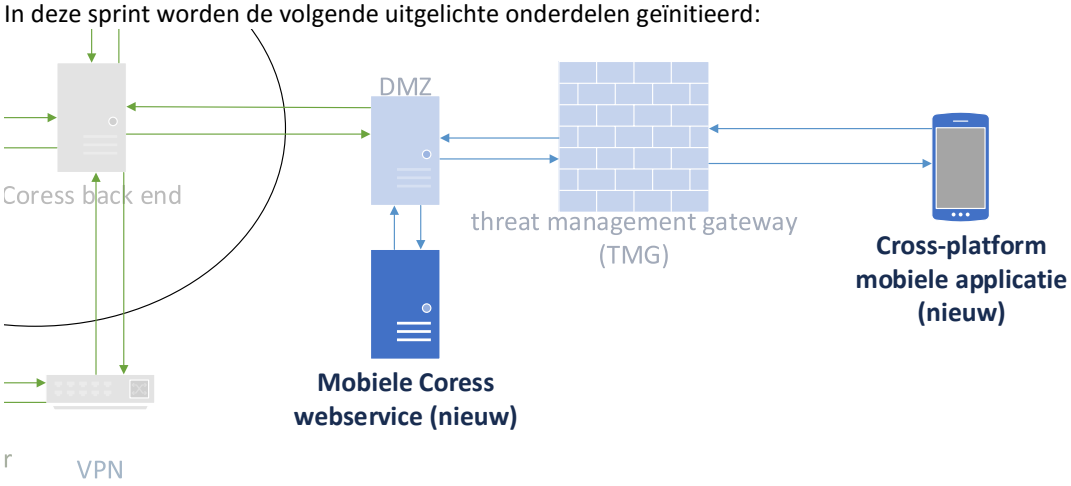
1. Wat de volgende sprint niet meer hoeft te gebeuren
2. Wat de volgende sprint wel moet gebeuren
3. Wat moet het team blijven doen in de volgende sprint

6. Sprint 1 skelet bouwen

De eerste sprint van dit project stond in het teken van het leren van Xamarin.
Er is een aantal fundamentele keuzes gemaakt waaronder het gebruik van Soap of REST en welke versie van Xamarin het beste voor dit project is.
Deze sprint is voldaan zodra er aan de volgende eisen is voldaan:

- Een basis van de mobiele applicatie op basis van Xamarin is gemaakt en werkt op alle drie de platformen
- De mobiele webservice kan uren uit de interne webservice halen en deze beschikbaar stellen voor de mobiele applicatie
- De mobiele applicatie kan de uren ophalen uit de webservice
- Er zijn unit testen geschreven die de functionaliteit test. Deze unit tests moeten slagen

In deze sprint worden de volgende uitgelichte onderdelen geïnitieerd:



Figuur 11 netwerk diagram mogelijke oplossing

De definition of done is in deze sprint behaald. De uren worden uit de interen Coress webservice gehaald door de mobiele webservice en de mobiele webservice geeft de uren door aan de mobiele applicatie.

6.1. Planning

Op de planning van sprint 1 staat ten eerste dat kennis over Xamarin wordt opgedaan. Tijdens deze periode zal een keuze gemaakt moeten worden tussen de twee soorten van Xamarin. Ook moeten de applicaties opgezet worden. Dit houdt in dat de basis van de applicaties moeten worden gebouwd. Tijdens dit proces moet een keuze gemaakt worden met welke techniek de webservice gemaakt moet worden. Alle lagen moeten opgezet worden omdat de user story die tijdens deze sprint uitgevoerd wordt over de gehele applicatie gaat. De user story die uitgevoerd wordt is:
“*Als gebruiker wil ik dat mijn uren die ik via de computer invul ook in mijn app komen*”.
Deze user story houdt in dat de uren, van de gebruiker, die in de Coress database staan ook in de applicatie in te zien zijn. De uren zijn ingevuld via de browser van een pc. Deze user story is gekozen omdat deze bij het opzetten van de applicaties de makkelijkste manier user story is.

6.2. Uitvoering

6.2.1. SOAP of REST

Omdat de applicatie een webservice nodig heeft om te kunnen communiceren met de Coress database zal deze gemaakt moeten worden. Er zijn twee mogelijke oplossingen voor de webservice: een Simple Object Access Protocol (SOAP) webservice of een Representational state transfer (REST) webservice.

De communicatie tussen de Coress back-end webservice en de services die daar gebruik van maken loopt via SOAP. De reden voor SOAP is een aantal jaar geleden gemaakt toen de huidige situatie ontworpen is. Voor de nieuwe situatie is er voor gekozen om opnieuw een keuze te maken tussen SOAP en REST.

SOAP kan ook via andere protocollen communiceren dan alleen het HTTP-protocol. Vaak wordt gekozen om het HTTP-protocol te gebruiken omdat hiermee eenvoudig door firewalls en proxies heen kan gegaan worden zonder het SOAP protocol zelf aan te passen.

REST is van recentere datum en wordt meer gebruikt dan SOAP. REST is geen protocol zoals SOAP maar een architectuur stijl. REST maakt namelijk gebruik de basisprincipes van HTTP. GET, POST, PUT, DELETE zijn de standaard HTTP methodes. SOAP is een vast protocol waaraan gehouden moet worden. Er kan niet vanaf geweken worden. REST kan op meerdere manieren geïmplementeerd worden terwijl dat bij SOAP niet kan.

Waarom zou SOAP gebruikt worden voor de webservice?

- SOAP maakt gebruik van een WSDL. Een WSDL is een contract hoe gecommuniceerd dient te worden. Er kan op geen andere manier met de webservice gecommuniceerd worden. Hierdoor is de manier van communiceren duidelijk voor de gebruiker.
- SOAP is niet stateless dus iedere request kan een vervolg zijn op een eerdere request.
- SOAP regelt een aantal zaken voor de developer zoals: transacties, veiligheid, coördinatie, adressering, vertrouwen

Waarom zou REST gebruikt worden voor de webservice?

- REST is gangbaarder en wordt bijna door alle mobiele applicaties gebruikt.
- REST gebruikt het bestaande http-protocol dus hoeft in de cross-platform applicatie geen extra SOAP libraries te gebruiken of eigen implementaties te schrijven
- Er kan zelf bepaald worden hoe de veiligheid en vertrouwen wordt geregeld. Er is meer controle over wijze waarop het geïmplementeerd is, zelf of via een library.
- De kennis over REST is al aanwezig.

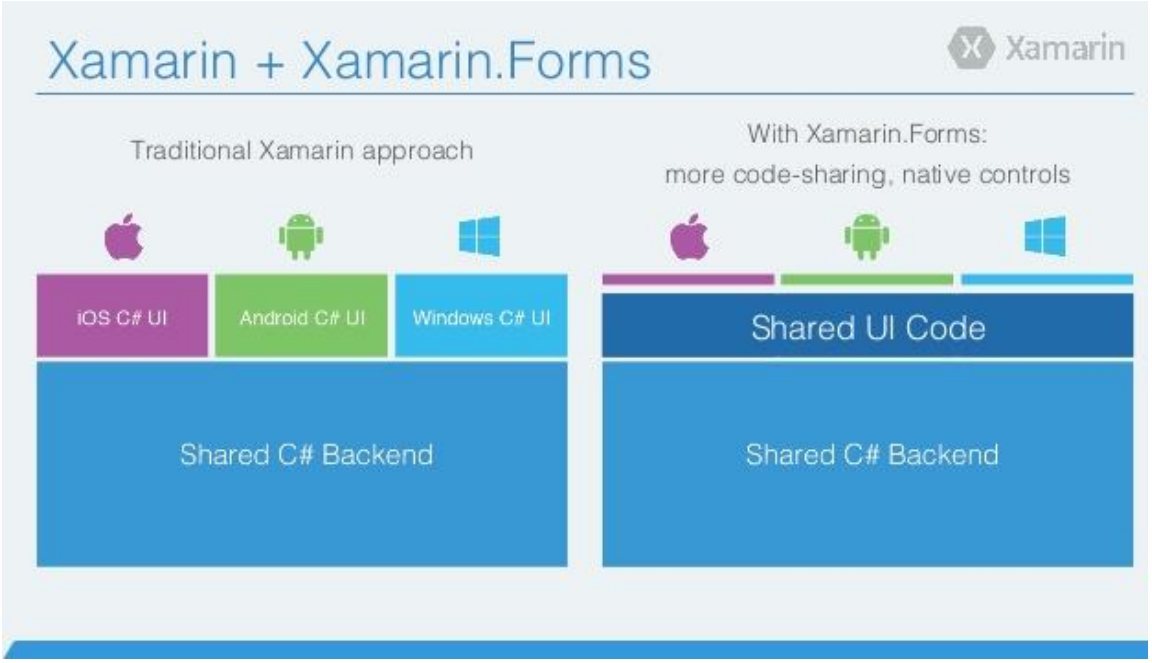
Omdat de webservice alleen communicatie met de cross-platform applicatie maakt, is een WSDL overbodig. Er zijn geen externe partijen die weten hoe ze moeten communiceren met de webservice. Daarnaast hoeft er geen extra kennis overgebracht te worden als er gebruik wordt gemaakt van REST. Als laatste en meest doorslaggevende reden is de gebruikelijke manier van communicatie tussen webservices en mobiele applicaties. Dus is er gekozen voor een REST webservice. Als er in de toekomst ook andere applicaties gebruik willen maken van de webservice zullen deze altijd intern zijn. Intern zal het functioneel ontwerp en technisch ontwerp beschikbaar zijn waardoor er nog steeds geen WDSL nodig zal zijn.
Omdat de X-platform applicatie in C# geschreven wordt, is voor ASP.NET Web api 2 gekozen.

6.2.2. Xamarin

Het .Net framework is ontwikkeld door Microsoft. C# code wordt door een Just in Time compiler van het Common Language Runtime (CLR) omgezet naar machine code wat tijdens het uitvoeren van de applicatie gebeurt. Deze CLR heeft Microsoft alleen voor Windows gemaakt. En daarom kan C# .NET officieel alleen op Windows worden uitgevoerd. Het opensource project Mono vervangt de Just in Time compiler van de Microsoft CLR. Mono zorgt ervoor dat C# .Net applicaties ook op Linux, Mac of andere platformen uitgevoerd worden. Xamarin is de eigenaar van Mono. De laatste ontwikkelingen van Microsoft zijn dat de code van het .Net library open-source gemaakt zijn waardoor mono nu gemakkelijker onderhouden kan worden.

Xamarin heeft meerdere producten die op Mono zijn gebaseerd. MonoTouch en Mono for Android. MonoTouch zorgt ervoor dat de code naar iOS gecompileerd wordt en Mono for Android zorgt voor de juiste code op Android. In tegenstelling tot Mono zijn MonoTouch en Mono for Android geen open source.

Voor dit project moeten cross-platform mobiele applicaties voor IOS, Android en Windows Phone gemaakt worden, Xamarin zorgt ervoor dat de applicaties niet allemaal in hun eigen taal geschreven moeten worden maar dat ze in C# geschreven kunnen worden. Xamarin zorgt dus dat de applicaties native draaien op de smartphones en is daarom voor dit project een goede keus.



Figuur 12 Xamarin + Xamarin.Form vergelijking

Xamarin heeft twee verschillende manieren om de applicaties te schrijven. De originele methode Xamarin platform of Xamarin.Forms. Het verschil van Xamarin platform en Xamarin.Forms is dat de UI bij platform per applicatie apart geschreven moet worden en bij Forms voor iedere applicatie hetzelfde is. Forms heeft een voordeel omdat er minder code geschreven hoeft te worden. En bij Platform is het voordeel dat de guidelines van ieder platform nagegaan kunnen worden. Forms is bedoeld voor het maken van formulieren en lijstjes zoals de naam al aangeeft.

De Coress mobiele applicatie zal geschreven worden in Xamarin.Forms. De keuze is gevallen op Xamarin.Forms omdat het uiterlijk voor deze applicatie niet uitmaakt. Het gaat om de functionaliteit. De mobiele applicatie wordt geen commercieel product waardoor Info Support meer klanten kan krijgen. Het moet simpel en snel zijn om te gebruiken. Daarnaast moet het voor alle drie de devices er precies hetzelfde uitzien, er hoeven geen platform specifieke handelingen uitgevoerd worden.

6.2.3. Implementatie

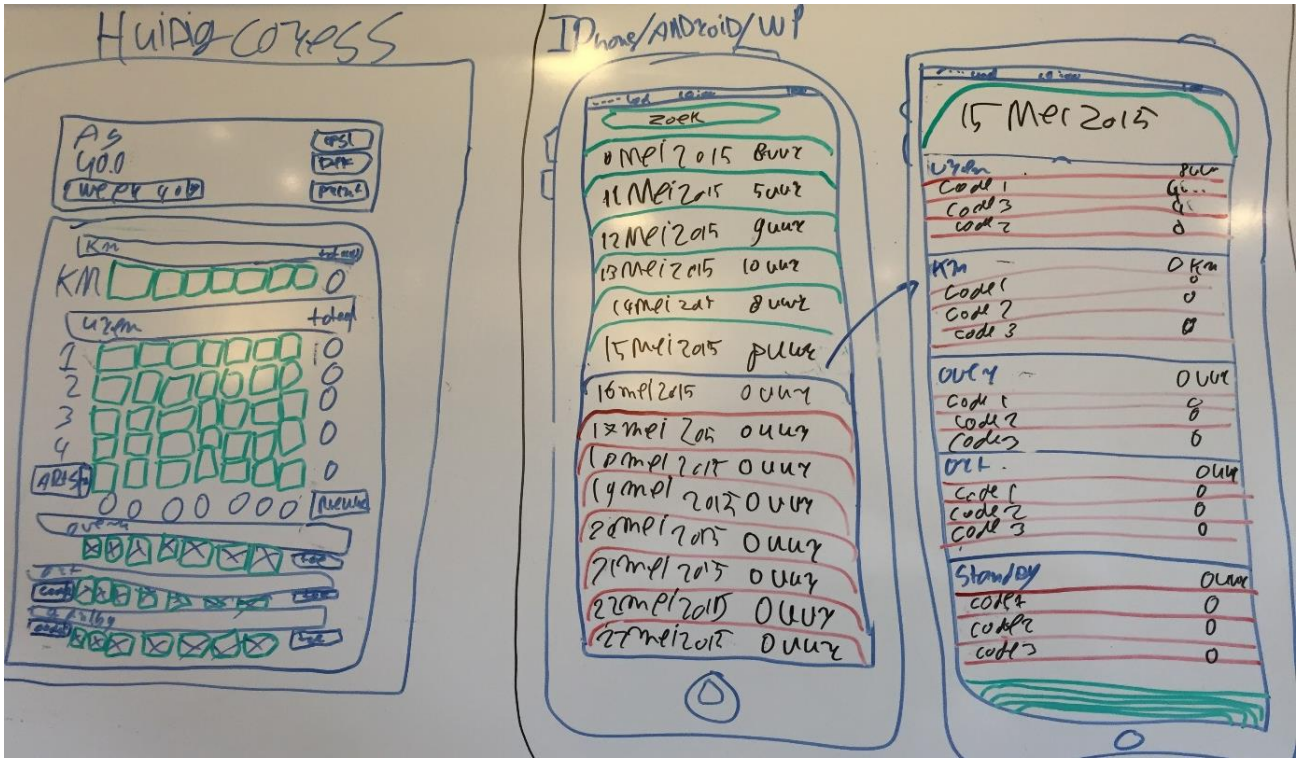
Tijdens de implementatie zijn twee projecten opgezet. De webservice en de mobiele cross-platform applicatie. Het versiebeheer van beide applicaties wordt via Microsoft Team Foundation Server (TFS) gedaan. TFS is een handig product omdat hierin ook een scrum template zit waarin al het werk op een grafische manier bijgehouden kan worden. Alle begeleiders vanuit Info Support en ikzelf kunnen hierdoor de voortgang goed bewaken. Alle requirements (user stories) worden hierin ook opgeslagen.

6.2.4. Webservice

De webservice is gemaakt met behulp van de template Web Api 2 uit Visual studio. Hier zitten alle functies die een REST webservice nodig heeft. In de webservice is een connectie gemaakt met de interne webservice van Coress. Dit is een SOAP-verbinding. Deze SOAP-verbinding is zonder problemen opgezet. De SOAP-verbinding is gemaakt met een library die al in de interne web front-end gebruikt werd. Omdat de web front-end dezelfde functies heeft als de mobiele applicatie kon deze library gebruikt worden.

6.2.5. Xamarin

Het cross-platform project is met behulp van de Xamarin template opgezet. In deze template zitten alle losse projecten zoals die van IOS, Android en Windows Phone. Om de gedeelde code te schrijven is een Portable class library, hierna PCL, gegenereerd. Het project IOS heeft een build server nodig. Deze build server moet een Apple product zijn. Info Support heeft een iMac geregeld dat bij het IOS-project gebruikt kan worden.



Figuur 13 wireframe: huidige Coress + nieuwe

De schermen zien er anders uit dan de huidige webversie. Zoals te zien in Figuur 13 Wireframe: Huidige Coress + nieuwe. Hier worden de totale uren per dag in een lijst getoond. En als er op een dag geklikt worden de individuele uren opgehaald. De interne webservice van Coress geeft meteen alle uren van deze week terug. Omdat de mobiele

versie andere data, per request, nodig heeft moet een conversie van de data plaats vinden. De data conversie moet iedere keer plaatsvinden als data opgevraagd worden. De data conversie is daarom zo licht mogelijk gemaakt.

6.2.6. Tests

De verbinding om de data van de ingevulde uren uit te wisselen is met een REST verbinding aan beide kanten opgezet. Bij de unit tests verliep dit perfect. Dit komt omdat de tests geschreven zijn tegen de portable class library, PCL. De PCL heeft een goede implementatie van de httpclient klasse waarmee een REST server bevraagd kan worden. Helaas was dit niet het geval toen de applicatie op de devices getest werd. De implementatie bij Windows Phone was dezelfde als de PCL en dit ging goed. Helaas verliepen de implementatie van IOS en Android niet meteen goed. Zij hebben een speciale versie van de httpclient. Deze werd officieel door Xamarin ondersteund maar werden niet standaard mee geïnstalleerd. Dit is omdat ieder platform een andere implementatie heeft met het bevragen van een internet.

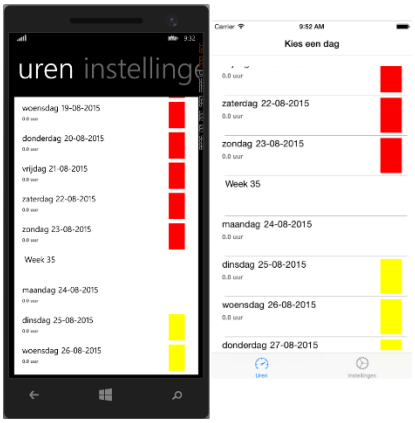
De emulator van Android is heel erg langzaam waardoor niet veel getest op Android is. IOS moet getest worden op de build server. Door deze stap extra is er ook niet veel getest op IOS. Windows phone heeft daarentegen een goede emulator waardoor veel getest is op Windows phone. Door deze verschillen is het een paar keer voorgekomen dat een onderdeel op een platform anders werkte dan een ander platform terwijl ervan uitgegaan is dat het onderdeel goed werkte. Hierdoor is afgesproken om bij de volgende sprint per onderdeel dat toegevoegd is op alle platformen een test uit te voeren.

6.2.7. UI

De UI is via code geschreven in de PCL. Deze wordt door middel van de Xamarin.Forms library geschreven. In deze library zitten alle UI-componenten waaronder buttons, labels en listviews. Deze componenten worden gecompileerd voor het juiste platform. Hierdoor zien bepaalde onderdelen van de apps op ieder platform niet identiek uit. Bijvoorbeeld een listview bij Windows phone ziet er anders uit dan van IOS. Terwijl het er in code er precies hetzelfde uitziet. In code ziet het er als volgt uit:

```
DayList = new ObservableCollection<DayViewModel>();
DayListView = new ListView
{
        ItemTemplate = new DataTemplate(typeof(DayViewCell)),
        ItemsSource = DayList,
};
```

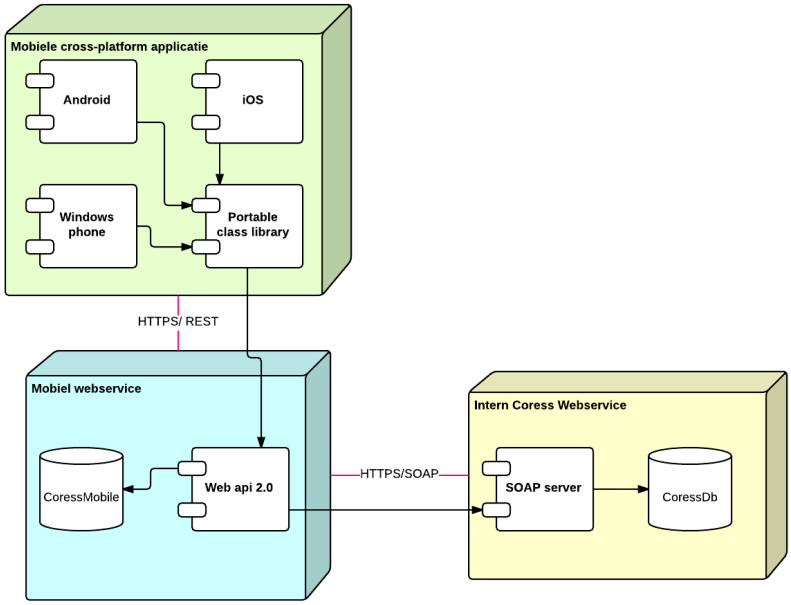
De DayViewCell zorgt ervoor dat de inhoud van de items er op een bepaalde manier uitziet. Hoe de lijst werkt en hoe deze getoond word mag het platform zelf weten. Daarom zijn er bij iOS streepjes tussen de items te zien.
In Figuur 14 WP + IOS is van de ui een voorbeeld te zien. Het grootste is de navigatie van de twee verschillende platformen. Ios heeft onderaan de navigatie in buttons en Windows Phone heeft bovenaan de navigatie in grote letters.



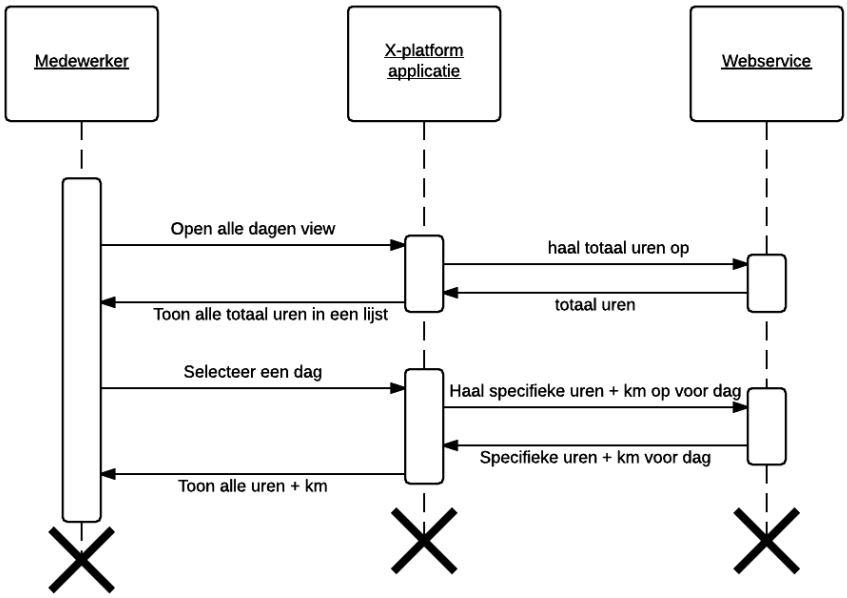
Figuur 14 WP + iOs

6.1. Ontwerp/Bouw

Na deze sprint zijn de twee projecten opgezet. Het architectuurontwerp zoals te zien in figuur 15 is tijdens het ontwerpen gemaakt waardoor meer inzicht is gegeven van de gekozen technieken en communicatiestromen. Daarnaast is in Bijlage D functioneel ontwerp de user story uitgewerkt. In figuur 16 Is de sequence diagram te zien die hier bij hoort.



Figuur 16 Architectuur model



Figuur 15 sequence diagram uren ophalen

7. Sprint 2 authenticatie kiezen

Sprint twee staat in het teken van authenticeren. In deze sprint wordt het authenticatie protocol gekozen en opgezet. Deze sprint is succesvol afgerond als er een authenticatie protocol is gekozen en dat er een implementatie voor geschreven is.

7.1. Planning

In de planning stonden twee user stories namelijk:

1. *"Als gebruiker wil ik kunnen inloggen omdat alleen ik dan mijn uren kan invullen en bekijken."*
2. *"Als gebruiker wil ik dat de verbinding over SSL loopt omdat ik dan weet dat er niemand kan afluisteren."*

In overleg met de opdrachtgever is er halverwege de sprint besloten om de tweede user story te verplaatsen naar de derde sprint. Het bepalen welk authenticatieprotocol het beste voor de applicatie is kostte meer tijd. De verplaatsing was mogelijk met de planning.

7.2. Uitvoering

Om een authenticatie protocol te kiezen kan dat op verschillende methodes. Eén van die methodes is de KPMG-pakketselectie methode. In deze methode worden vier verschillende fase gebruikt: de analysefase, longlist-fase, shortlist-fase en contract fase. In dit project wordt tijdens de analysefase een aantal verschillende methodes van authenticatie geanalyseerd met deze methodes wordt er een longlist opgesteld. Nadat er een keuze is gemaakt welke methodes door een zwaarwegende reden niet gebruikt kan worden zal deze afvallen. Met de overgebleven methodes zal er een shortlist op worden gesteld. Er zal in de shortlist-fase knock-out criteria opgesteld worden. Deze knock-out criteria zijn criteria waar de methode aan moet voldoen. Omdat niet iedere methode aan alle criteria voldoet of een beetje worden er bij iedere criteria punten gegeven aan de hand van een verdeling. Welke methode de meeste punten heeft zal uiteindelijk gebruikt worden. De contract fase zal hierdoor vervallen. De methode zal dan geïmplementeerd worden.

7.2.1. Analyse fase

Tijdens de analyse fase is er een literatuuronderzoek uitgevoerd. Er bestaan vele authenticatie mogelijkheden en veel van deze protocollen lijken op elkaar maar worden net iets anders uitgevoerd. Tijdens het onderzoek zijn een aantal protocollen gevonden die van elkaar verschillen. Deze zijn opgenomen in de longlist. De longlist is mede gebaseerd op de Wikipediapagina met verschillende protocollen [8].

7.2.2. Longlist

Door middel van onderzoek is er een longlist opgesteld, deze longlist ziet er als volgt uit:

Methode	Omschrijving
1. **Basic**	Basic protocol is een protocol waarmee wachtwoorden plain tekst naar de server worden gestuurd.
2. **Digest**	Digest is een uitbreiding van het basic protocol maar hierin worden de credentials gehasht
3. **SRP**	Deze methode zorgt ervoor dat het wachtwoord nooit over het internet wordt gestuurd. Door een algoritme zal het wachtwoord gecontroleerd worden.
4. **OAuth 2.0**	OAuth 2.0 maakt gebruik van een identity provider. Dat betekent dat een externe applicatie zorgt voor de authenticatie.
5. **Kerberos**	Kerberos (protocol vernoemd naar Griekse driekoppige hellehond) maakt gebruikt van een aparte authenticatieserver net zoals OAuth. Deze Kerberos zorgt voor autorisatie en authenticatie. Deze genereerd tickets om de communicatie tussen de client en service op te zetten
6. **RFID, Fingerprint**	Een RFID-chip of de vingerafdruk van de gebruiker authentiseert en autoriseert de gebruiker.

1.1.1 Afgevallen methodes

Kerberos is sterk afhankelijk van tijd. Bij het verifiëren van de identiteit van client of server wordt gebruik gemaakt van de lokale tijd. Deze tijd mag niet verschillen van de ontvangende kant. Bij een mobiele applicatie kan het voorkomen dat de mobiele telefoon op een langzaam internet zit waardoor de tijd te veel verschilt. Kerberos wordt ook bijna altijd in een lokaal netwerk gebruikt. Dezelfde tijd kan niet verzekerd worden, waardoor Kerberos al meteen afvalt.

De volgende methode die afvalt is de RFID-reader en fingerprintscanner omdat niet iedere telefoon een RFID-reader of fingerprintscanner heeft. Wel zouden deze methodes bovenop de later gekozen methode toegevoegd kunnen worden. Dit zorgt voor een extra laag beveiliging.

Hierdoor is er voor gekozen om OAuth, SRP en Basic en Digest mee te nemen naar de Shortlist.

7.2.3. Werking

Basic

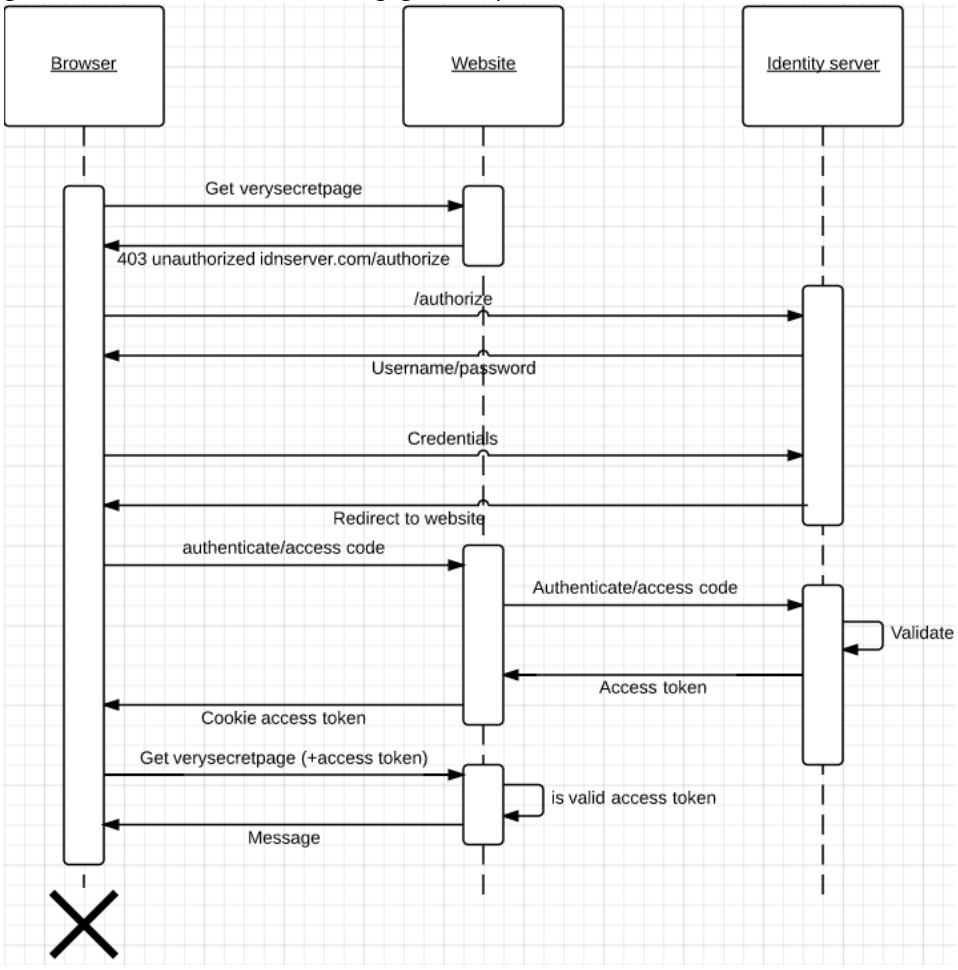
Basic, basis, gebruikt geen keys of tokens waardoor de username en wachtwoord plain tekst met de header meegestuurd worden. De server zal deze gegevens controleren of ze bekend zijn bij het systeem.

Digest

Digest is een uitbreiding op basic authentication. Het wachtwoord wordt niet plain tekst verstuurd maar het wordt gehashed. De ontvangende partij moet ook het wachtwoord hashen om te kijken of de gestuurde hash hetzelfde is. Hierdoor kan de aanvaller wel de gehashde versie onderscheppen maar dan heeft hij misschien nog niet het juiste wachtwoord. En zou hij niet het wachtwoord plain tekst krijgen zonder een dictionary attack of iets dergelijks uit te voeren.

OAuth 2.0

OAuth 2.0 wordt gebruikt door Google, Facebook, Twitter en nog een aantal van deze grote internetbedrijven. Het is ontstaan doordat een website (ma.gnolia) graag twitter gebruikers zijn website wilde laten gebruiken en daarbij gegevens over die gebruiker wilde ophalen. Er moest dus een manier komen om de gebruikers via twitter in te laten loggen en dat ma.gnolia via de geauthentiseerde gebruiker via twitter gegevens op halen. Hierdoor is Oauth geboren. In 2006 kwam OAuth 2.0 op de markt via onderstaande sequence diagram is te zien hoe een gebruiker geauthentiseerd wordt en daarna gegevens ophaalt.

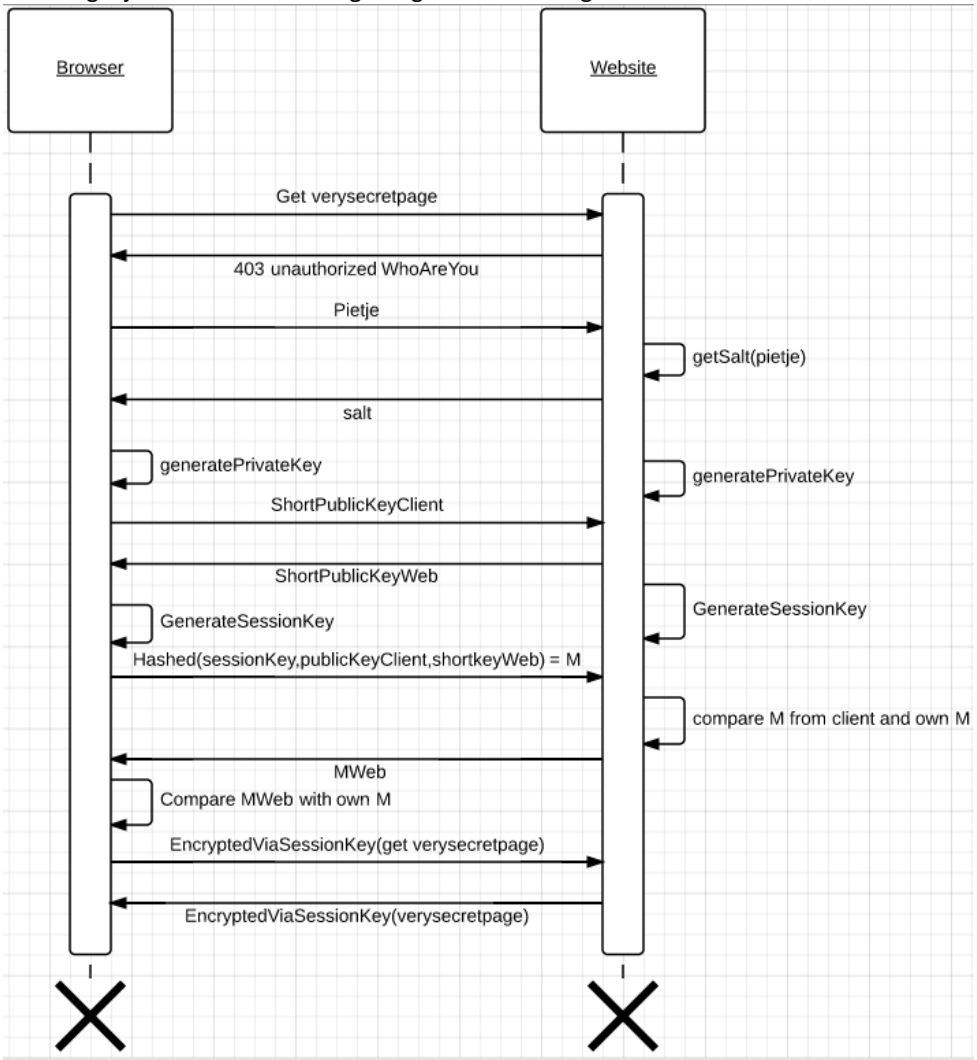


Figuur 17 sequence diagram OAuth 2.0

Zodra de browser een beveiligde pagina opvraagt terwijl hij nog niet geauthentiseerd wordt de browser door gestuurd naar een identity server. Op de identity server logt de gebruiker in, meestal gebeurt dit via Basic authentication, en zodra de gebruiker geauthentiseerd is wordt de gebruiker weer terug gestuurd naar de website met een access code. Met deze access code kan de website een access token opvragen aan de identity server, hierdoor kan de identity server de website controleren of hij wel een access token mag ontvangen. De website slaat een cookie met de access token op bij de browser en met die cookie kan de browser zijn toekomstige aanvragen doen bij de website. Bij iedere aanvraag wordt de cookie gecontroleerd. In dit geval wordt er maar één keer de username en password over het internet verstuurd en niet bij iedere aanvraag zoals bij basic authentication gebeurd.

SRP

SRP is een protocol wat op augmented password-authenticated key agreement (APAKA) gebouwd is. APAKA werkt met public en private keys om elkaars identiteit te vertrouwen. En beide hebben een shared-secret, doormiddel van deze shared secret en public keys kan een session key worden berekend. Hierdoor is een man-in-the-middle attack niet mogelijk. Onderstaande diagram geeft de werking van SRP weer.



Figuur 18 sequence diagram SRP

Zodra een website een aanvraag doet bij de website probeert deze eerst te decrypten zodra dat niet kan of de identiteit kan niet geverifieerd worden zal er een melding geven worden dat er geautoriseerd moet worden.
De browser zal de website de username sturen. De website haalt de salt op van de gebruiker en stuurt deze terug naar de browser. De browser zal een longtime private key genereren. Daarna zal hij een korte publieke key generen

en naar de website sturen. De browser krijgt van de website zijn korte publieke key. De browser kan een session key creëren. De server kan met de ontvangen gegevens en eigen gegevens ook een session key berekenen. Zodra de session keys berekend zijn kan de verbinding in principe geencrypt worden. De server en browser weten op dat moment niet of zij dezelfde session key hebben waardoor zij een validatie procedure starten. Deze validatie procedure begint bij de client die naar de server een gehashde waarde van onder andere de session key stuurt. De server controleert deze waarde en creëert een andere hash met andere gegevens die de client dan ook moet controleren. Als alle beide dezelfde waardes hebben dan weten de server en browser van elkaar dat zij de juiste session key hebben. De verdere communicatie volgt via symmetrische encryptie met de session key.
Een uitbereide uitleg over SRP is te vinden in bijlage A – SRP.

7.2.4. Knock-out criteria

Om te bepalen welke methode uiteindelijk gebruikt gaat worden moeten knock-out criteria worden opgesteld. De knock-out criteria zullen gebaseerd zijn op de eisen van de cross-platform applicatie.
Er wordt naar de volgende criteria gekeken:

1. Er wordt goed omgegaan met wachtwoorden (of andere gevoelige informatie)
2. Makkelijk te implementeren
3. Well-known (wordt veel gebruikt)
4. Veel documentatie

Er zal bij iedere methode gekeken worden of deze aan de criteria voldoet. Het kan voor komen dat er gedeeltelijk wordt voldaan. Daarom is er voor gekozen om punten te verdelen. Met het onderstaande schema zullen de punten worden verdeeld:

	Aantal punten
Voldoet volledig aan de criteria	3 punten
Voldoet bijna aan de criteria	2 punten
Voldoet gedeeltelijk aan de criteria	1 punten
Voldoet niet aan de criteria	0 punten

Er is een criteria die belangrijker zijn dan andere criteria, namelijk:

1. Er wordt goed omgegaan met wachtwoorden (of andere gevoelige informatie)

Deze criteria is het belangrijkst. Bij deze criteria worden de punten verdubbeld. Deze verdeling ziet er als volgt uit:

	Aantal punten
Voldoet volledig aan de criteria	6 punten
Voldoet bijna aan de criteria	3 punten
Voldoet gedeeltelijk aan de criteria	2 punten
Voldoet niet aan de criteria	0 punten

Om de punten juist te verdelen is er per criteria duidelijk gemaakt wat er verstaan wordt onder de punten verdeling. Dit is als volgt:

Er wordt goed omgegaan met wachtwoorden (of andere gevoelige informatie)

Gebruik van credentials wordt mee bedoelt hoeveel gevoelige informatie over de gebruiker naar de server wordt gestuurd:

- 6 punten: er wordt geen gevoelige data verstuurd
- 3 punten: er wordt alleen bij de initiële aanvraag credentials gestuurd (gehashed)
- 1 punten: er wordt wel gevoelige data verstuurd maar is gehashed
- 0 punten: er wordt wel gevoelige data verstuurd en is plain tekst.

Makkelijk te implementeren

Bij makkelijk te implementeren wordt bedoeld of er veel gebouwd moet worden of dat er maar een één of meer klassen bij gebouwd moeten worden.

- 3 punten: er hoeft niks bij geschreven worden of een aantal klassen.
- 0 punten: er moet één of meerdere projecten toegevoegd worden.

Well-known standard (goed getest/veel gebruikt)

Well-known standard wordt mee bedoelt dat het door bekende bedrijven gebruikt wordt. Een groot bedrijf is een bedrijf die meer dan 1.000.000 gebruikers heeft. Een klein bedrijf heeft minder dan 1.000.000 gebruikers

1. 3 punten: veel grote bedrijven maken er gebruik van
2. 2 punten: Een aantal grote bedrijven maken er gebruik van
3. 1 punt: een aantal kleine bedrijven maken er gebruik van
4. 0 punten: niemand maakt er gebruik van

Veel documentatie

Als er veel documentatie beschikbaar is kunnen problemen sneller opgelost worden.

5. 3 punten: er is documentatie beschikbaar.
6. 1 punt: er is documentatie beschikbaar maar is beperkt.
7. 0 punten: er is geen documentatie beschikbaar.

7.2.5. Uitslag

In de tabel hieronder zijn alle punten toegewezen.

	Gebruik credentials	Makkelijk te implementeren	Well-known	Veel documentatie	Total
OAuth	3	0	3	3	**9**
SRP	6	3	2	1	**13**
Basic	0	3	1	3	**7**
Digest	1	3	1	3	**8**

Een aantal punten vallen op dus worden deze nader toegelicht:

Oauth makkelijk te implementeren:

Oauth heeft een aparte Identity server nodig om de identiteit te valideren en controleren.
Dit is het orspronkelijke idee van OAuth tegenwoordig bestaat er een versie waarbij de server ook de identity server is. Omdat er dan afgeweken wordt van het oorspronkelijke protocol is er toch gekozen voor 0 punten. Deze punten beïnvloeden niet de uiteindelijke uitslag.

Oauth well-known:

Oauth wordt door veel grote bedrijven gebruikt zoals Google, Twitter en Facebook. Waardoor er vanuit kan gaan dat dit protocol door en door getest is.

SRP-gebruik credentials:

SRP stuurt nooit het wachtwoord over naar de andere partij. Door bepaalde algoritmes en vooraf gedefinieerde verifiers te gebruiken kan er bepaald worden of het ingevoerde wachtwoord goed is.

SRP makkelijk te implementeren:

De theorie van SRP is complex maar de implementatie kan al in een aantal klassen gemaakt worden. Hierdoor heeft SRP 3 punten gekregen.

SRP well-known:

SRP wordt niet door de grote partijen zoals Google, Twitter en Facebook gebruikt. Maar in het artikel van mount knowledge[10] wordt er van uitgegaan dat de ING mobiele applicatie SRP gebruikt wordt. Helaas mocht de main

developer van de mobiele tak van ING hier niks overzeggen. Hij is in dienst bij Info Support. ING[9] geeft aan 2.000.000 gebruikers te hebben.

SRP veel documentatie:
Bij SRP is nauwelijks documentatie omdat het niet door veel ontwikkelaars wordt geïmplementeerd.

Basic gebruik credentials:
Basic authentication stuurt geen gehashde wachtwoorden maar deze worden plain tekst verstuurd.

Conclusie
Als conclusie is er gekomen dat SRP als beste kandidaat uit de test komt. SRP zal dus ook gebruikt worden als authenticatie protocol.

7.2.6. Gebruik SRP

Op internet zijn een aantal implementaties te vinden van SRP. Zelf heb ik de Bouncy castle library gevonden. Bouncy castle is een cryptografie library waar veel cryptografie onderdelen in zitten. Verschillende soorten encryptie algoritmes maar ook een SRP-implementatie. Om toch de wiskundige formules te begrijpen heb ik eerst zelf een SRP test project gemaakt. In dit project heb ik de handshake nagebouwd hoe het in de paper van Tom Wu[9] staat. Na een dag programmeren kwam ik er niet meer uit. Priemgetallen/modulo/generator waren op dat moment nog te cryptisch. Ik besloot om de bouncy castle implementatie naast de paper te leggen en bekijken of de juiste wiskundige formules gebruikt werden. Na de versie van bouncy castle begreep ik ook veel meer van alle formules en hoe hiermee om moet worden gegaan in C#. Het is lastige materie maar na de bouncy castle implementatie bekeken te hebben is het duidelijk geworden hoe het werkt.
Om SRP te verduidelijken staat in de bijlage SRP een uitleg met code voorbeelden.

De implementatie van bouncy castle zijn eigenlijk alleen de formules. Zelf moet de uitvoering van de juiste formules ontworpen worden. Ook het versturen van data van de client naar de server en terug moet zelf geïmplementeerd worden. Er is een wrapper gemaakt om de implementatie te maken. De handshake wat door de client geïnitieerd wordt gaat als volgt:

```
var result = await client.GetAsync(string.Format("api/auth/{0}/Salt/", identity));
```
De salt wordt opgehaald bij de server via de httplcient.

```
if (result.StatusCode == HttpStatusCode.Unauthorized && result.ReasonPhrase ==
Coress.Mobile.AppModelContract.Constants.TooManyLogin)
        return new KeyValuePair<bool,string>(false,"Te veel verkeerde pogingen");
```
Als de status code terug gegeven wordt in plaats van Ok dan zal er een fout melding weergegeven worden. Voor de overzichtelijkheid wordt hier maar één foutmelding weergegeven. Er zijn meerdere custom foutmeldingen.

```
var saltString = await result.Content.ReadAsStringAsync();
Salt = Encoding.UTF8.GetBytes(JsonConvert.DeserializeObject<string>(saltString));
```
Het resultaat staat in een json formaat deze zal eerst omgezet moeten worden naar een string

```
var AJson = JsonConvert.SerializeObject(GetA().ToString());
```
Om de A naar de Server te sturen moet deze in json formaat omgezet worden
```
result = await client.PostAsync(string.Format("api/auth/{0}/B/", identity),
        new StringContent(AJson, Encoding.UTF8, "application/json"));
```
Er wordt gevraagd om de B en stuurt de A mee.

```
var BJsonConvert = JsonConvert.DeserializeObject<string>(await
        result.Content.ReadAsStringAsync());
```
de B wordt weer terugezet in een string.

```
B = new BigInteger(BJsonConvert);
SetS(B);
```
Er wordt een bigInteger van gemaakt en daarna wordt de session key berekend

```
result = await client.PostAsync(string.Format("api/auth/{0}/validateM/", identity),
new StringContent(JsonConvert.SerializeObject(GetM()), Encoding.UTF8,"application/json"));

if(result.StatusCode == HttpStatusCode.Forbidden)
        return new KeyValuePair<bool, string>(false, "Pincode onjuist");
```
Als de M niet goed is wordt de status code fobidden terug gestuurd. Hierdoor is het duidelijk dat de pincode niet juist is.

```
var m2ServerJson =JsonConvert.DeserializeObject<string>(await     result.Content.ReadAsStringAsync());
```
Als hij wel goed is dan heeft de server de M2 terug gegeven.

```
var m2Server = new BigInteger(m2ServerJson);
if (!CheckM(m2Server))
        return new KeyValuePair<bool, string>(false, "Pincode onjuist");
IsAuthenticated = true;
```

7.3. Bijzonderheden

Op de helft van de sprint zijn de bevindingen van het authenticatie protocol gepresenteerd aan systeembeheer. Systeembeheer reageerde hier niet positief op. Zij kende het protocol niet en gebruikte liever het alternatief Basic authentication maar dan over SSL. Basic authentication kwam uit de test als het slechtste protocol. Omdat het over SSL gaat en dit vaker wordt gebruikt vindt systeembeheer dit een veiligere optie. Na overleg met de opdrachtgever hebben wij besloten om toch wel SRP te gebruiken om de volgende redenen:

1. Het is veiliger dan Basic. Zodra SSL wegvalt door een of andere reden ligt bij basic direct je gebruikersnaam en wachtwoord bloot. SRP stuurt nooit een password over de verbinding.
2. Bij basic authentication moet je je email en wachtwoord steeds opnieuw intypen. Bij SRP kan je gebruik maken van een simpele pincode, wat nog steeds veilig genoeg is.

Om ervoor te zorgen dat de applicatie wel een ander authenticatie protocol kan gebruiken, is besloten om een module te maken waarmee berichten verstuurd kunnen worden. Hierdoor is het makkelijker om van protocol te wisselen indien systeembeheer, na uitleg van het protocol, niet akkoord gaat met de SRP-implementatie.

Tijdens het implementeren van SRP in de applicatie was er weer een probleem met de httpclient. De httpclient werkte bij het Windows Phone project wel maar niet bij Android en iOs. Tijdens het debuggen verdween de thread helemaal en er kon geen foutmelding gevonden worden. Er waren veel andere gebruikers van Xamarin.forms die hetzelfde probleem hadden. De oplossing bleek achteraf simpel want de httpclient in het project van SRP was gebaseerd op de httpclient op een hele andere httpclient dan gebruikt werkt in de andere projecten. Windows Phone, iOS en Android hebben allen een andere manier om een http request te sturen. Dit heeft te maken met verschillende hardware die het OS moet aanspreken. Xamarin heeft voor iOS en Android een implementatie gemaakt in C#. Omdat niet iedere applicatie een http request hoeft te maken is dit niet standaard ingeschakeld. Door de httpclient dezelfde te maken als die van de PCL was het probleem opgelost. Tijdens het probleem is er getwijfeld om toch over te stappen naar SOAP maar omdat er over SOAP veel klachten op de forums van Xamarin te vinden waren was deze twijfel maar van korte duur.

8. Sprint 3 authenticatie toevoegen

Sprint drie staat in het teken van het uitwerken van SRP en het initieel registreren van gebruikers bij de webservice. Deze sprint is af als gebruikers in kunnen loggen bij de cross-platform applicatie met hun vooraf gedefinieerde pincode en als de verbinding over SSL loopt.

8.1. Planning

De volgende user strories staan op de planning voor sprint drie:

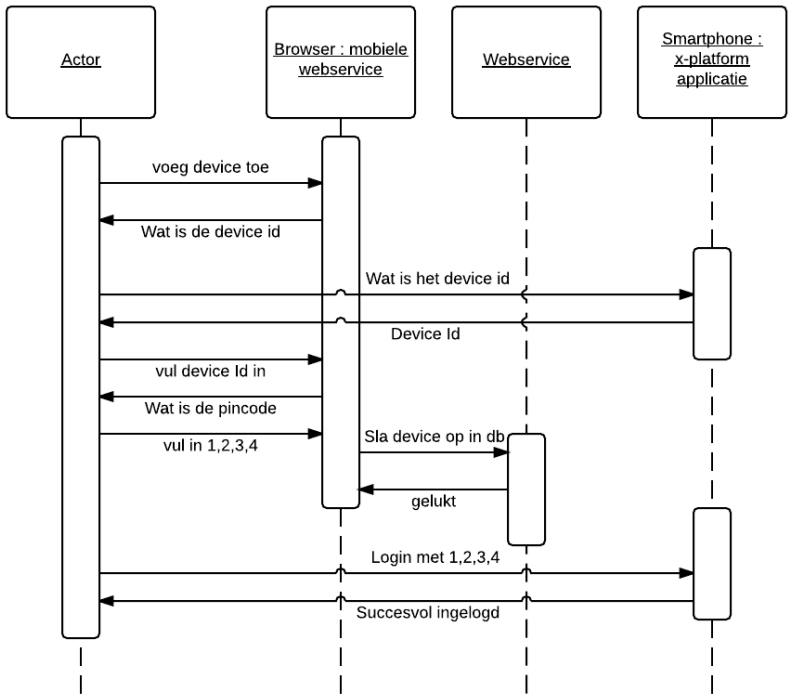
1. *"Als gebruiker wil ik mijn device aanmelden en ook af kunnen melden via de webservice omdat ik mijn device wil koppelen aan mijn Coress account"*
2. *"Als gebruiker wil ik een pincode invullen om in te loggen omdat ik dat makkelijker vind dat mijn email en wachtwoord van Coress"*
3. *"Als gebruiker wil ik dat de verbinding over SSL loopt omdat ik dan weet dat er niemand kan afluisteren."*

Ook is er uit de sprint review gekomen dat er visueel iets moest veranderen aan de applicatie. In het inlog scherm was een tekst met Info Support te zien. De opdrachtgever wilde hier liever een afbeelding van Info Support. Dit zal in deze sprint ook worden verwerkt.

8.2. Uitvoering

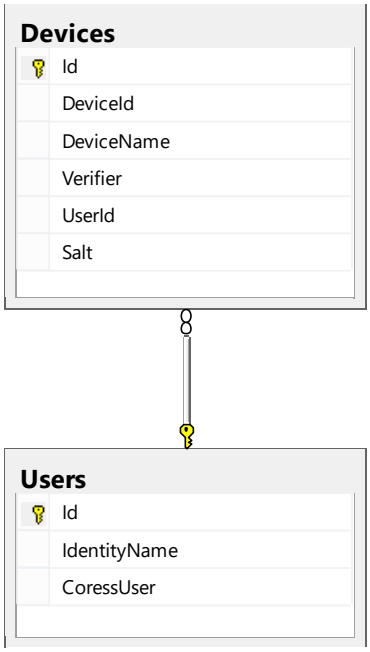
8.2.1. Initiële registratie

Omdat SRP geen wachtwoord verstuurt kan er geen authenticatie uitgevoerd worden via Windows authentication. De interne Coress webservice verwacht dat de gebruiker met Windows authentication is ingelogd. Om functies aan te roepen bij de interne Coress webservice verwacht de interne Coress webservice een Coress medewerker. Deze medewerker kan opgevraagd worden door de methode WhoAmI(). De WhoAmI methode vraagt dan aan de Windows authentication wie er ingelogd is. Totdat de interne Coress webservice een andere methode heeft om een Coress medewerker op te halen moet er een alternatief worden bedacht. Het Coress medewerker object wat terug gegeven wordt door de WhoAmI methode kan opgeslagen worden in de database van de mobiele webservice. En iedere keer dat er een functie aangeroepen wordt, kan dit object dan worden gebruikt. Dit is een tijdelijke oplossing totdat Coress zelf is aangepast. Het Coress medewerker object heeft namelijk een timestamp en een concurrency key. Deze concurrency key zorgt voor goede concurrency. Een nieuwe methode om een Coress medewerker terug te geven valt buiten de scope van dit project.



Figuur 19 sequence diagram device toevoegen

Om een device te kunnen gebruiken moet het device eerst worden toegevoegd aan de webservice. Via bovenstaande sequence diagram kan dit worden gedaan.



Figuur 20 Database model (export van Sql management studio)

De mobiele webservice heeft een database nodig. In deze database worden gegevens over de devices en de medewerkers bewaard. Er moet zo min mogelijk beschikbaar zijn omdat deze database in de DMZ zal staan. De DMZ is open voor al het internetverkeer en wordt als onbetrouwbaar gezien.

Een user heeft een CoressUser. Deze CoressUser is de tijdelijke WhoAmI object. De user heeft ook een naam. Hiermee kan de medewerker controleren of hij met het juiste account is ingelogd.

Verder kan de User meerdere Devices hebben. Deze devices hebben allemaal een DeviceId dit is het Id wat bij de telefoon hoort. Deze kan via het loginscherm opgevraagd worden. De devicename is de naam die getoond wordt in het overzicht van alle devices. Hier kan de gebruiker zijn device aan herkennen als hij deze wilt verwijderen. De UserId is de foreign key naar de User. De verifier is de verifier die in het SRP-protocol wordt gebruikt dit is een combinatie van Salt, deviceId, N en g.

Iedere medewerker moet vooraf dat hij gebruik gaat maken van de applicatie op zijn Mobile device zijn device aanmelden bij de webservice. Om ervoor te zorgen dat de juiste Coress medewerker in de database gezet wordt moet er via het interne netwerk op de webservice worden ingelogd. Ieder device heeft zijn eigen pincode omdat de

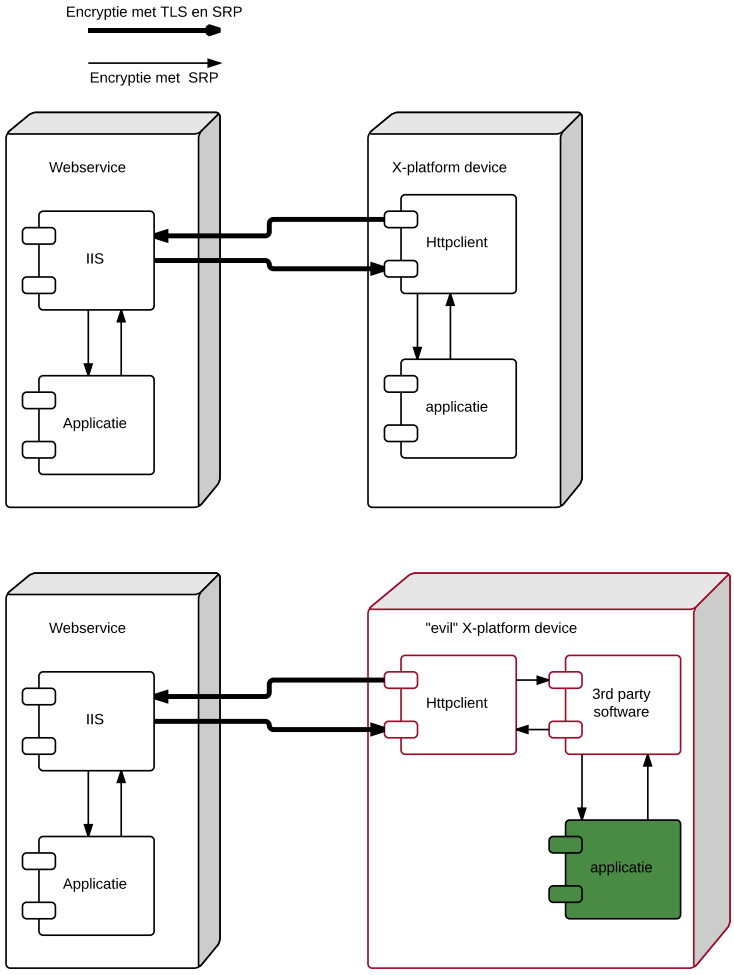
verifier een combinatie is van de pincode en device id. De verifier wordt dus ook berekend zodra het device wordt aangemaakt.

8.2.2. SSL/TLS

De applicatie maakt gebruik van SSL. SSL staat voor secure sockets layer. Deze layer zorgt ervoor dat de verbinding tussen de httpclient in de native mobiele applicatie tot en met IIS, waar de webservice gehost wordt, genencrypt wordt. SSL wordt tegenwoordig veel gebruikt om aan te geven dat de verbinding beveiligd is. SSL is sinds een aantal jaar onveilig door ouderwetse encryptie technieken. TLS (Transport Layer Security) is de opvolger van SSL en deze wordt tegenwoordig gebruikt. Er bestaan een aantal versies van TLS die ook onveilig zijn. De huidige en veilige versie is op het moment van schrijven TLS 1.2.

IIS heeft SSL/TLS ingebouwd en dit is simpel in te schakelen voor de webservice.
Bij de native httpclient moet de aanvragen naar de https website gaan. Er hoeft niet specifiek aangegeven worden dat het om een SSL request gaat.

Er bestaat een extensie van het SSL/TLS met SRP maar het is niet gelukt om deze te implementeren omdat er geen implementatie voor .Net is. Dus is er gekozen voor twee lagen beveiliging. Ook is het veiliger om niet 100% te vertrouwen op één protocol.



Figuur 21 SSL/SRP security

In figuur 21 model is een situatie beschreven waarbij de x-platform device in slechte handen is gevallen. Op het moment dat er alleen op SSL/TLS vertrouwd wordt kan kwaadaardige software de verbinding onderscheppen waardoor de data bloot ligt. Omdat er in deze applicatie een extra encryptie laag is toegevoegd zal de hacker alleen maar geencrypte data en de device identifier tegenkomen. Deze device identifier is alleen encrypt met SSL/TLS omdat de webservice deze nodig heeft om de juiste key op te halen die nodig is om de data te decrypten.

8.2.3. Encrypten/decrypten

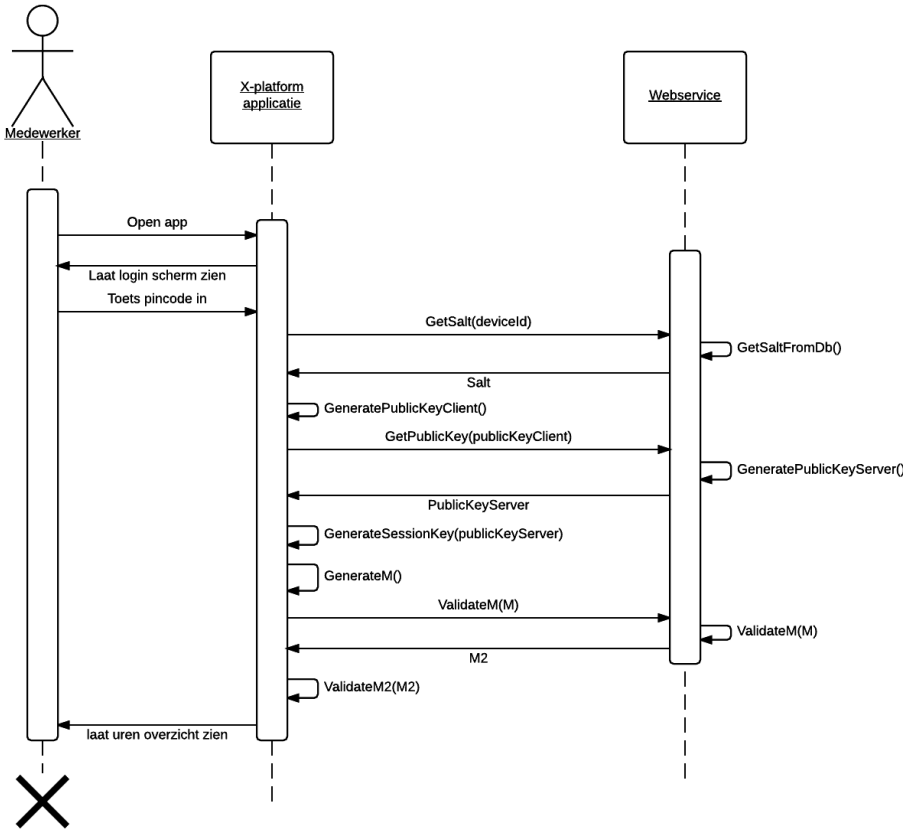
Als tweede security laag is er gekozen voor AES (Advanced encryption standard) encryptie in combinatie met een GCM (Galois/counter mode) algoritme. AES is tegenwoordig één van de betere encryptie standaarden. GCM is een manier om de geencrypte data van vertrouwelijkheid en authenticiteit te voorzien. Dit zorgt ervoor dat de aanvaller de tekst niet kan aanpassen en dus corrupt kan maken.
Met de session key als wachtwoord wordt de body die verstuurd wordt geencrypt. Dit zorgt ervoor dat de body zelfs voordat het bij de webhost IIS aankomt geencrypt is en nadat de body bij de httpclient in de applicatie is. Pas in de applicatie laag zal er een decryptie plaats vinden. Met het bouncy castle library worden de berichten geencrypt en gedecrypt.

8.3. Ontwerp/Bouw

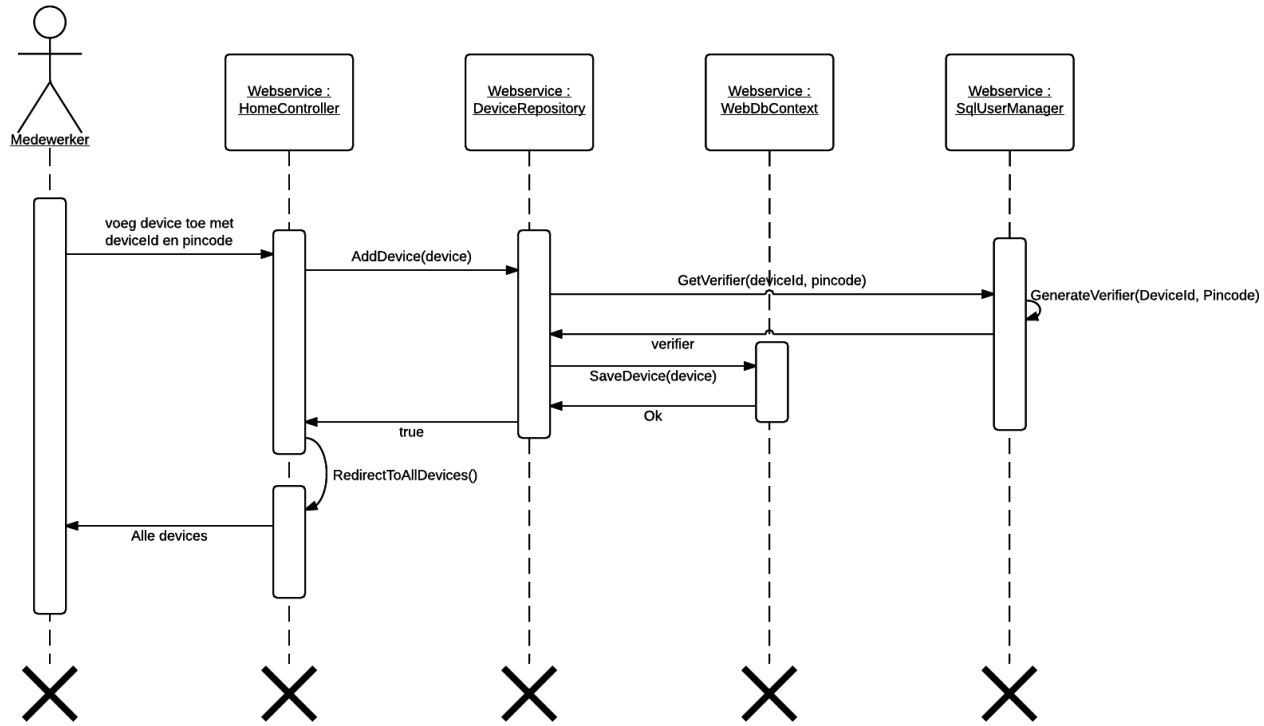
In deze sprint zijn de volgende user stories uitgewerkt in Bijlage D – Functioneel ontwerp:

1. *“Als gebruiker wil ik mijn device aanmelden en ook af kunnen melden via de webservice omdat ik mijn device wil koppelen aan mijn Coress account”*
2. *“Als gebruiker wil ik een pincode invullen om in te loggen omdat ik dat makkelijker vind dat mijn email en wachtwoord van Coress”*

In figuur 22 is de technische sequence diagram voor het inloggen te zien. En in Figuur 23 is de technische sequence diagram te zien voor het toevoegen van een device.

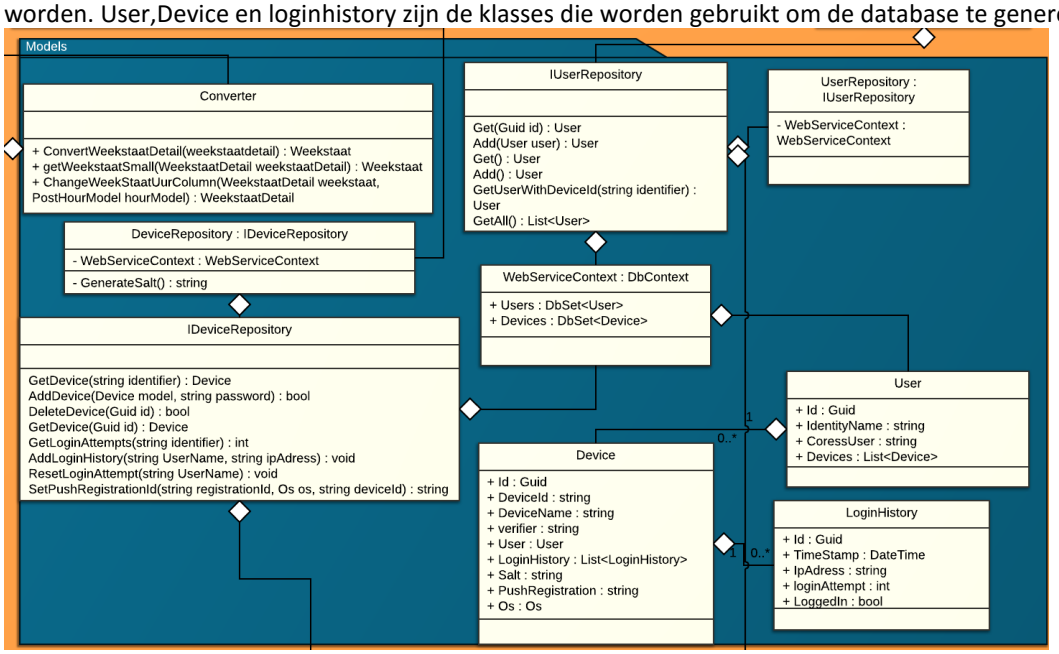


Figuur 22 Sequence diagram inloggen



Figuur 23 sequence diagram device toevoegen

Om de database aan te spreken is er gebruik gemaakt van de ORM mapper Entity Framework. Er is voor deze ORM mapper gekozen omdat er zeer gemakkelijk in een web api 2.0 project vanuit klasses (via code-first) automatisch een database gebouwd wordt. Om niet overal los in het project objecten uit de database te halen is er voor gekozen om het repository pattern te gebruiken. Het repository pattern houdt in dat een klasse de database bevragingen afhandelt. In Figuur 24 is een klasse diagram te zien van het repository pattern. De WebserviceContext is de main klasse waarmee het entity framework wordt ingesteld en waarmee objecten uit de database bevraagt kunnen worden. User,Device en loginhistory zijn de klasses die worden gebruikt om de database te genereren.



Figuur 24 klasse diagram

9. Sprint 4 functionaliteit bouwen

Deze sprint staat in het teken van uren invullen. De applicatie kon tot heden niet meer dan inloggen en uren bekijken. Dat verandert in deze sprint. Deze sprint is succesvol afgerond als er uren ingevuld kunnen worden maar ook als er een login historie bijgehouden wordt. Als laatste moet de applicatie een foutmelding geven als er vaker dan tien keer een foutieve pincode wordt ingevoerd en dus niet meer mag proberen.

9.1. Planning

In deze sprint zijn de volgende user stories behandeld:

1. *"Als gebruiker wil ik uren invullen a.d.h.v. uur codes omdat ik graag mijn uren gemakkelijk wil invullen"*
2. *"Als gebruiker wil ik uren pas definities maken als ik dat expliciet aangeef omdat ik mijn uren soms tussentijds wil toevoegen"*
3. *"Als gebruiker wil ik niet kunnen inloggen als ik 10 keer een foutief wachtwoord heb ingevuld omdat ik niet wil dat een ongeautoriseerde persoon mijn account kan brute forcen"*
4. *"Als gebruiker wil ik mijn login geschiedenis zien omdat ik dan weet of ik gehackt ben omdat ik dan controle heb over mijn account."*
5. *"Als gebruiker wil ik dat mijn uren opgeslagen worden zodra ik ze invul zodat ik niet een extra handeling hoef uit te voeren om mijn ingevoerde uren op te slaan in de interne Coress"*
6. *"Als gebruiker wil ik mijn uur code een alias geven zodat makkelijker uren toegevoegd kunnen worden"*

In de sprint is er nog een user story bij gekomen namelijk:

7. *"Als een week definitief gemaakt wordt en er wordt niet aan de business logica voldaan moet de error van de interne Coress gegeven worden omdat ik wil weten wat er mis is gegaan. "*

In de sprint review van sprint 1 was er nog een todo item naar voren gekomen wat in deze sprint pas verwerkt was namelijk dat de gewerkte uren prominenter aanwezig moesten zijn dan de andere uren. De gewerkte uren worden namelijk het meest gebruikt.

9.2. Uitvoering

In deze sprint is de meeste functionaliteit toegevoegd. De belangrijkste functionaliteit is het invullen van uren, user story 1. Dit invullen gebeurd aan de hand van een uur code, dag en aantal uur. De interne Coress webservice verwacht een hele week van gewijzigde informatie. Omdat de mobiele applicatie op dag niveau werkt moet er een conversie van de data worden toegevoegd. Deze conversie wordt op de volgende manier uitgevoerd.
De mobiele applicatie stuurt de volgende informatie naar de mobiele webservice:

- Soort weekstaat
- Weekstaat uur Column
    - Code
    - Is diverse
    - Value
- Datum

De interne Coress verwacht de volgende relevante informatie:

- Week
  - Maandag
  - Weeknummer
  - Jaar
- Status
- Lijst van GewerkteUren
  - Code
  - Is diverse
  - Maandag
  - Dinsdag
  - Etc
- Lijst van OverUren
  - Code
  - Is diverse
  - Maandag
  - Dinsdag
  - Etc
- Lijsten van andere uren

Om de juiste uren toe te voegen wordt er eerst naar de juiste soort lijst van uren gekeken. Uit deze lijst wordt de juiste uurcode gehaald en dan wordt er bij de juiste dag de waarde toegekend. Als de code een diverse code is die nog niet bestaat, dan wordt deze eerst toegevoegd. Een projectcode moet bestaan in een weekstaatdetail omdat dezelfde weekstaatdetail opgehaald wordt om de juiste uren en projectcodes in de applicatie te tonen.
Iedere keer als er een uur toegevoegd wordt worden deze uren direct opgeslagen via de interne webservice.

9.2.1. Definitief maken

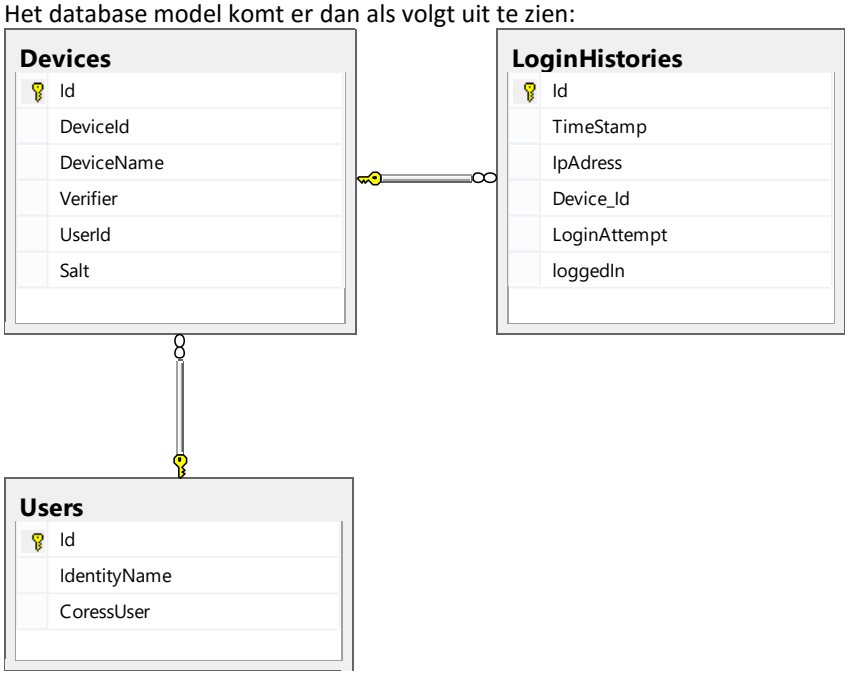
Als er een weekstaat definitief gemaakt, user story 2 wordt kan alleen de hele week definitief gemaakt worden. Omdat de applicatie bijna met alle functies per dag werkt is er naar een mogelijkheid gezocht om het definitief maken ook per dag te laten lopen. Helaas kan er in de interne Coress niet tussentijds definitief gemaakt worden. Een mogelijke oplossing is om in de webservice bij te houden welke dagen definitief zijn en als alle dagen van de week definitief zijn dit doorgeven aan de interne webservice. Omdat het definitief maken van een weekstaat aan constraints hangt moeten deze constraints dan ook in de mobiele webservice gebouwd worden. Dit zorgt voor veel extra werk en foutgevoeligheid wat niet opweegt tegenover de functionaliteit.
Deze constraints worden door de interne webservice door gegeven als foutmelding. Deze foutmelding wordt door gestuurd naar de mobiele applicatie waardoor de medewerker weet waarom de weekstaat niet geaccepteerd wordt.

9.2.2. Lock-out mechanisme

Om te voorkomen dat een onbevoegde persoon de vier cijferige pincode kan raden in een korte tijd is ervoor gekozen om een lock-out mechanisme in te bouwen, user story 3. Als het device probeert in te loggen zal dit opgeslagen worden in de database van de webservice. Zodra er een foutieve code is ingevuld wordt dit ook geregistreerd door de webservice. Als er 10 keer een foutieve pincode ingevuld is wordt de toegang voor 20 min geblokkeerd. Na 20 min kan het device weer proberen in te loggen. Een vier cijferige pincode heeft 10.000 mogelijke combinaties. Als al deze combinaties uitgeprobeerd worden is er 14 dagen nodig, dan worden de 20 minuten bijgerekend.

Omdat de login pogingen toch al bijgehouden worden wordt het ip adres en datum en tijd van de login ook bijgehouden, user story 4. Hierdoor kan de medewerker altijd kijken of er een andere persoon toegang heeft tot zijn

applicatie. Zodra dit het geval is kan de medeweker zijn device uit de mobiele webservice verwijderen. Hierdoor kan er met het device id niet meer ingelogd worden.
Het database model komt er dan als volgt uit te zien:



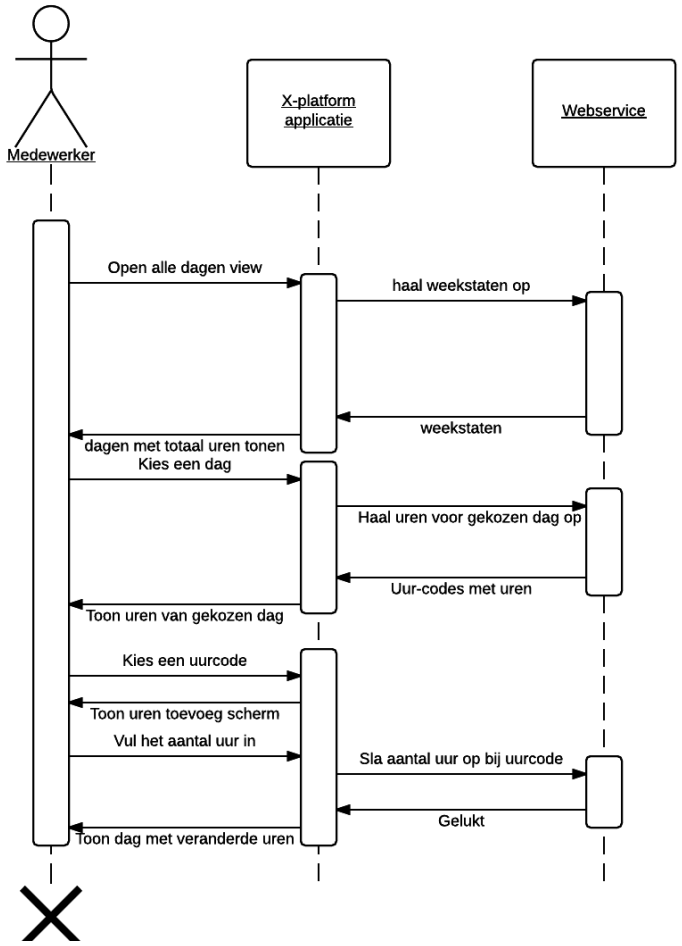
Figuur 25 Database model met Loginhistory

9.2.3. Aliassen

Een van de grootste ergernissen van de medewerkers is dat de projectcodes korte cryptische codes zijn. Bijvoorbeeld: INS-REC-SPR/ND of INS-INT-FG/ND. Om deze ergernis weg te halen kunnen de medewerkers in de applicatie aliassen instellen, user story 6. Een alias zal de projectcode in de hele applicatie vervangen. Dus als een medewerker een uur wilt invullen zal hij overal de alias zien. Alleen bij de allerlaatste stap bij het invullen zelf zal de projectcode zelf worden getoond. Deze keuze is gemaakt omdat de medewerker dan zeker weet of hij wel de juiste code heeft. Als hij bij het invullen van de aliassen een fout heeft gemaakt kan hij dat in de laatste stap iedere keer dubbelchecken.

9.3. Ontwerp/Bouw

In deze sprint is er veel door gebouwd op de al gekozen methodes. Het repository pattern is uitgebreid met de huidge user stories. In figuur 26 is het functioneel sequence diagram voor het invullen van de uren te zien.



Figuur 26 sequence diagram invullen uren

10. Sprint 5 push, uitloggen en pentesten

Deze sprint bestond voornamelijk uit het penetratie testen(pentest) van de applicatie. Daarnaast zijn er drie user stories behandeld. Deze sprint is met succes afgerond als de applicatie kan uitloggen, er een push notificatie gestuurd wordt als de weekstaat nog niet definitief is gemaakt en als een medewerker niet een weekstaat hoeft in te vullen ook geen device kan aanmelden. Daarnaast moet de pentest uitgevoerd zijn, een algehele pentest hoeft niet uitgevoerd te worden.

10.1. Planning

In sprint vijf zijn de volgende user stories behandeld:

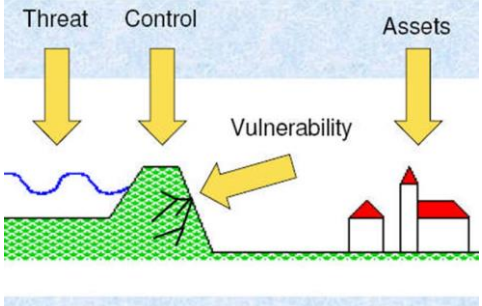
1. *"Als gebruiker wil ik uitloggen zodra ik klaar ben omdat ik dan zeker weet als ik mijn applicatie afsluit dat er geen ongeautoriseerde de applicatie kunnen gebruiken."*
2. *"Als gebruiker wil ik een push notificatie krijgen als ik vergeten ben mijn uren in te vullen omdat ik dan niet kan vergeten om mijn uren in te vullen"*
3. *"Als gebruiker wil ik dat als ik niet de weekstaat in hoef te vullen dit ook niet kan doen omdat ik dan niet voor niks de applicatie kan gebruiken"*

Ook is er in deze sprint een pentest uitgevoerd. Dit is niet een hele pentest geweest omdat dat veel tijd kost. Tijdens deze sprint zijn de belangrijkste testen uitgevoerd.

10.2. Pentest

Testen is een belangrijk onderdeel van softwareontwikkeling. Dit project heeft een grote focus op de beveiliging, met name authenticatie. Om een web applicatie goed te testen kan er een pentest uitgevoerd worden. Een pentest is een manier om de gehele applicatie te testen.

Een pentester kijkt in de ogen van een hacker en probeert net zoals een hacker binnen te komen om belangrijke gegevens te stelen. Om een pentest uit te voeren moet de pentester weten wat de belangrijkste onderdelen zijn waar de hacker mogelijk naar opzoek is, deze onderdelen, ook wel assets genoemd, kunnen data van de applicatie zijn maar ook bestanden die op hetzelfde netwerk zitten als de applicatie die slachtoffer is. Om bij deze assets te komen moet een hacker vulnerabilities vinden. Deze vulnerabilities zijn plekken in het systeem waar de hacker binnen kan komen wat niet zou mogen. In figuur 27 Threat, vulnerability, Assets is dit visueel weergegeven.



Figuur 27 Threat, vulnerability, Assets

Een vulnerability kan worden opgelost door een measurement. Een voorbeeld van een measurement tegen SQL injection is een ORM mapper gebruiken. Een vulnerability hoeft niet per se in de code te zitten maar kan ook in de serverconfiguratie zitten, als er bijvoorbeeld poorten openstaan die niet gecontroleerd worden. Daarom moet er op alle onderdelen waar de applicatie ook mee te maken heeft getest worden. Dit kost veel tijd en deze tijd is er niet tijdens dit project. Hierdoor is er gekozen om een test risico analyse uit te voeren. Welke onderdelen hebben het meeste risico om aangevallen te worden. In deze test risico analyse zijn de volgende vulnerabilities geanalyseerd:

- Authentication
- Authorization
- Session hijacking
- Injection
- Jailbreak
- Local storage

Door de impact en waarschijnlijkheid te berekenen is op de volgende score uitgekomen:

	Waarschijnlijkheid	Impact	Totaal	Rank
Authentication	9	8	72	1
Authorisation	7	10	70	2
Session hijacking	8	5	40	2
Injection	3	8	24	4
Jailbreak	9	2	18	5
Local storage	9	1	9	6

In “Bijlage B – Security tests” is de bovenstaande tabel uitgewerkt met de redenen voor deze punten.
Uit deze tabel blijkt dat authenticatie , autorisatie en session hijacking het belangrijkst zijn. De autorisatie tests volgens de OWASP testing guide v4[7] waren niet van belang bij dit project. En omdat de tests voor de autorisatie ook uitgevoerd kunnen worden tijdens de authenticatie is dit ook gedaan. Omdat er tijdens het project zo veel mogelijk measurements genomen zijn, zoals een ORM mapper gebruiken tegen SQL injection, zijn niet alle genomen measurements beschreven. Veel van deze measurements zijn voortgekomen uit andere beslissingen. Om al deze measurements en andere vulnerabilities te vinden moet er een algehele pentest uitgevoerd worden.

Terwijl ik de tests opgeschreven heb kwam ik erachter dat de applicatie veel bloot legde over de gebruikte methoden en het device id. Deze gegevens zouden geen lek opleveren maar een applicatie moet niet te veel blootleggen over de gebruikte methoden en andere gegevens als dit ook verborgen kan worden. Hierdoor is er voor gekozen om alle communicatie, behalve de authenticatie, van de applicatie naar de webservice via één end point te laten gaan. Dit zorgt ervoor dat er een geencrypt bericht gestuurd wordt naar de webservice met daarin de volgende gegevens:

- Methodenaam
- Json object met parameter waardes.

Om dit bericht te decrypten moet de juiste session key gebruikt worden deze wordt opgehaald met behulp van de device id. Dus wordt deze mee gestuurd samen met het encrypte bericht. Omdat er SSL plaatsvind zal deze ook genecrypt worden.

Er is één test waar de test faalde. De test OTG-SESS-003 moet testen of hetzelfde session id gebruikt kan worden voor iedere sessie. Omdat de applicatie gebruikt maakt van de device id om de sessie te bepalen was de test in eerste opzicht gefaald maar omdat met alleen de device id de sessie niet overgenomen kan worden was de test toch wel geslaagd. De berichten worden met behulp van de session key geencrypt dus deze moet in combinatie met de device id in handen zijn van de hacker om de sessie over te nemen. Tijdens het testen is er wel een vulnerability aan het licht gekomen. Na het uitloggen moet getest worden of er met dezelfde session id(device id) nog steeds de sessie voort kan zetten. Helaas bleek dit het geval dus is hier een measurement voor verzonnen. Deze vulnerability is opgelost door bij de session een laatste handeling datum bij te houden. Als er 30 min na de laatste handeling zit of er uitgelogd wordt moet er opnieuw worden ingelogd en wordt de sessie gedeactiveerd.

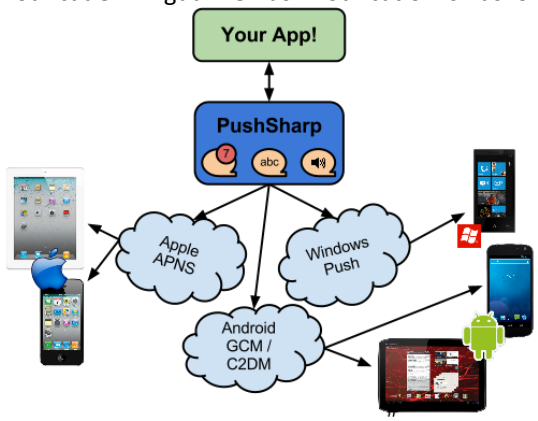
10.3. Push notificaties

Daarnaast is er bij de applicatie bebouwd dat de medewerker een push notificatie krijgt als hij vergeten is zijn uren in te vullen. Er zijn twee manieren om push notificaties te sturen:

1. Local push notificaties
2. Online push notificaties

Om lokaal push notificaties naar de gebruiker te sturen moet de applicatie aan zijn maar hoeft niet actief op de voorgrond te draaien. Deze applicatie stuurt dan vanuit de background een notificatie dat de medewerker zijn uren moet invullen. Het probleem bij deze methode is dat de applicatie alleen de gegevens heeft die hij voor het laatst heeft opgehaald. Dus als de medewerker op een andere manier wel zijn uren heeft ingevuld geeft de applicatie voor niks een push notificatie. De applicatie zal niet iedere dag gebruikt worden en daarom zal de applicatie niet altijd aan staan, zeker als de medewerker het vergeten is. Op dat moment kan de applicatie geen notificatie naar de medewerker sturen.

Om deze problemen op te lossen kan er gebruik gemaakt worden van de tweede keuze, een online push notificatie. Deze online push notificatie gebruikt een aantal componenten: De mobiele webservice die de notificatie stuurt, een push service van het platform en uit eindelijk de applicatie zelf die de notificatie ontvangt.
De push notificatie service is per platform verschillend Windows Phone gebruikt de Windows push Notification Services (WNS), iOS gebruikt de Apple Push Notification Services(APNS) en Android heeft de Google Cloud Messaging(GCM). Om niet al deze services zelf te implementeren is er gekozen om een implementatie te gebruiken. Een van de meest gebruikte implementatie is PushSharp. PushSharp zorgt voor de afhandeling van de notificatie. In figuur 28 Push notification is PushSharp visueel weer gegeven.



Figuur 28 Push notification

Omdat ieder platform anders werkt moet er ook op een andere manier PushSharp worden aangesproken.
Voor windows phone moet dat op de volgende manier:

```
push.RegisterWindowsPhoneService();
```
De service moet eerst geinitialiseerd worden.

```
push.QueueNotification(new WindowsPhoneToastNotification()
```
De notification moet in de queue gezet worden. Zodra de service klaar is kan de queue verwerkt worden.

```
        .ForEndpointUri(new Uri(ChannelUri))
```
Iedere applicatie heeft zijn eigen channelUri. Met deze uri kan de serivce de notificatie naar de juiste gebruiker sturen.

```
        .ForOSVersion(WindowsPhoneDeviceOSVersion.Eight)
```
Bij Windows phone hebben verschillende versies van het OS een andere implementaties om de notificatie te sturen. Dit wordt allemaal door PushSharp geregeld. Er moet wel aangegeven worden welke versie het is.

```
        .WithBatchingInterval(BatchingInterval.Immediate)
```
Wanneer de notificatie gestuurd op worden.

```
        .WithText1("Coress")
        .WithText2(Message));
```
De notificatie van Windows phone heeft twee regels met tekst.

iOS implementatie:

```
var appleCert = File.ReadAllBytes(Path.Combine(AppDomain.CurrentDomain.BaseDirectory,
         "MobileCoress.Cert.Development.p12"));
push.RegisterAppleService(new ApplePushChannelSettings(false,appleCert,
         "Test123Test123"));
```
iOS heeft een certificaat nodig met dit certificaat kan Apple controleren door wie de notificatie gestuurd is.

```
push.QueueNotification(new AppleNotification()
         .ForDeviceToken(deviceRegistrationId)
         .WithAlert(Message)
         .WithBadge(1));
```
Net zoals Windows Phone moet de notificatie in de queue gezet worden. De badge is het rode getalletje bij de applicatie pictogram.

Figuur 29 Badge

Android Implementatie:
```
push.RegisterGcmService(new
         GcmPushChannelSettings("AIzaSyBaYPlguEsV4wbFe_37Cv1Q1pd4SWA5lEE"));
```
Google kan de afzender controller met behulp van een API secret. Deze secret wordt hierboven geinitialiseerd.

```
push.QueueNotification(new GcmNotification()
         .ForDeviceRegistrationId(deviceRegistrationId)
         .WithJson("{\"alert\":\""+Message+"\",\"badge\":1,\"sound\":\"sound.caf\"}"));
```
Android verwacht een JSON-formaat om de notificatie te laten zien.

Als de notificatie naar de service gestuurd is zal deze de notificatie naar het device sturen. Hier hoeft niks voor gedaan worden. Als er op de notificatie geklikt wordt zal de applicatie meteen starten. Er is een mogelijkheid om de notificatie op te vangen zodra de applicatie start maar hier is niet voor gekozen.

Wat wel bij de applicatie moet gebeuren is het verkrijgen van de deviceToken, deviceRegistrationId of ChannelUrl. Dit gebeurt in de applicatie zelf. Op ieder platform gebeurt dit ook op een andere manier. Bij Windows Phone ging dit gemakkelijk er was een sample beschikbaar via de GitHub pagina van PushSharp. iOS en Android verliep dit niet geheel zonder problemen.

Bij iOS was er een probleem bij de certificaten. De simulator kan geen push notificaties ontvangen waardoor mijn eigen iPhone gebruikt moet worden om te testen. Via Apple moet de iPhone geregistreerd worden als developer device. Maar ook moet de applicatie geregistreerd worden met de mogelijkheid om push notifications te sturen. Helaas was het koppelen van de iPhone niet meteen gelukt. Dit is nog niet opgelost dus is het nog niet zeker of de implementatie klopt. De applicatie moet ook gedeeltelijk aangepast worden om de push notifications productie ready te maken.

Bij android was het probleem al bij de implementatie, de sample was erg onduidelijk waardoor deze niet gebruikt kon worden als voorbeeld. Een medewerker van Info Support heeft ook met PushSharp gewerkt waardoor deze een uitleg geven heeft over de Android implementatie.

11. Project afsluiting

11.1. HR-demo

Tijdens het afsluiten van de afstudeerperiode heb ik een demo gegeven aan de personen die functioneel betrokken zijn bij het project. Dit was aan de manager van Human Resource, de opdrachtgever, de procesbegeleider en de unitmanager van data solutions en mobile development. In deze demo heb ik de applicatie laten zien en de werking hiervan uitgelegd.

De manager van Human Resource heeft een verzoek ingediend dat alle ingevulde uren op een dag bij elkaar opgeteld worden en in het code overzicht ergens zichtbaar getoond moet worden. En de unitmanager heeft bovendien geëist dat in het dagenoverzicht een totaal aantal ingevulde uren per week getoond moet worden.

11.2. Systeembeheer demo

In de laatste fase is een technische demo gegeven. Deze demo is gegeven om de technische kant van de applicatie te tonen aan systeembeheer. In week 7 was gebleken dat systeembeheer niet direct SRP zal accepteren. Hierdoor is het aan mij de taak om de werking van SRP op papier te zetten en om aan systeembeheer te laten zien dat SRP gebruikt kan worden voor een mobiele applicatie zoals deze. In deze presentatie zal ik mijn werk tonen en uitleg geven over het protocol.

11.3. Advies

Tijdens de initiële registratie is een tijdelijke manier gevonden om de mobiele webservice zich voor te laten doen als een Coress medewerker. Dit is een tijdelijke oplossing en er wordt geen authenticatie uitgevoerd dat het ook echt de medewerker is. Om de mobiele webservice officieel te authentiseren als gebruiker moet hier een methode voor bedacht worden. Dit onderzoek zou te groot worden voor deze afstudeerperiode waardoor hier geen verder onderzoek naar gedaan is. Als de applicatie verder ontwikkeld zal worden moet hier nog een juiste en veilige methode voor bedacht moeten worden.

Daarnaast is weinig gedaan aan het uiterlijk van de applicatie. De UI van de applicatie is gebouwd zodat alles functioneel werkt. Een designer zou het uiterlijk mooier kunnen maken zodat de medewerker het niet erg vindt om de applicatie te starten.

Qua security is alleen rekening gehouden met de authenticatie en transport beveiliging. Er zou een volledige pentest uitgevoerd moeten worden om de andere aspecten van security ook zo veilig mogelijk te maken. Ook zal de huidige implementatie getest moeten worden door iemand anders dan ik. Ik heb het zelf ontworpen en gebouwd dus ik weet waar de lekken gedicht zijn. Door iemand anders te laten testen zal deze tests bedenken die ik niet had kunnen bedenken. Zelf heb ik wel de OWASP testing guide v4 gebruikt waardoor ik wel tests heb uitgevoerd die ik niet zelf verzonnen heb. Ook zou Coress zelf getest kunnen worden. Omdat Coress nu openbaar gemaakt wordt en op alle fronten aan beveiliging gedaan moet worden is het slim om voor Coress ook een pentest uit te voeren.

Een een aantal user stories is nog niet behandeld omdat deze pas later voort zijn gekomen uit de HR-demo.
De user stories zijn:

“Als gebruiker wil ik in het dagen overzicht per week het aantal uur van die week opgeteld worden omdat ik dan weet of al genoeg gewerkt hebt.”
“Als gebruiker wil ik in het dagen overzicht per week het aantal contracturen voor die week zodat ik deze met de totaaluren kan vergelijken.”
“Als gebruiker wil ik in het dag overzicht het totaal aantal uur van deze dag in de header omdat ik dan het overzicht niet kwijt raak als er veel uurcodes zijn.”

11.4. Procesevaluatie

Tijdens dit project is gebruik gemaakt van verschillende technieken om een zo goed mogelijk product op te leveren. In dit hoofdstuk zal ik reflecteren op de gemaakte keuzes zoals de keuze voor welke manier van projectmanagement is uitgevoerd wat door middel van een aangepaste versie van SCRUM is gedaan. De selectie van het juiste protocol is gedaan doormiddel van een aangepaste versie van de pakketselectie van KPMG. Daarnaast is het testen gebeurd met behulp van de OWASP testing guide v4 en ook met hulp van een security expert van Info Support. De manier van versiebeheer, door TFS, en de keuzes tijdens het ontwikkelen zal ik in dit hoofdstuk ook op reflecteren.

11.4.1. Scrum

De keuze voor SCRUM is een goede keuze geweest omdat ik goed mijn proces in de gaten kon houden. Met behulp van de scrum template werden er burndown charts gemaakt. Deze charts hield ik iedere dag in de gaten. Zodra ik merkte dat meer werk bleek te zijn dan vooraf bepaald, zoals in sprint 2, ben ik met mijn opdrachtgever gaan overleggen hoe ik dat probleem zou oplossen. Gelukkig is dit maar één keer voorgekomen en is de planning voor de rest goed verlopen. De keuze voor een drie-wekelijkse sprint is een goede beslissing geweest omdat tijdens het selecteren van het juiste protocol veel onderzoek gedaan moest worden. Het gekozen protocol was technisch zeer ingewikkeld waardoor veel tijd naar het uitwerken van SRP is gegaan. Omdat de sprint drie weken duurde kon ik in de sprints waar meer functionele user stories behandeld zijn goed indelen. Door de drie-wekelijkse sprint wist ik dat er genoeg tijd was om de proces documenten goed te beschrijven. Hier heb ik rekening mee gehouden.

Ik ben erg tevreden met de keuze voor SCRUM. Ik zal SCRUM vaker gaan gebruiken voor projecten, ook voor een eenmans project is SCRUM goed te gebruiken. Het geeft vrijheid maar toch veel zekerheid. De vrijheid omdat je iedere user story tussentijds kan verplaatsen. Maar het geeft je wel de zekerheid als je een user story verplaatst ook meteen weet of het project hierdoor uit zal lopen of juist niet.

11.4.2. KPMG

Het selecteren van een methode om het authenticatie protocol te kiezen was nog lastig. Tijdens de studie hebben wij dit gedaan maar daar was geen duidelijke vaste methode voor. Om toch gebruik te maken van een duidelijke geverifieerde methode is de KPMG pakketselectie methode gebruikt. Omdat de KPMG pakketselectie methode niet voor een protocol selectie is, is er een aangepaste versie gebruikt. De KPMG pakketselectie methode leek veel op de manier die wij tijdens de studie hebben geleerd waardoor deze toch vertrouwd voelde. Tijdens het analyse fase was het lastig om een long list op te stellen. Er zijn veel protocollen en manier om een gebruiker te authentiseren, veel van deze methodes lijken op elkaar waardoor het lastig was om niet te veel dezelfde protocollen te vergelijken. Door de selectie maar met een klein aantal protocollen te doen is de focus van dit project niet te veel op de selectie gevallen. Dit was ook niet de bedoeling, de focus lag op het beveiligen zelf. Ik wilde liever het protocol zelf dieper onderzoeken dan de uitwerking van de maker zelf te vertrouwen.

Omdat de pakketselectie goed verlopen is zal ik vaker deze manier van selecteren gebruiken. De KPMG-pakketselectie lijkt veel op de manier hoe ik tijdens de studie heb geleerd. Doordat de KPMG-pakketselectie ontworpen is vanuit een businesskant ga ik deze manier ook voor niet programeer keuzes gebruiken.

11.4.3. OWASP testing guide v4

Tijdens mijn studie heb ik te maken gehad met een docent die nauw betrokken was bij het ondersteunen van de OWASP organisatie. Door die periode is mij duidelijk geworden dat OWASP een goed en betrouwbare organisatie is. Ik kwam al snel bij verschillende OWASP cheatsheets en testing guides. Omdat ik de site van OWASP niet heel duidelijk vond werken kostte het mij een tijdje voordat ik de OWASP testing guide v4 gevonden had. Deze guide is erg duidelijk en uitgebreid. Er staan veel manieren van tests in geordend op een aantal aspecten van voornamelijk webapplicaties. De aspecten die ik in mijn applicatie behandeld heb, heb ik dus ook gevolgd. Een aantal tests zouden niet met een mobiele applicatie kunnen werken waardoor deze ook niet uitgevoerd zijn.

Als ik een webapplicatie maak zou ik als eerst een inschatting van de situatie maken. Als de situatie bekend is zal ik de onderdelen die het belangrijkst zijn via de OWASP testing guide testen.

11.4.4. Ontwikkel keuzes

De keuze om de applicatie te ontwikkelen in Xamarin forms is goed uitgevallen. Dit heeft veel tijd gescheeld dan de applicatie in Xamarin platform te maken. De UI's zijn voor alle applicaties hetzelfde wat echt heel veel tijd heeft gescheeld. Helaas waren er wel een aantal problemen met Xamarin maar deze problemen zijn niet zo belemmerend geweest wat niet zou opwegen als ik alle applicaties in de native talen had geschreven. Xamarin.forms zal ik vaker gebruiken bij projecten waarbij de UI niet het allerbelangrijkste is van de applicatie.

De keuze om REST te gebruiken is heel er goed geweest. De ondersteuning in Xamarin.forms voor SOAP is heel erg beperkt en zou niet altijd even goed werken. REST is beter ondersteund omdat deze gebruik maakt van de httpclient. Deze httpclient heeft een aantal keer voor problemen gezorgd maar ik verwacht dat deze problemen niet opwegen tegen de problemen die ik bij SOAP zou hebben gehad. REST zal ik voortaan altijd boven SOAP kiezen.

Omdat ik gebruik maak van SCRUM en TFS een scrum template heeft was het een hele makkelijke manier om de user stories te koppelen aan de code change. Per change kon er bekeken worden bij welke user story of task hij hoort. Dit zorgt voor een perfecte versiebeheer. Bij een volgend project zal ik zeker nog een keer deze opstelling gebruiken.

De juiste implementatie vinden voor het protocol bouncy castle was niet heel eenvoudig te vinden. Er zijn verschillende implementaties van SRP. Merendeel is te vinden op Github. Uiteindelijk is voor bouncy castle gekozen omdat deze library ook implementaties heeft voor het encrypten van data. Bouncy Castle bestaat al lang en wordt nog steeds verder ontwikkeld. Dit geeft mij een vertrouwen dat deze library goed werkt. Bij een volgende project waar ik encryptie of iets wat daarmee te maken heeft, toe moet passen zal ik weer bouncy castle gebruiken.

1.2 Product evaluatie

Er is een cross-platform mobiele applicatie gemaakt waarmee de functionaliteit van Coress voor de medewerkers is geïmplementeerd. De applicatie heeft de volgende functies:

- Uren bekijken die al ingevuld zijn
- Uren wijzigingen
- Toevoegen en
- Weekstaten definitief maken

Dit zijn de functionaliteiten die erin moesten zitten en dit is gelukt. De applicatie werkt zowel op iOS, Android en Windows Phone. Hoe de applicatie werkt ben ik tevreden mee. Als de applicatie geopend wordt is in één oogopslag te zien hoeveel uur geboekt is, op de dag, en of de vorige week wel definitief is gemaakt. Het invullen van uren gaat met twee klikken wat snel is. Ik ben blij met de usability. De applicatie werkt zoals hij moet werken. Xamarin is een goed product waarmee cross-platform mobiele applicaties gemaakt kunnen worden. Dit zal ik vaker gaan gebruiken.

11.4.5. SRP

De authenticatie met SRP is gelukt en deze is goed geïmplementeerd. SRP is een goede, veilige manier om gebruikers te authentiseren. Het beste zou zijn geweest al SRP met SSL/TLS geïmplementeerd kon worden. Als ik een volgende project een veilige manier van inloggen moet hebben zal ik dit met SRP doen.

11.4.6. Webservice

De webservice is goed ontworpen en kan gemakkelijk nieuwe functies bij komen en daar hoeft niet veel voor worden aangepast. Ook is de keuze om gebruik te maken van maar één end-point goed geweest. Het zorgt voor iets meer werk maar de veiligheid is hierdoor beter geworden. Dat er gebruik gemaakt is van de Web Api 2 template heeft voor weinig opzet werk gezorgd. Hierdoor kon meer aandacht besteed worden aan het rest van het project. Volgende keer zal ik weer Web Api (2) gebruiken.

12. Beroepstaken

In dit hoofdstuk worden aan de hand van de vooraf gedefinieerde beroepstaken aangegeven of er aan de beroepstaken is voldaan. Per beroepstaak zullen de werkzaamheden die ervoor zorgen dat er wordt voldaan aan de beroepstaak, worden uitgelegd.

12.1. Selecteren methoden, technieken en tools

Om het selecteren van het protocol om de authenticatie uit te voeren is er gebruik gemaakt van de volgende stappen. Als eerst werd er een longlist opgesteld [zie 7.2.1]. Alle protocollen die zeker niet in applicatie paste vielen direct af en met de overgebleven protocollen is een shortlist opgesteld. Aan het uiteindelijk gekozen protocol zijn eisen gesteld. Deze zijn door middel van interviews met de opdrachtgever naar boven gekomen. Door punten toe te kennen aan de protocollen op basis of ze aan de eisen voldoen of gedeeltelijk voldoen is er uiteindelijk een protocol uitgekomen die gebruikt kan worden. Dit protocol zal samen met de verdere security eisen een secure mobiele applicatie worden. Doordat deze methode juist is uitgevoerd is er voldaan aan deze beroepstaak.

12.2. Ontwerpen systeemdeel

Het huidige Coress systeem is een vast systeem wat al jaren door Info Support wordt gebruikt. Door aan dit systeem een mobiele applicatie te koppelen moet de mobiele applicatie gebruik maken van de bestaande ontwerpen. De applicatie heeft een aantal beveiligingseisen zoals dat er een juist authenticatie protocol gekozen is [zie 7]. Dit authenticatieprotocol moest modulair worden ontworpen zodat het makkelijk kan worden vervangen [zie 8]. Door aan deze eisen te voldoen en hiermee rekening te houden is voldaan aan deze beroepstaak

12.3. Bouwen applicatie

De mobiele applicatie is in C# gebouwd met de libaries van Xamarin [zie 6.3]. De mobiele webservice is gebouwd in C# met de template van ASP.NET Web api 2.0. De applicaties zijn getest met unit tests en er is aan versiebeheer gedaan met Team Foundation Server(TFS). Er is ook gebruik gemaakt van een test omgeving. Deze tests werden namelijk uitgevoerd op de devices zelf. Via systeembeheer van Info Support zijn er devices van de drie platformen beschikbaar gesteld. Deze applicaties zijn op de juiste manier gebouwd denkend aan uitbreidbaarheid en hergebruik[zie 8]. Hierdoor is deze beroepstaak gehaald.

12.4. Uitvoeren van en rapporteren over het testproces

Door gebruik te maken van de testing guide van OWASP is er penetration test uitgevoerd. Met deze testen is er aan systeembeheer bewezen dat de applicatie veilig genoeg is voor productie. [zie 10]. Door de juiste testen te kiezen, deze op de juiste manier uit te voeren en te rapporteren is deze beroepstaak behaald. Daarnaast is er in week 19 een gesprek geweest met een security expert die mij gewezen heeft op een aantal belangrijke tests om uit te voeren.

13. Bronnenlijst:

1. http://xamarin.com/platform
2. http://xamarin.com/forms
3. https://msdn.microsoft.com/en-us/library/aa287558(v=vs.71).aspx
4. Http://www.mono-project.com
5. http://www.itinbedrijf.info/0208_KPMG.pdf
6. https://math.berkeley.edu/~kpmann/encryption.pdf
7. https://www.owasp.org/images/5/52/OWASP_Testing_Guide_v4.pdf
8. https://en.wikipedia.org/wiki/Authentication_protocol
9. https://www.ing.nl/nieuws/nieuws_en_persberichten/2014/11/drie_jaar_na_introductie_ing_mobiel_bankieren_2_miljoen_gebruikers.html
10. https://www.mountknowledge.nl/2011/11/09/ing-mobiel-bankieren-authenticatie/

Bijlage A - Afstudeerplan

Mobiele Coress

Allard Soeters

Auteur	Allard Soeters
Studentnummer	11114762
Studie	Informatica
School	Haagse Hogeschool
Eerste examinator:	G.A. Mijnarends
Tweede examinator:	H.G.J. Bechet
Opdrachtgever:	R. Flohil
Afstudeerbegeleidster:	P. Hijl
Technisch Begeleider:	E. van Alebeek
Business Unit manager:	H. Brands
Plaats	Zoetermeer
Datum	24-08-2015
Bedrijf	Info Support B.V.

Afstudeerplan

Informatie afstudeerder en gastbedrijf

Afstudeerblok: 2015-1.2
Startdatum uitvoering afstudeeropdracht: 11 mei 2015
Inleverdatum afstudeerdossier volgens jaarrooster: 5 oktober 2015

Studentnummer: 11114762
Achternaam: dhr Soeters
Voorletters: A
Roepnaam: Allard
Adres: Westduinweg 54C
Postcode: 2583EJ
Woonplaats: Den Haag
Telefoonnummer: 0631691829
Mobiel nummer:
Privé emailadres: allardsoeters@gmail.com

Opleiding: Informatica
Locatie: Den Haag
Variant: voltijd

Naam studieloopbaanbegeleider: de heer G. Tuk
Naam begeleidend examinator: de heer G.A. Mijnarends
Naam tweede examinator: de heer W. van Vliet

Naam bedrijf: Infosupport
Afdeling bedrijf:
Bezoekadres bedrijf: Kruisboog 42
Postcode bezoekadres: 3905TG
Postbusnummer:
Postcode postbusnummer:
Plaats: Veenendaal
Telefoon bedrijf: +31 318552020
Telefax bedrijf:
Internetsite bedrijf: http://www.infosupport.nl

Achternaam opdrachtgever: dhr Flohil
Voorletters opdrachtgever: R
Titulatuur opdrachtgever:
Functie opdrachtgever: Software Engineer .NET
Doorkiesnummer opdrachtgever: +31 (0)6 552 523 39
Email opdrachtgever: rutger.flohil@infosupport.com

Achternaam bedrijfsmentor: mw Hijl
Voorletters bedrijfsmentor: P
Titulatuur bedrijfsmentor:
Functie bedrijfsmentor: Opleidingsadviseur
Doorkiesnummer bedrijfsmentor: +31 (0)6 45378507
Email bedrijfsmentor: Pascalle.Hijl@infosupport.com

NB: bedrijfsmentor mag dezelfde zijn als de opdrachtgever

Doorkiesnummer afstudeerder:
Functie afstudeerder (deeltijd/duaal):

Titel afstudeeropdracht:

Verbeteren van het urenregistratiesysteem bij Infosupport

Opdrachtomschrijving

1. Bedrijf

Info Support is een specialist in het ontwikkelen, beheren en hosten van software op maat en het leveren van BI- en integratieoplossingen. Ook heeft Info Support een aanbod van ruim 300 trainingen, veelal gericht op softwareontwikkeling. Het hoofdkantoor is gevestigd in Veenendaal. Verspreid over vestigingen in Nederland en België heeft Info Support meer dan 400 medewerkers in dienst. Info Support levert innovatieve, branchegerichte oplossingen, waarbij zij de nodige branche- en IT-kennis inzet. Als het gaat om de achterliggende technologie kiest Info Support voor Microsoft (.NET, C#, SQL Server, Azure), Oracle (Fusion Middleware, Weblogic Suite) en Java.

2. Probleemstelling

In de huidige samenleving geven smartphones ons veel nieuwe mogelijkheden die ons het leven eenvoudiger maken. De drijfveer achter deze ontwikkelingen is echter vaker de consumentenmarkt dan de zakelijke.

Binnen Info Support wordt er gebruik gemaakt van een applicatie om onze uren te registreren. Deze applicatie is wegens veiligheidsredenen verstopt achter een intern netwerk en hierdoor niet gericht om even snel wat uren te registreren via, bijvoorbeeld, een mobiele telefoon.

Vooral bij consultants die voor veel klanten tegelijkertijd werken is dit soms erg onhandig. Zij hebben veel urencodes in te vullen.

Daarom is er de wens om dit moderner te maken door urenregistratie mogelijk te maken met een smartphone.

3. Doelstelling van de afstudeeropdracht

Met en proof of concept van een multi-platform mobiele applicatie aantonen dat de beveiliging van infosupport gewaarborgd kan blijven als er een zo'n dergelijke applicatie extern wordt uitgebracht.

4. Resultaat

Met het proof of concept kan een volwassen applicatie uitgewerkt worden zodat deze uiteindelijk gebruikt kan worden de mobiele applicatie om gemakkelijk uren in te voeren. Zeker voor concultants is dit een hele verbetering.

De taal waar het proof of concept in geschreven zal worden is c# met behulp van het Xamarin framework zal er daar een multi-platform applicatie van gemaakt worden.

5. Uit te voeren werkzaamheden, inclusief een globale fasering, mijlpalen en bijbehorende activiteiten

Als allereerste zal er een plan van aanpak geschreven moeten worden. Dit zal ik in de eerste week uitvoeren. Daarnaast zal ik in de eerste week inwerken en een afspraak maken met de systeembeheerder van het huidige systeem. Ik hoop die afspraak in de tweede week te hebben. In de tijd dat ik nog geen afspraak heb zal ik de tijd besteden aan het verdiepen in het Xamarin framework. Ik zal met de systeembeheerder kijken naar welke eisen, qua beveiliging, er gesteld worden aan de toekomstige multi-platform mobiele applicatie. Samen met de opdrachtgever/begeleider zal daar een selectie op gemaakt worden op deze eisen, rekening houdend met de tijd. Het kan ook zijn als alle eisen binnen de tijd onderzocht kunnen worden dat alle eisen gekozen zullen worden. Zodra er een lijst met beveiligings eisen is moet er een risicoanalyse worden uitgevoerd. Waar liggen de grootste gevaren dit zal ongeveer twee weken in beslag nemen. Deze eisen zullen ieder de volgende stappen ondergaan:

- Onderzoeken naar de eis zelf.
- Welke techiek(en) bestaan er om deze eis te voldoen.
- Indien nodig een techniek kiezen op basis van de risico
- Met een proof of concept bevestiging dat er voldaan is aan de eis met de geselecteerde techniek.
- Als de eis niet voldaan is zal er een andere techniek gekozen moeten worden.
- Er wordt gekeken hoeveel impact de beveiligingseis heeft op de bestaande infrastructuur

Per eis wil ik ongeveer twee weken besteden. Sommige zullen langer duren dan andere.

Als alle techieken gekozen zijn zullen deze verwerkt worden in een ontwerp. Ook zullen alle deel proof of concepts verwerkt worden tot één proof of concept met behulp van het xamarin framework in C#. Dit zal in de overige weken zitten, rond de twee à drie weken. In de laatste drie weken zal ik het afstudeerverslag schrijven.

6. Op te leveren (tussen)producten

Tussenproducten:

- Alle deel rapporten
- Alle deel proof of concepts (in C#)

Eind producten:

- Ontwerp van de proof of concept.
- Proof of concept (met het Xamarin framework in C#)
- Test rapport

7. Te demonstreren competenties en wijze waarop

Competentie 1.1:
Er moet een onderzoek worden uitgevoerd naar de beveiliging van de communicatie tussen beide software. Ook moet er onderzoek gedaan worden naar het huidige systeem en hoe dit nu werkt. Deze competentie zal getraceerd kunnen worden naar de deel rapporten van iedere beveiligingseis. Hier wordt de techniek gekozen die het beste bij die eis past. Als er geen onderzoek uitgevoerd worden kan de rest van de opdracht niet uitgevoerd worden dus is deze competentie erg belangrijk.

Competentie 3.2:
Het bouwen van een communicatie tussen het urenregistratiesysteem en de mobiele applicatie is het ontwerpen van een systeemdeel. Het urenregistratiesysteem bestaat al. Deze competentie kan getraceerd worden naar het ontwerp van de proof of concept. De beveiliging staat centraal in deze opdracht. Het systeemdeel is een belangrijke schakel daarin.

Competentie 3.3:
Bij het bouwen van de proof of concept moet nadrukkelijk rekening gehouden worden met de beveiliging. Deze competentie kangetraceerd worden naar het ontwerp van de proof of concept en de proof of concept zelf. Deze competentie heeft een grote impact op de opdracht omdat er niks bewezen kan worden als de applicatie niet gemaakt wordt.

Competentie 3.5:
Na het bouwen van de proof of concept zal ik deze moeten testen of wel aan alle beveiliginseisen is voldaan. Deze competentie kan getraceerd worden naar het test rapport. De impact op de opdracht is bij deze competentie niet groot maar het betreft wel een belangrijk onderdeel van het systeem.

Bijlage C - SRP

Mobiele Coress

Allard Soeters

Auteur	Allard Soeters
Studentnummer	11114762
Studie	Informatica
School	Haagse Hogeschool
Eerste examinator:	G.A. Mijnarends
Tweede examinator:	H.G.J. Bechet
Opdrachtgever:	R. Flohil
Afstudeerbegeleidster:	P. Hijl
Technisch Begeleider:	E. van Alebeek
Business Unit manager:	H. Brands
Plaats	Zoetermeer
Datum	24-08-2015
Bedrijf	Info Support B.V.

1. Geschiedenis

Encryptie bestaat al heel lang men zegt dat er niet standaard hiëroglyfen rond de 2000 jaar voor christus op muren geschreven werden zodat andere mensen het niet konden lezen behalve de mensen waarvoor het bedoelt was, alhoewel dit niet zeker is. De hiëroglyfen zouden ook bedoelt kunnen zijn voor amusement. 5 jaar voor christus is de scytale door spartanen de scytale bedacht. Deze scytale is een houten staf, dit kan een potlood zijn maar ook een wapenstok. Om deze staf wikkelt men een stuk perkament of ander middel om tekst op te schrijven. Zodra deze diagonaal om de staf gewikkeld is kan er horizontaal een tekst op geschreven worden. Zodra de tekst klaar is kan het stuk perkament van de staf gehaald worden. Er zal dan een geencrypte tekst op het perkament te lezen zijn. Zodra de ontvanger hem om zijn staf heen draait zal de tekst weer leesbaar zijn. De dikte van de staf van de ontvanger moet wel even dik zijn als de staf van de verzender.

Deze vorm van encryptie waar beide partijen, die we Alice en Bob noemen, een afgesproken middel hebben om de tekst te ontcijferen wordt hedendaags nog steeds gebruikt. De mogelijke persoon die de tekst niet mag lezen maar wel wilt lezen noemen we Steve. De staf is vervangen door een code. Deze code kan met behulp van een algoritme(cipher) de tekst decrypten en encrypten. Dit werkt goed zodra Alice en Bob dezelfde code hebben. Als er maar twee partijen met elkaar willen communiceren is dit geen probleem maar zodra er een stuk of 100 partijen met elkaar moeten communiceren is dit een nachtmerrie om voor iedereen codes uit te delen en om deze te bewaren. In de 1980's moest er een heftruck aan de pas komen om alle codes van de US Navy op een schip te laden om met andere schepen te kunnen communiceren. Deze methode van encryptie wordt symmetrische encryptie genoemd.

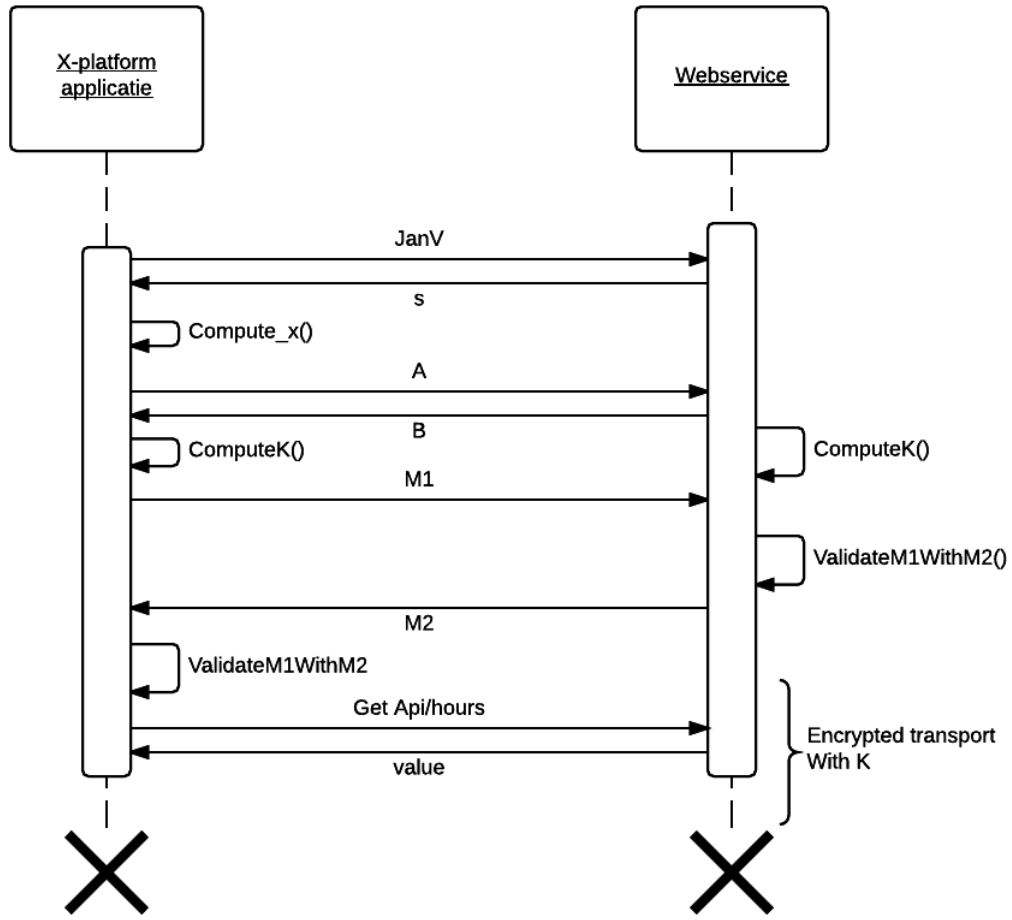
In 1976 hebben twee wiskundige van Standford dit probleem opgelost. Dit waren Whitfield Diffie en Martin Hellman. Zij bedachten de asymmetrische encryptie. Deze soort encryptie zorgt ervoor dat er een code is om de tekst te encrypten maar dat de geencrypte tekst met een andere code gedecrypt moet worden. In theorie hebben zij dit bedacht maar later is dit pas door drie wiskundige van MIT, Rivest, Shamir en Adleman(RSA) in praktijk gebracht. RSA encryptie heeft het basis principe dat het heel gemakkelijk is om twee getallen met elkaar te vermenigvuldigen maar het is heel moeilijk om de uitkomst weer terug te brengen naar de twee getallen. Dit wordt in de cryptografie een trapdoor genoemd.
RSA werkt in het simpel op de volgende manier:
encryptie
M = bericht
e = (p-1)*(q-1) en daar een relatieve prime van.
N = p*q
Geencrypt bericht = (M^e)mod N
N en e zijn publiek bekend.

Decryptie
ed = 1%(p-1)(q-1)
e weet je maar d wil je weten om het bericht te decrypten.
(C^d)mod N = gedecrypte bericht..

Omdat asymmetrische encryptie een zwaardere taak is dan symmetrische encryptie wordt er een combinatie van beide gebruikt. De code om de symmetrische encryptie uit te voeren wordt doormiddel van asymmetrische encryptie uitgewisseld.

2. SRP

SRP is een manier van authenticatie en encryptie. SRP wordt augemented password key agreement genoemd. Dit houdt in dat er vooraf een wachtwoord afgesproken is en deze wordt doormiddel van public keys en wiskundige formules gecontroleerd. Hierdoor hoeft er geen password over het internet gestuurd worden.
In het volgende sequence diagram is te zien hoe de authenticatie van SRP gaat.



De identifiers zijn via onderstaande tabel uitgelegd.

Naam	Identifier	Uitleg
Salt	s	een hulpmiddel om het password te hashen. De salt bestaat uit een aantal bits.
Groot priemgetal	n	Een groot priemgetal wat bij meerdere functies wordt gebruikt.
Password	P	Het password van de identiteit is bekend bij de server en client.
Lange Private key	x	Een gehashed idnetity+password+salt modulo n. Deze zal bij iedere sessie hetzelfde zijn.
Generator	g	Een primitieve wortel van n.
Verifier	v	De generator tot de macht x modulo n

Naam	Identifier	Uitleg
Korte private keys	a, b	Moet groter zijn dan 1 maar kleiner moet zijn dan n. Deze is geheim en wordt niet gedeeld.
Korte public key client	A	g tot de macht a modulo n
Korte public key server	B	De public key van de server is g tot de macht b modulo n de uitkomst plus v en dan modulo n.
Een random getal	u	Deze is voor client en server hetzelfde. Hash(A,B) modulo n
Session key	S	Zie session Key
Gehashde session key	K	Hash(S)
Waarde om te valideren	M1	Gehashede waarde van (A,B,S) modulo n
Waarde om te valideren	M2	Gehashede waarde van (A,M1,S) modulo n

Zodra de K berekent is zal bij de client en server gekeken worden of ze allebei wel dezelfde S waarde berekent hebben. Dit gebeurd met de M1 en M2.
Er zijn maar een aantal waardes die Steve, man in the middle, af kan luisteren:

1. Identity
2. Salt
3. Public key A
4. Public key B
5. Gehashde validatie waarde M1
6. Gehashde validatie waarde M2

Steve kan met deze waardes niet achter de session key of password komen. Dit komt door de asymmetrische kant van SRP.

Omdat de berekeningen alleen met een BigInteger gedaan kunnen worden moeten alle identity tekst omgezet worden naar een BigInteger. Dit moet bij de volgende de x gebeuren. met de onderstaande code wordt x gegenereerd:

```
public static BigInteger CalculateX(IDigest digest, BigInteger N, byte[] salt, byte[]
identity, byte[] password)
{
        byte[] output = new byte[digest.GetDigestSize()];
        digest.BlockUpdate(identity, 0, identity.Length);
        digest.Update((byte)':');
        digest.BlockUpdate(password, 0, password.Length);
        digest.DoFinal(output, 0);
        digest.BlockUpdate(salt, 0, salt.Length);
        digest.BlockUpdate(output, 0, output.Length);
        digest.DoFinal(output, 0);
        return new BigInteger(1, output).Mod(N);
}
```

De identity, password en salt worden eerst gehashed. Deze hash zijn bytes en van deze bytes worden een BigInteger gemaakt.

2.1 Session key

Om de session key te bereken gebruikt bouncy castle, een cryptografie library, een andere methode dan de theorie zegt.

2.1.1 client

In Theorie:
1: (((u*x)%n) + A)%n 2: (((b – g)%n)^x)%n
S = (2^1)%n
Bouncy castle:
B = B%n
K= hash(n,g)%n
Exp = (u * x)+a
Tmp = (((g^x)%n) * K)%n
S = (((B – tmp)%n) ^ exp)%n

De session key wordt met de volgende code berekend:

```
this.B = Srp6Utilities.ValidatePublicValue(N, serverB);
this.u = Srp6Utilities.CalculateU(digest, N, pubA, B);
this.S = CalculateS();

private BigInteger CalculateS()
{
	BigInteger k = Srp6Utilities.CalculateK(digest, N, g);
	BigInteger exp = u.Multiply(x).Add(privA);
	BigInteger tmp = g.ModPow(x, N).Multiply(k).Mod(N);
	return B.Subtract(tmp).Mod(N).ModPow(exp, N);
}
```

2.1.2 Server

In theorie:
1: (A*v)%n 2: ((1)^u)%n
S = (2^b)%n
Bouncy castle:
A = A%n u = hash(A,B)%n
1: (v^u)%n 2 (1*A)%n
K= (2^B)%n

De session key wordt op de volgende manier berekent:

```
this.A = Srp6Utilities.ValidatePublicValue(N, clientA);
this.u = Srp6Utilities.CalculateU(digest, N, A, pubB);
this.S = CalculateS();
private BigInteger CalculateS()
{
	return v.ModPow(u, N).Multiply(A).Mod(N).ModPow(privB, N);
}
```

2.2 Example

#	Client	Server	Hoe theorie
1	client bereken x		Hash(Salt,password)%n
2	client bereken a		1 < a < n
3	client bereken A		(g^a)%n
4		server bereken b	1 < a < n
5		server bereken B	(v + (g^b)%n)%n
	Bereken S op basis van de public keys		
6	client breken u		Hash(A,B)%N
7	client bereken s		Zie session Key client
8		server bereken u	Hash(A,B)%N
9		server bereken s	Zie session key

Simpel 1:
n = 5
g = 2
1: x = 20
v = (g^x)%n = (2^20%5 = 1
2: a = 1 < a < n = random 3
3: A = (g^a)%n = (2^3)%5 =3
4: b = 1 < b < n = random 3
5: B = (((g^b)%n)+v)%n =(((2^3)%5)+1)%5 = 3
6: u = Hash(A,B)%n = Hash(3,3)%5 =
213490145027444934348930817281906707139273646721790006192559375592522238257251017403039479428059380086102319350325780482944971956061941762820935849595234 7
7a: k = Hash(n,g)%n = Hash(5,2)%5
=473606881918153382656322107629534773070856299796717614893027144998834586155899640522478676721772600035605627318949163875832435460645534221091548105381826 0
7b: exp = (u*x)+a = (u*20)+3
=426980290054889868697861634563813414278547293443580012385118751185044476514502034806078958856118760172204638700651560965889943912123883525641871699190469 43
7c: tmp = ((g^x)%n *k)%n = ((2^20)%5 *k)%5 =0
7: Sclient = ((b-tmp)%n ^ exp)%n = ((3-tmp)%5 ^ exp)%5 =2
8: u =
213490145027444934348930817281906707139273646721790006192559375592522238257251017403039479428059380086102319350325780482944971956061941762820935849595234 7
9a: 1 = (v^u)%n = (1^u)%5 = 1
9b: 2 = (1*A)%n = (1*3)%5 = 3
9: sServer = 2

Simpel 2:
1: x = 15212154
v = (g^x)%n = (2^15212154%53 = 43
2: a = 1 < a < n = 6
3: A = (g^a)%n = (2^6%53 =11
4: b = 1 < b < n = 25
5: B = (((g^b)%n)+v)%n =(((2^25)%53)+43)%53 = 23

6: u = Hash(A,B)%n = Hash(11,23)%53 =
84168918201528360400734849876441797869499447865168428346377048078285132602031962521373537752597576830394094661645660300773693495150807048374037998596016127a: k = Hash(n,g)%n = Hash(53,2)%53 =
73255645310541572845649145694635237149552181711202250122410667470968268976053194934580789732564350459114609303293783044028254573498800021705839186711656617b: exp = (u*x)+a = (u*x)+6
=12803905456950524539750383991336794305299743227749305253092679736254781732823877078684868353167792705574459703233004415131563649438100392254
7c: tmp = ((g^x)%n *k)%n = ((2^15212154)%53 *k)%53 =50
7: Sclient = ((b-tmp)%n ^ exp)%n = ((25-tmp)%53 ^ exp)%53 =6

8: u =
84168918201528360400734849876441797869499447865168428346377048078285132602031962521373537752597576830394094661645660300773693495150807048374037998596016129a: 1 = (v^u)%n = (43 ^u)%53 = 49
9b: 2 = (1*A)%n = (49*11)%53 = 9
9: sServer = (9^23)%53 = 6

3. Bronnenlijst

1. https://math.berkeley.edu/~kpmann/encryption.pdf
2. http://www.cypher.com.au/crypto_history.htm
3. http://www.australianscience.com.au/technology/a-scytale-cryptography-of-the-ancient-sparta/
4. http://srp.stanford.edu/ndss.html

Bijlage D – Functioneel ontwerp

Mobiele Coress

Allard Soeters

Auteur	Allard Soeters
Studentnummer	11114762
Studie	Informatica
School	Haagse Hogeschool
Eerste examinator:	G.A. Mijnarends
Tweede examinator:	H.G.J. Bechet
Opdrachtgever:	R. Flohil
Afstudeerbegeleidster:	P. Hijl
Technisch Begeleider:	E. van Alebeek
Business Unit manager:	H. Brands
Plaats	Zoetermeer
Datum	24-08-2015
Bedrijf	Info Support B.V.

User stories

#	Wat	Waarom
1	Als gebruiker wil ik uren invullen aan de hand van de uur codes	Omdat ik dan gemakkelijk van de app de uren in kan vullen. En niet speciaal naar de website moet gaan.
2	Als gebruiker wil ik uitloggen zodra ik klaar ben	Hierdoor kan een ongeautoriseerde persoon mijn uren invullen.
3	Als gebruiker wil ik uren pas definitief maken als ik dat explicitiet aangeven	Hierdoor kan ik geen foutieve weekstaat doorsturen naar de fiatteur
4	Als gebruiker wil ik niet in kunnen loggen als ik 10 keer een foutief wachtwoord heb ik in gevuld	Omdat een ongeautoriseerde persoon dan niet een brute force aanval kan uitvoeren
5	Als gebruiker wil ik dat de gewerkte uren prominenter aanwezig zijn dan de overige uren	Omdat ik dan gemakkelijke de meeste gebruikte uren kan vinden
16	Als gebruiker wil ik uren invullen aan de hand van een diverse code	Omdat ik soms ook ziek ben of verlof neem.
6	Als eigenaar wil ik dat de verbinding veilig opgezet en veilig is	Omdat dan een ongeautoriseerde persoon niet mijn gegevens kan onderscheppen.
7	Als gebruiker wil ik dat mijn uren opgeslagen worden zodra ik ze invul	Omdat ik dan niet een extra handeling moet uitvoeren
8	Als gebruiker wil ik dat ik de app niet kan gebruiker als ik dat ook niet hoef te doen	Omdat als ik niet weekstatigplichtig ben ook niet voor niks uren in ga vullen
9	Als gebruiker wil ik een push notificatie krijgen als ik vergeten ben mijn uren in te vullen	Omdat ik soms vergeet mijn uren in te vullen.
10	Als gebruiker wil ik een pincode gebruiken om in te loggen	Omdat ik dan gemakkelijker dan de huidige situatie kan inloggen
11	Als gebruiker wil ik mijn device aanmelden en ook af kunnen melden bij een interne website	Zodat ongeautoriseerde personen niet mijn gegevens kunnen inzien en

		bewerken als ik mijn telefoon bijv verkocht heb
12	Als gebruiker wil ik veilig in kunnen loggen	Omdat ik niet wil dat een ongeautoriseerde persoon mijn inlog gegevens kan stelen
13	Als gebruiker wil ik dat mijn uren die ik via de computer invul ook in mijn app komen	Omdat ik dan niet dubbel werk doe.
14	Als gebruiker wil ik een alias geven aan de uur codes die voorkomen.	Omdat ik dan sneller de juiste uurcode kan vinden.
15	Als gebruiker wil ik als ik offline ben toch nog mijn uren in kan vullen	Omdat ik niet altijd internet heb en toch mijn uren wil invullen.

1.1 Ingevulde uren bekijken.

1.1.1 User story titel

“Als gebruiker wil ik dat mijn uren die ik niet via de cross-platform applicatie invul ook in mijn applicatie komen omdat ik dan altijd weet wat ik al heb ingevuld.”

1.1.2 Uitleg

Als de medewerker het uren overzicht opent. Worden de benodigde gegevens van de server opgehaald. In deze gegevens zitten per dag alle verschillende soorten uren namelijk:

- Ort-uren
- Standby uren
- Over uren
- Normale uren
- Aantal kilometers gereden die dag.

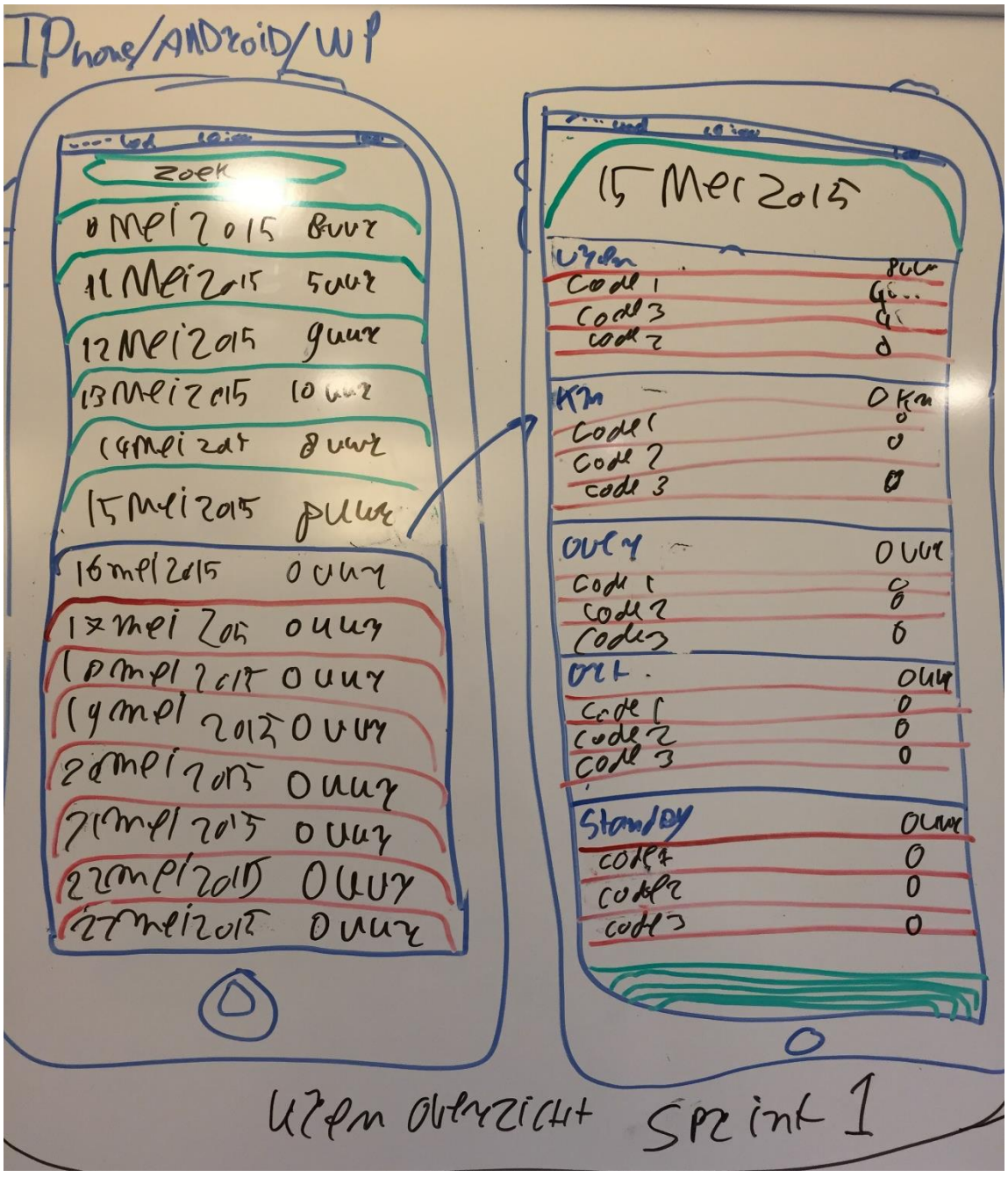
In het overzicht worden per dag de volgende gegevens getoond:

- Totaal aantal uren van die dag,
- De datum
- De status van de weekstaat waar die dag bij hoort.

De status van de weekstaat zal door middel van een gekleurd blokje geuit worden. Rood staat voor nog niet goed gekeurd. Geel staat voor toekomstige dagen. Wit staat voor vandaag en groen staat voorgoed gekeurde weekstaat.

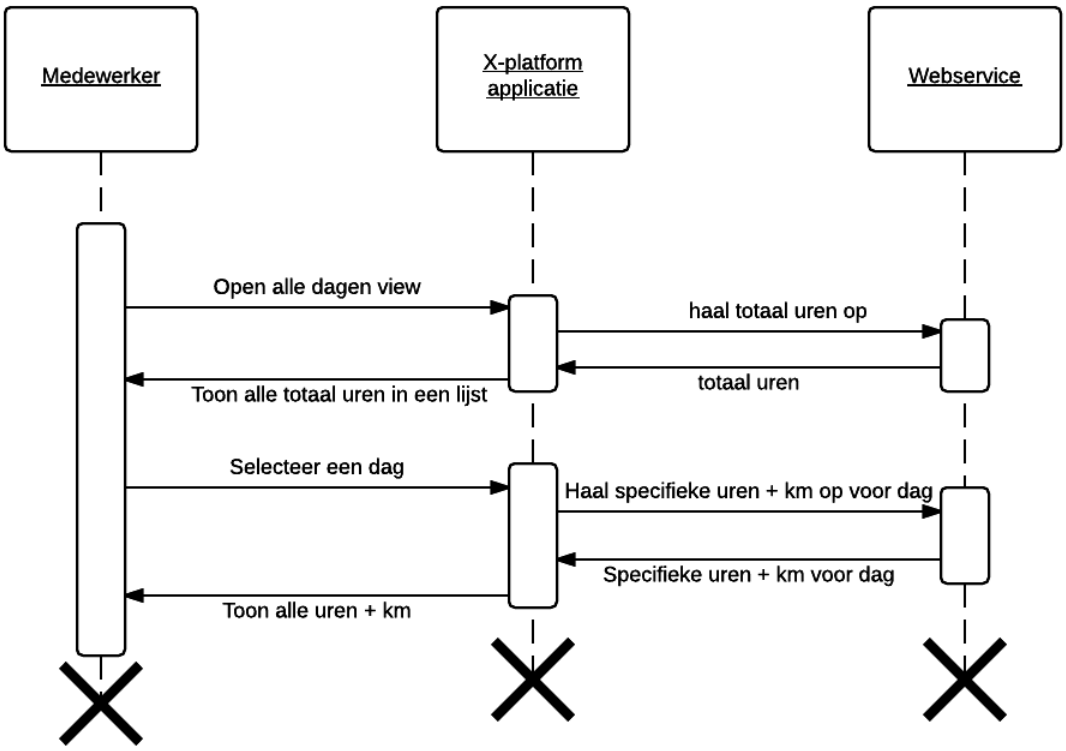
Als op een dag geklikt wordt zal er een overzicht getoond worden met de verschillende soorten uren en de aantal kilometers. De uren zijn onderverdeeld in uur codes.

1.1.3 Schermontwerp



Figuur 1 wireframe “Ingevulde uren kijken”

1.1.4 Sequence diagram



Figuur 2 sequence digagram "Ingevulde uren kijken"

1.2 Inloggen.

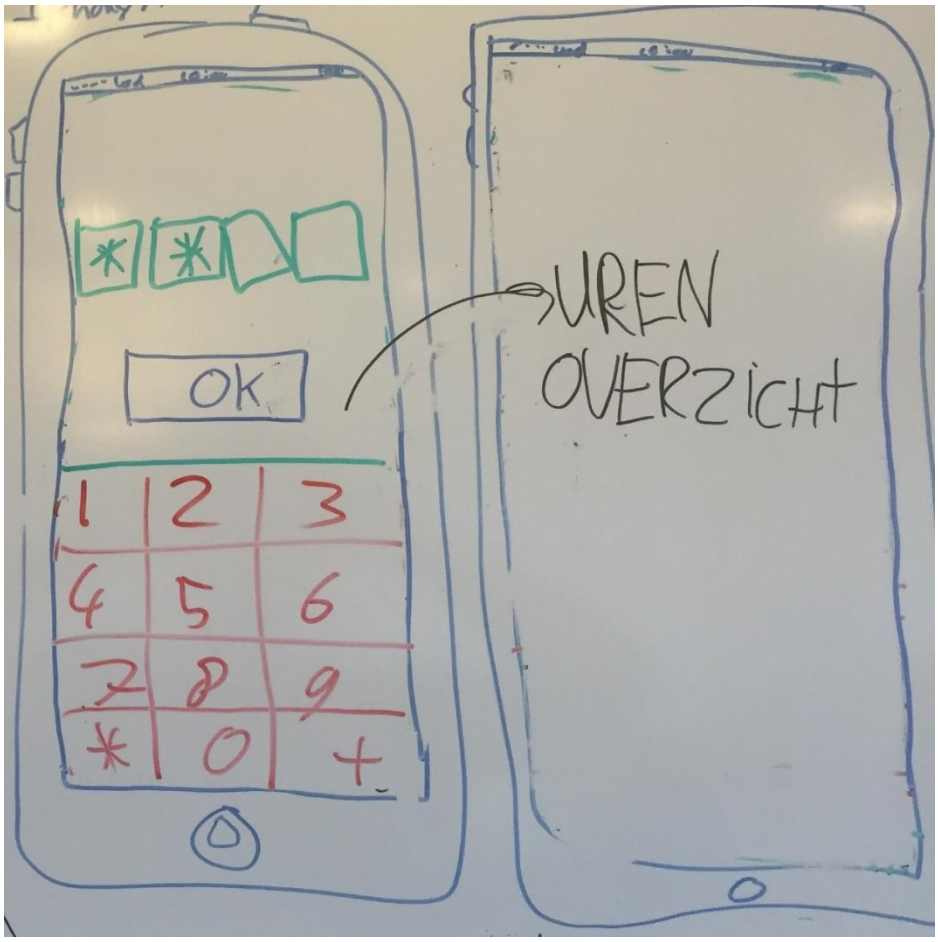
1.2.1 User story titel

“Als gebruiker wil ik inloggen zodat ik gebruik kan maken van de applicatie.”

1.2.2 Uitleg

De actor moet zijn device geregistreerd hebben. En moet een pincode hebben. Met deze pincode kan de actor inloggen. Zodra de pincode is ingevoerd zal er aan de backend gevraagd worden of de pincode juist is. Zodra deze juist is kan de applicatie gebruikt worden.

1.2.3 Schermontwerp



Figuur 3 Wireframe “Inloggen”

1.3 Uren toevoegen (niet divers)

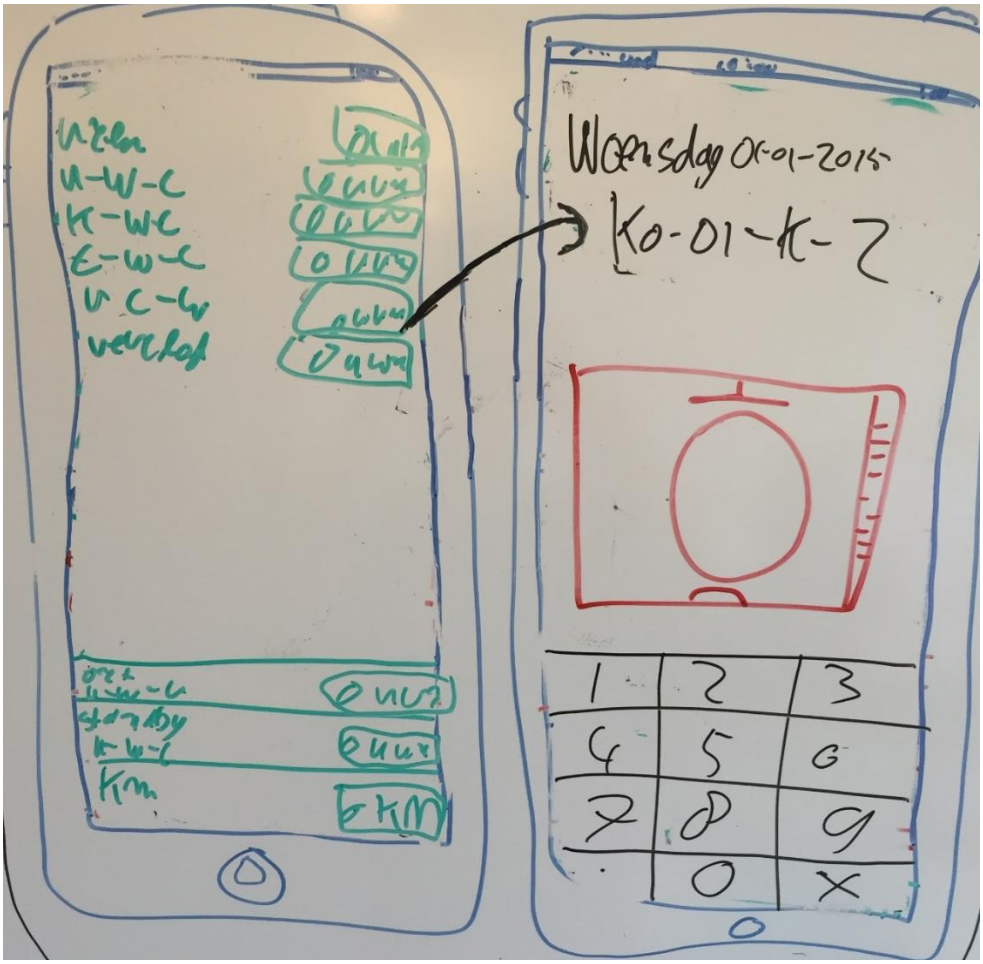
1.3.1 User story titel

“Als gebruiker wil ik uren invullen aan de hand van de uur codes”

1.3.2 Uitleg

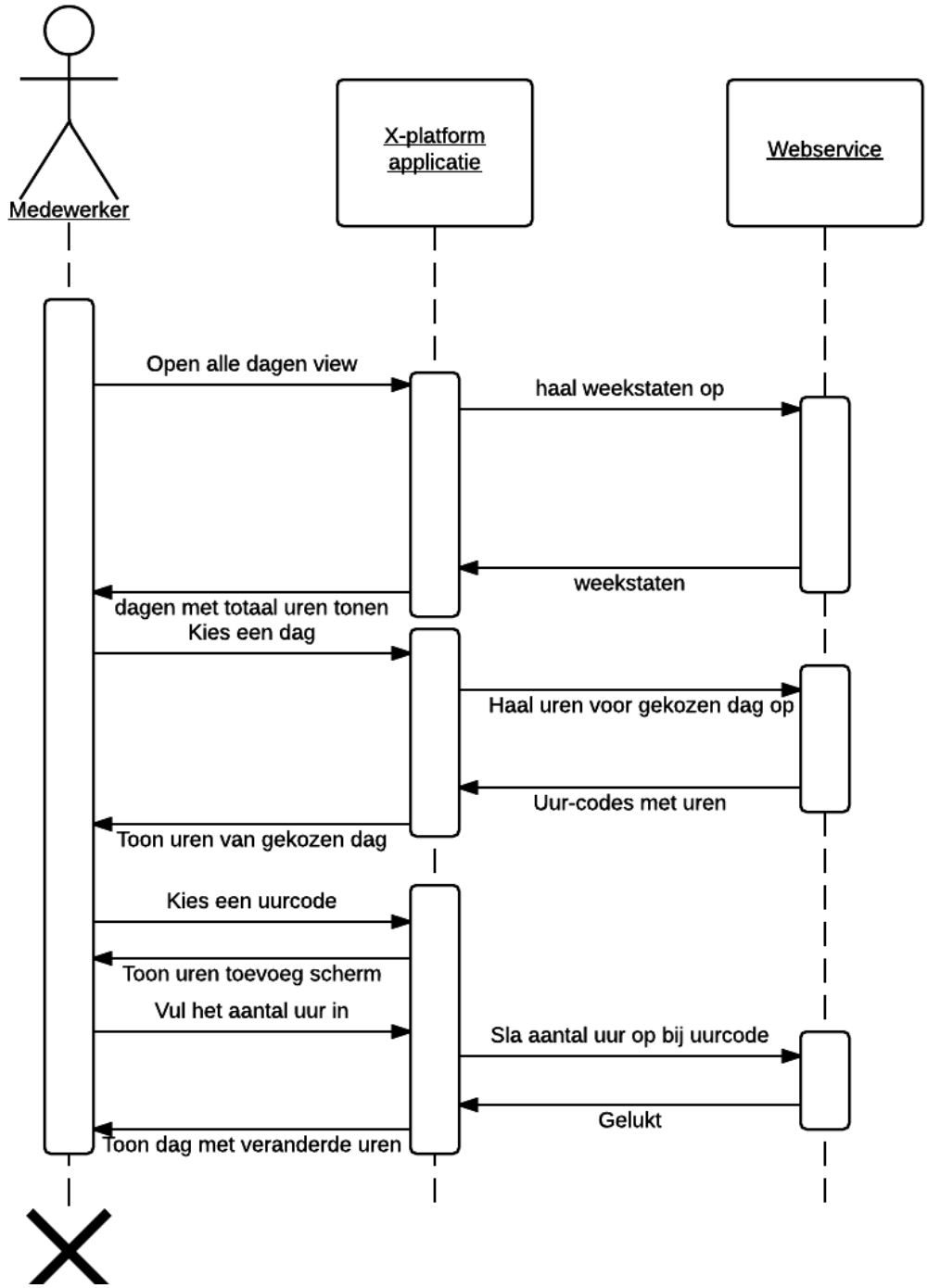
Een medewerker moet zijn uren in kunnen vullen. Dit doet hij per dag per uurcode. De uurcodes worden per dag getoond. De medewerker klikt op de uurcode en dan kan de medewerker zijn uren invullen.

1.3.3 Schermontwerp



Figuur 4 Wireframe "Uren toevoegen"

1.3.4 Sequence diagram



Figuur 5 sequence diageram "Uren toevoegen"

1.4 Diverse uren toevoegen

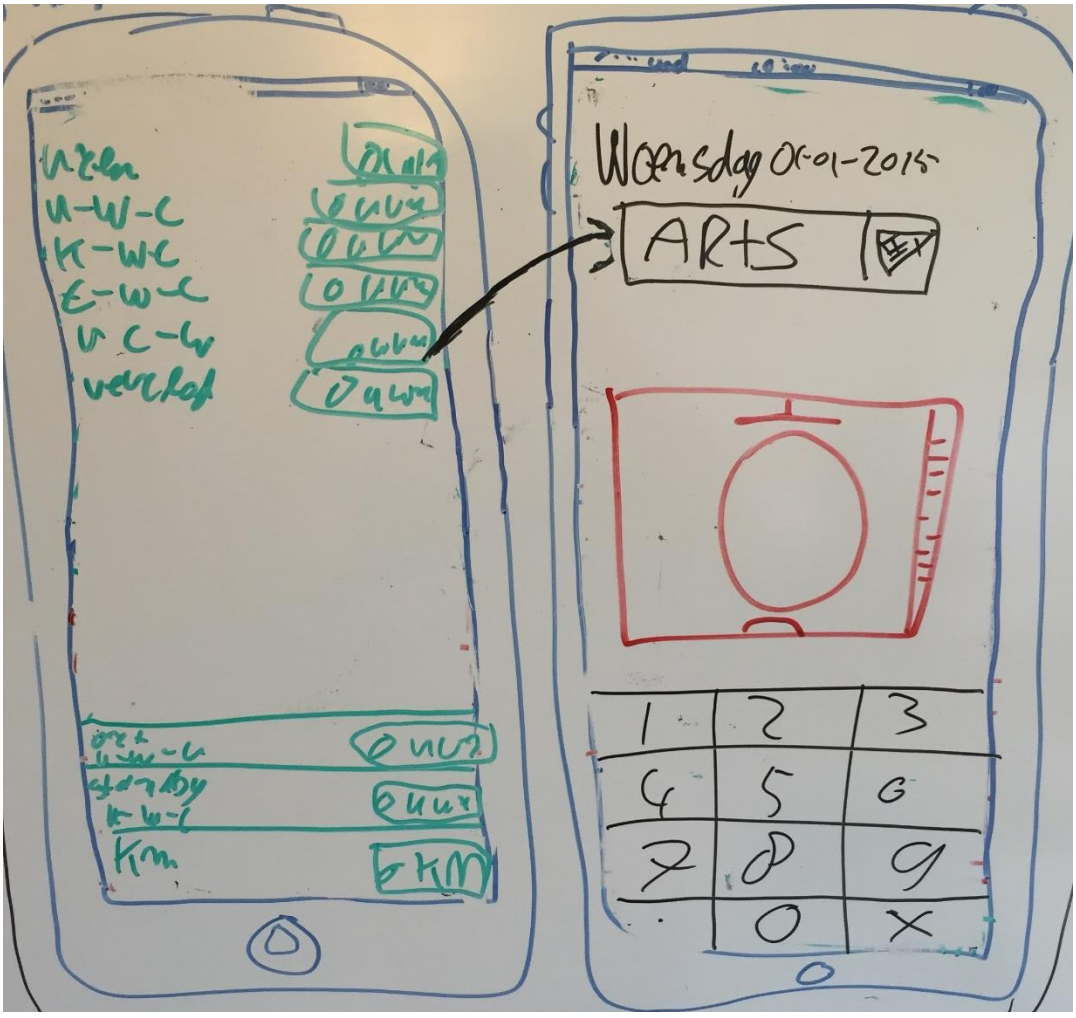
1.4.1 User story titel

“Als gebruiker wil ik uren invullen aan de hand van de uur codes”

1.4.2 Uitleg

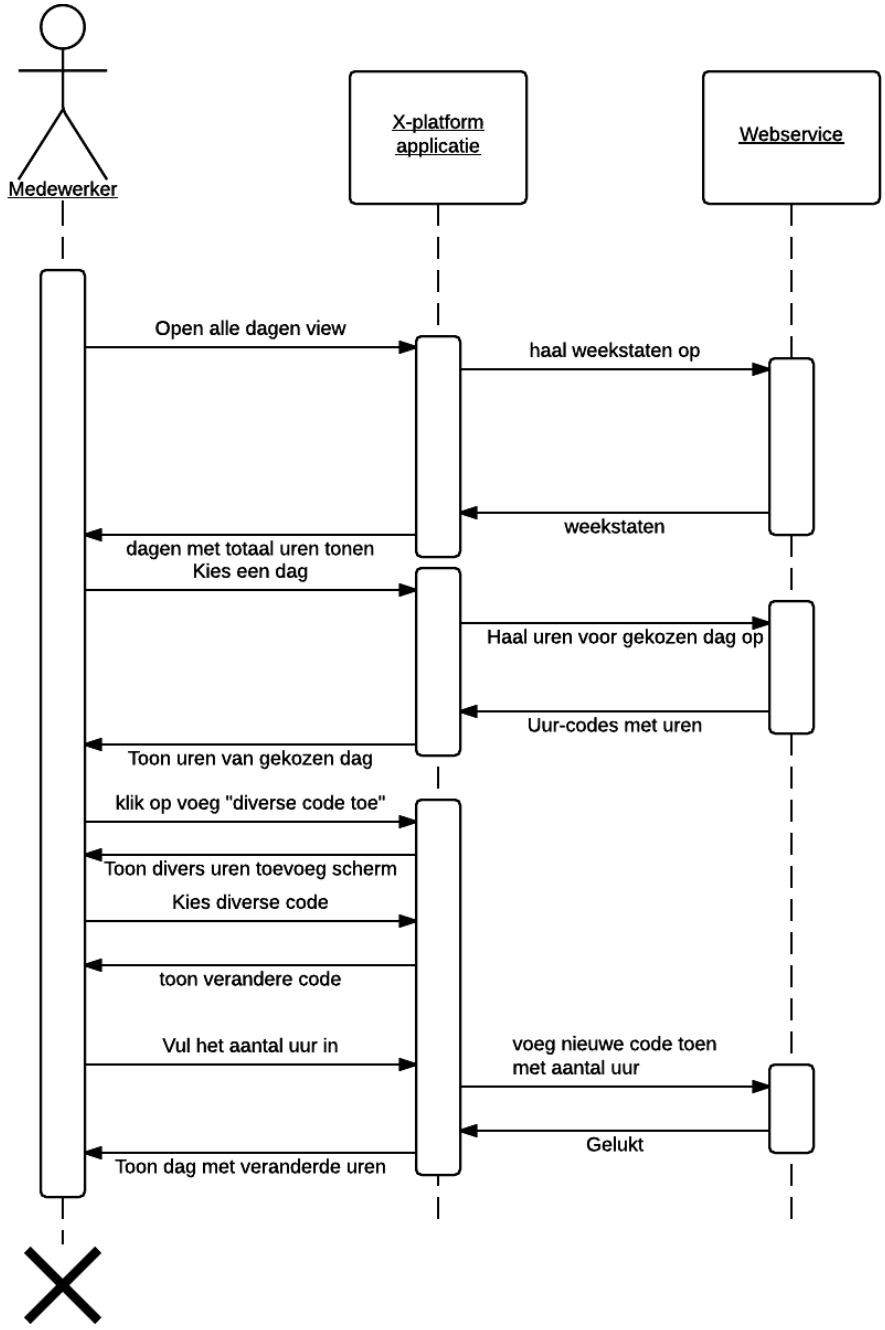
De diverse codes worden op een iets andere manier toegevoegd. Hij valt onder dezelfde user s

1.4.3 Schermontwerp



Figuur 6 Wireframe “Diverse uren toevoegen”

1.4.4 Sequence diagram



Figuur 7 Sequence digram “Diverse uren toevoegen”

1.5 Device verwijderen.

1.5.1 User story titel

“Als gebruiker wil ik mijn device kunnen verwijderen zodat de applicatie niet meer gekoppeld is met mijn account.”

1.5.2 Uitleg

Een device moet gekoppeld zijn om gebruik te maken van de applicatie. Dit gebeurd doormiddel van het device id. Zodra de applicatie niet meer gebruikt wordt of het device is niet meer in bezit van de medewerker, zal het device ontkoppeld moeten worden van het account. Dit word via de webservice gedaan.

1.6 Device toevoegen.

1.6.1 User story titel

“Als gebruiker wil ik mijn device kunnen toevoegen zodat het device gekoppeld is met mijn account.”

1.6.2 Uitleg

Een device moet gekoppeld zijn om gebruik te maken van de applicatie. Dit gebeurd doormiddel van het device id. Het device id kan opgehaald worden in de applicatie zelf. Met dit device id is de applicatie gekoppeld aan het Coress account. Ieder device kan een ander wachtwoord hebben.

Bijlage E – Technisch ontwerp

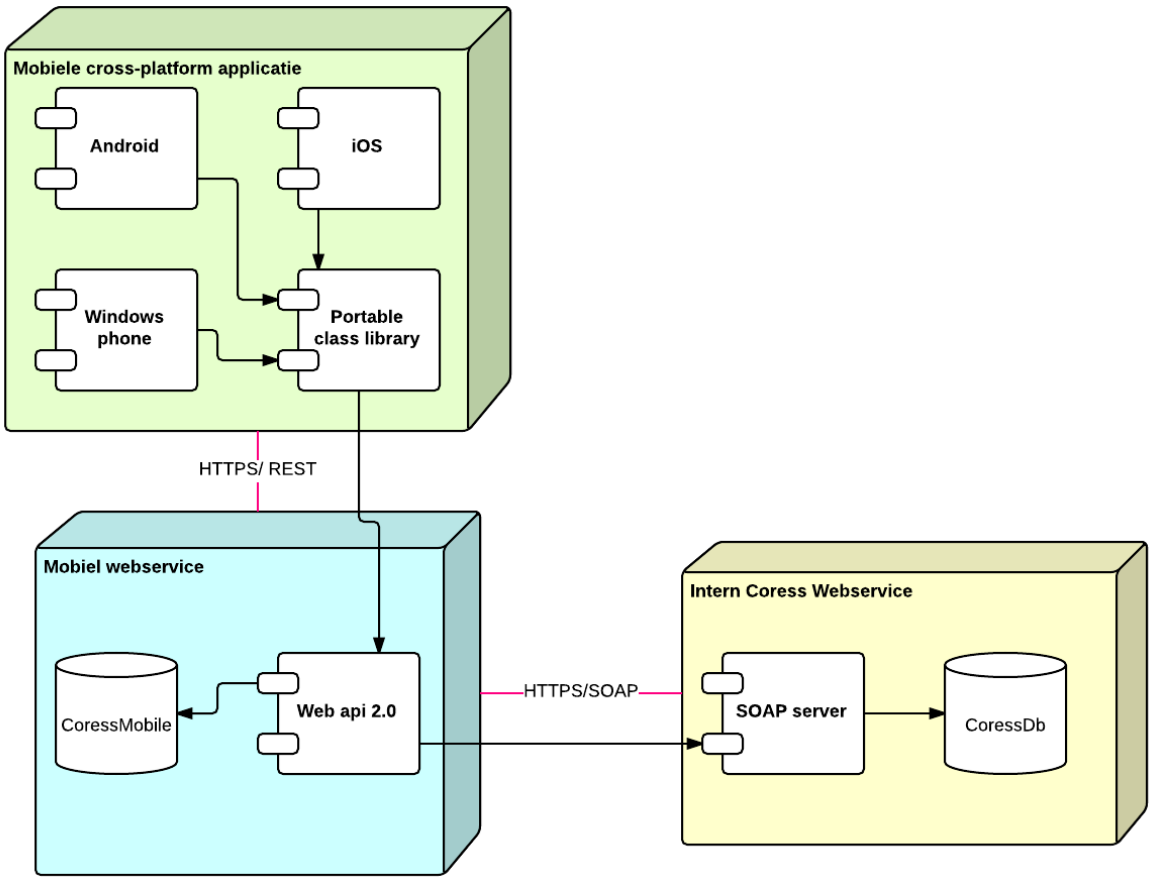
Mobiele Coress

Allard Soeters

Auteur	Allard Soeters
Studentnummer	11114762
Studie	Informatica
School	Haagse Hogeschool
Eerste examinator:	G.A. Mijnarends
Tweede examinator:	H.G.J. Bechet
Opdrachtgever:	R. Flohil
Afstudeerbegeleidster:	P. Hijl
Technisch Begeleider:	E. van Alebeek
Business Unit manager:	H. Brands
Plaats	Zoetermeer
Datum	24-08-2015
Bedrijf	Info Support B.V.

1. Architectuur

De architectuur ziet er als onderstaand schema uit.



Figuur 1 Architectuur ontwerp

De intern Coress webservice bestaat al en zal niet binnen de scope van dit project vallen. De mobiel webservice en de mobiele cross-platform applicatie zijn in de scope van dit project. De verbinding van de mobiel webservice naar de interne coress webservice wordt al gebruikt door de web front-end van de huidige situatie, zie bron “Huidig functioneel/technisch ontwerp”. Deze verbinding staat in een project genaamd proxy in de klasse `ServiceProxy` in de namespace `InfoSupport.Coress.WebFrontEnd`. Deze klasse wordt herbruikt omdat alle methodes die de webfrontend gebruikt ook gebruikt worden in de mobiel webservice.

De drie applicaties staan in verschillende zones van het netwerk van Info Support.

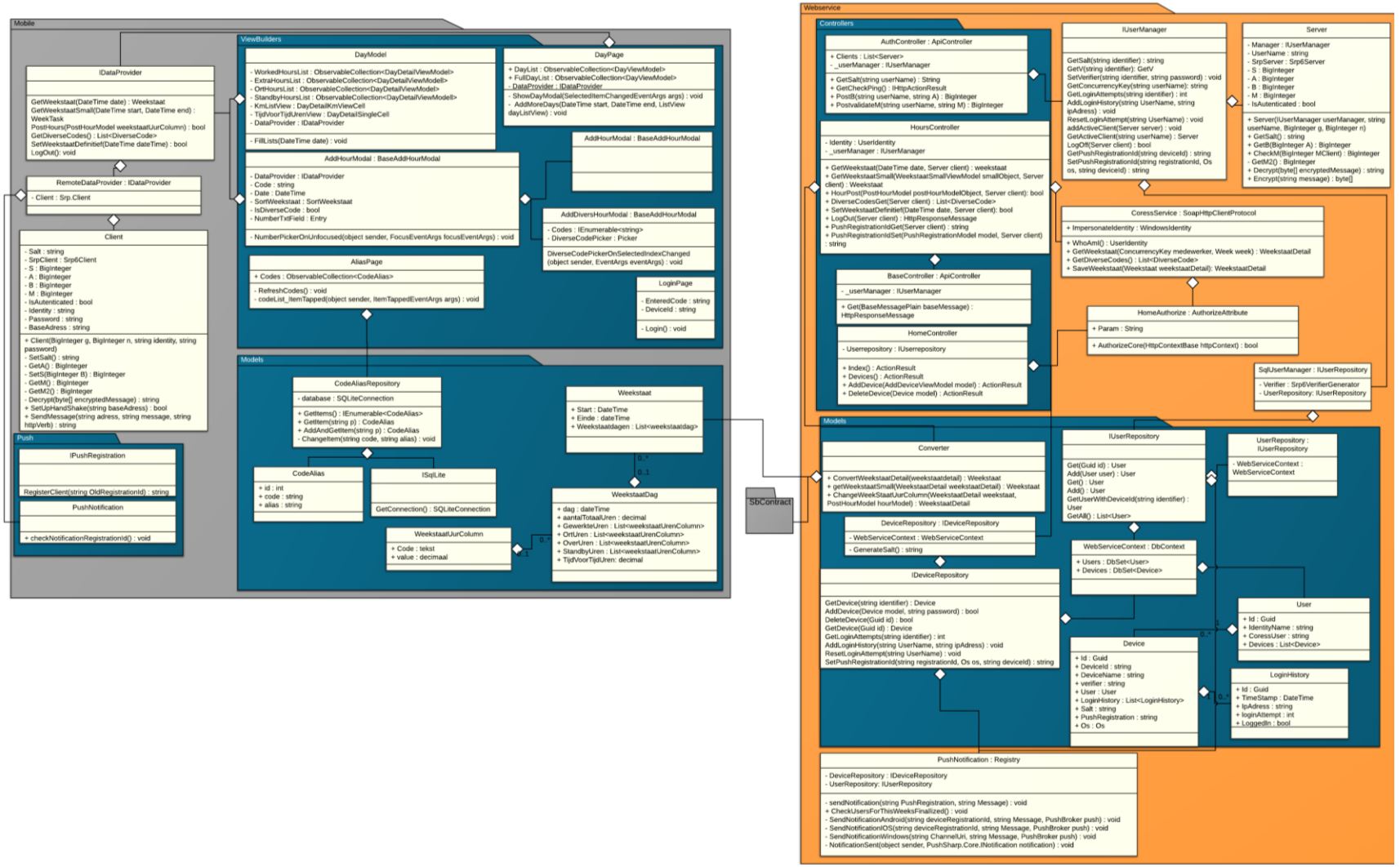
De interne Coress Webservice staat in het interne netwerk van Info support.

Er is voor gekozen om de mobiele webservice in de demilitarized zone(DMZ) van Info Support te zetten omdat deze zone van buiten het netwerk bereikt kan worden. Maar de webservice kan zelf ook de applicaties benaderen die intern zijn. Hierdoor wordt ervoor gezorgd dat de interne Coress

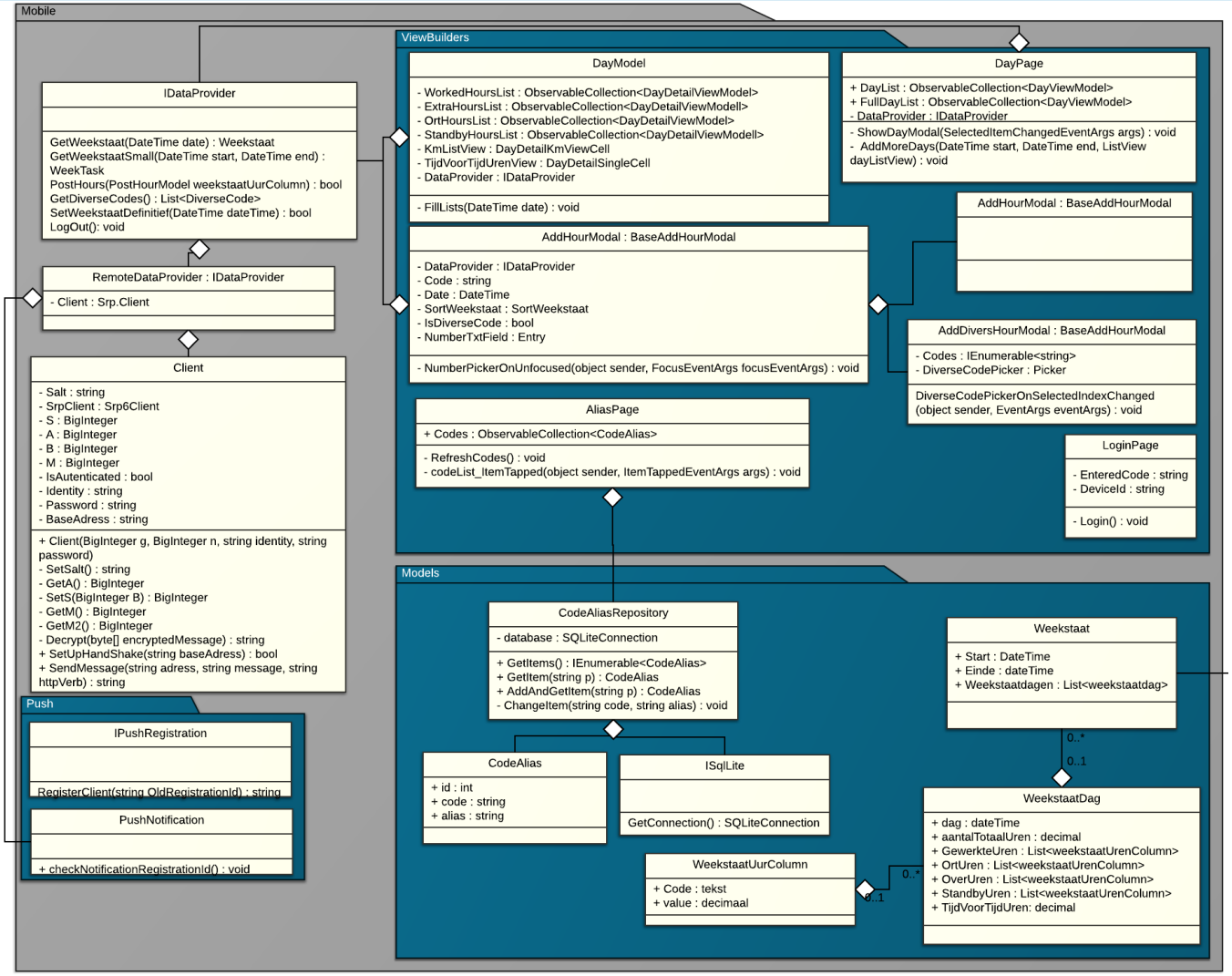
webservice niet blootgesteld wordt aan de buitenwereld maar dat de mobiele applicaties wel gegevens hieruit kunnen halen.

De cross-platform applicatie staat op de telefoons van de medwerkers. Deze medewerkers zijn vaak verbonden met een GSM netwerk of een netwerk dat niet van Info support is. Hierdoor zal deze applicatie beschouwd worden als een externe applicatie

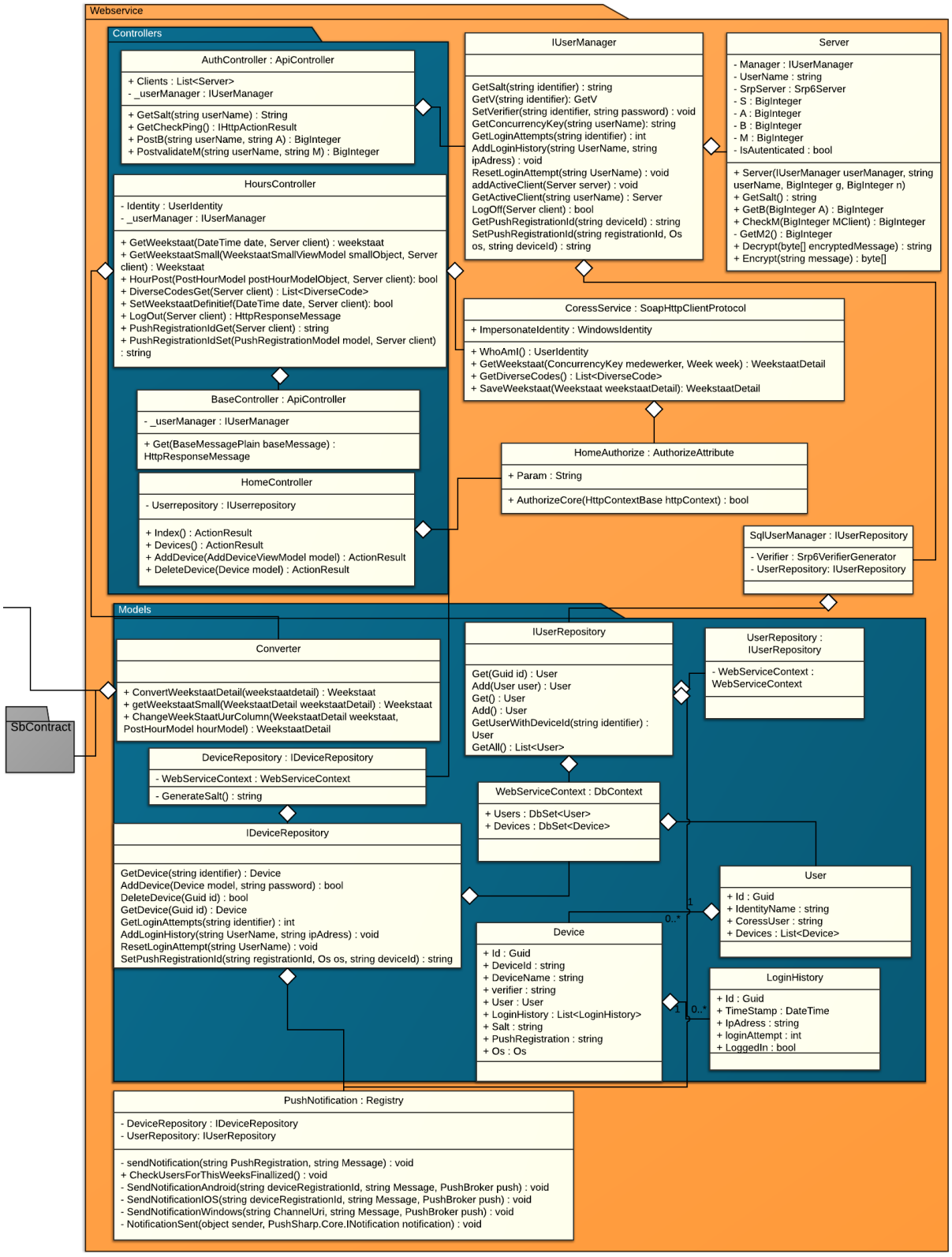
2. Klassendiagram



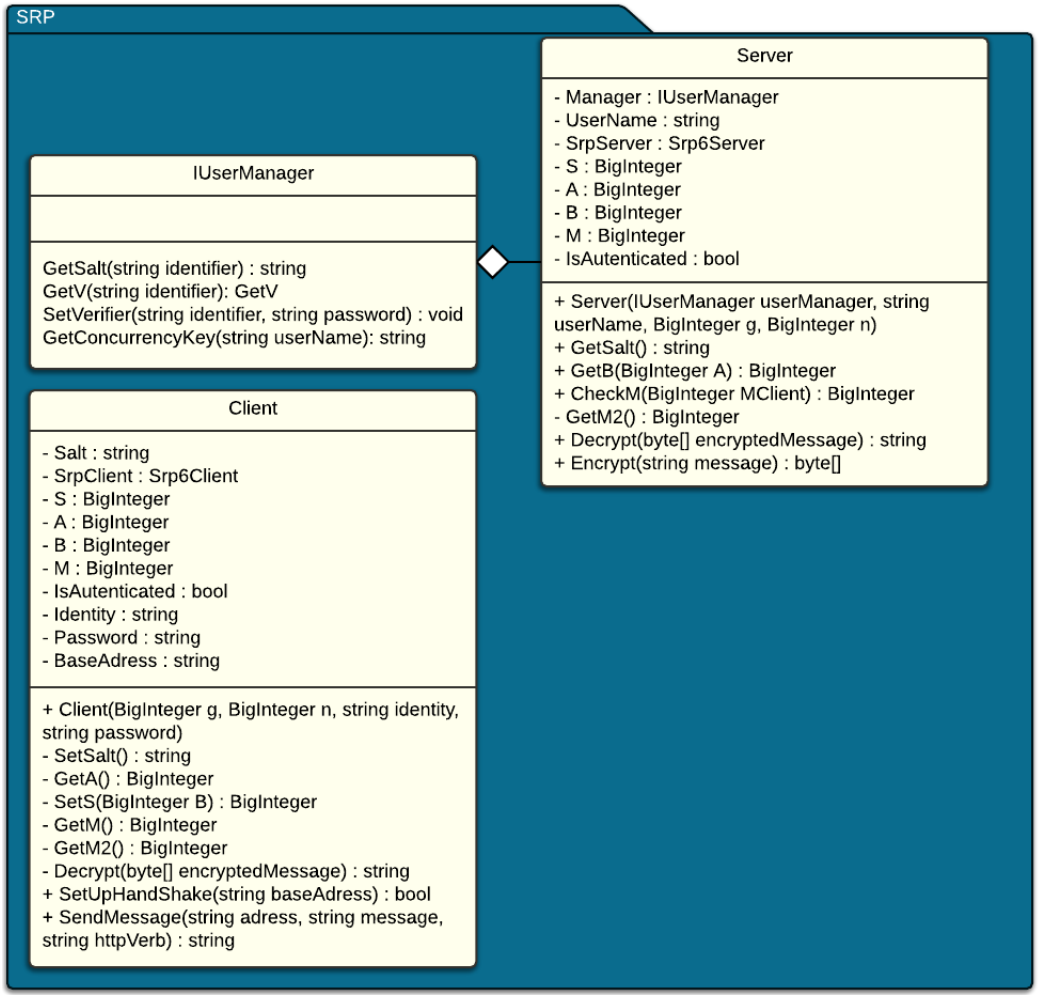
2.1 X-Platform applicatie



2.2 Webservice



2.3 SRP



SRP is het protocol wat gebruikt wordt om de medewerkers te authenticeren. Dit protocol is opgedeeld in twee delen een client deel en een server gedeelte.

2.3.1 Gebruikersdata

Het server gedeelte wordt gebruikt in de webservice. De server moet de gebruikers bijhouden en de gegevens uit een data bron halen. Dit gebeurt via het IUserManager interface. Dit is een interface geworden omdat er dan makkelijk een andere databron gebruikt kan worden. In de webservice wordt gebruik gemaakt van een SQL implementatie en in het SRP test project wordt gebruikt van een in memory implementatie.

Het client gedeelte wordt gebruikt bij de cross-platform applicatie. Deze klasse kan met alle data die de gebruiker ingeeft de controle uitvoeren of de gebruiker de juiste pincode heeft ingegeven.

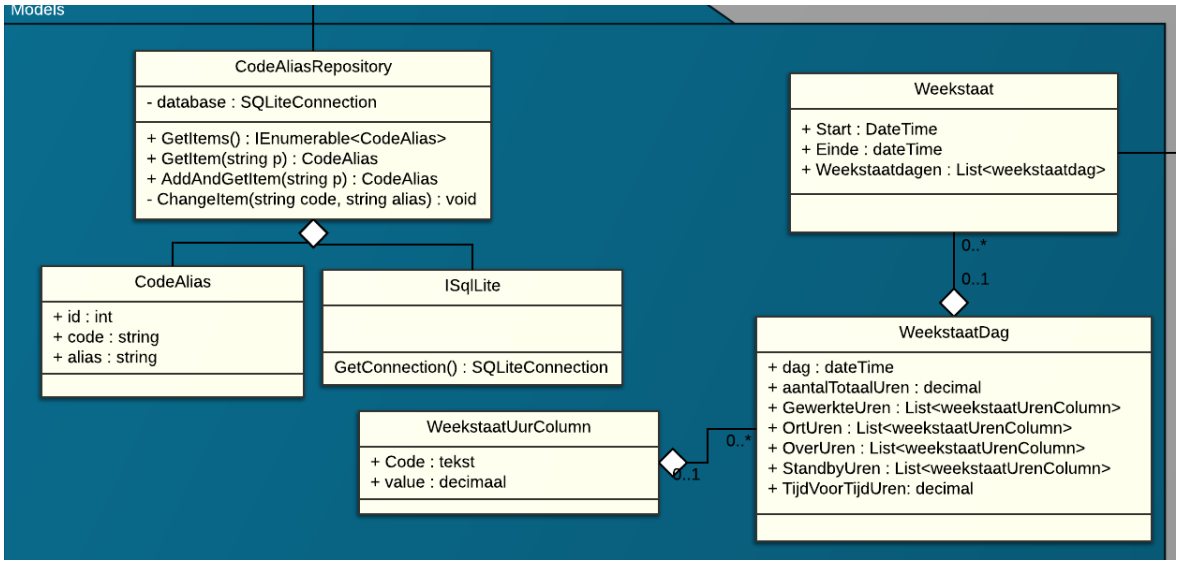
De BigInteger B en BigInteger N zijn alle twee hetzelfde. Deze zijn essentieel bij het uitvoeren van de authenticatie.

2.3.2 Library

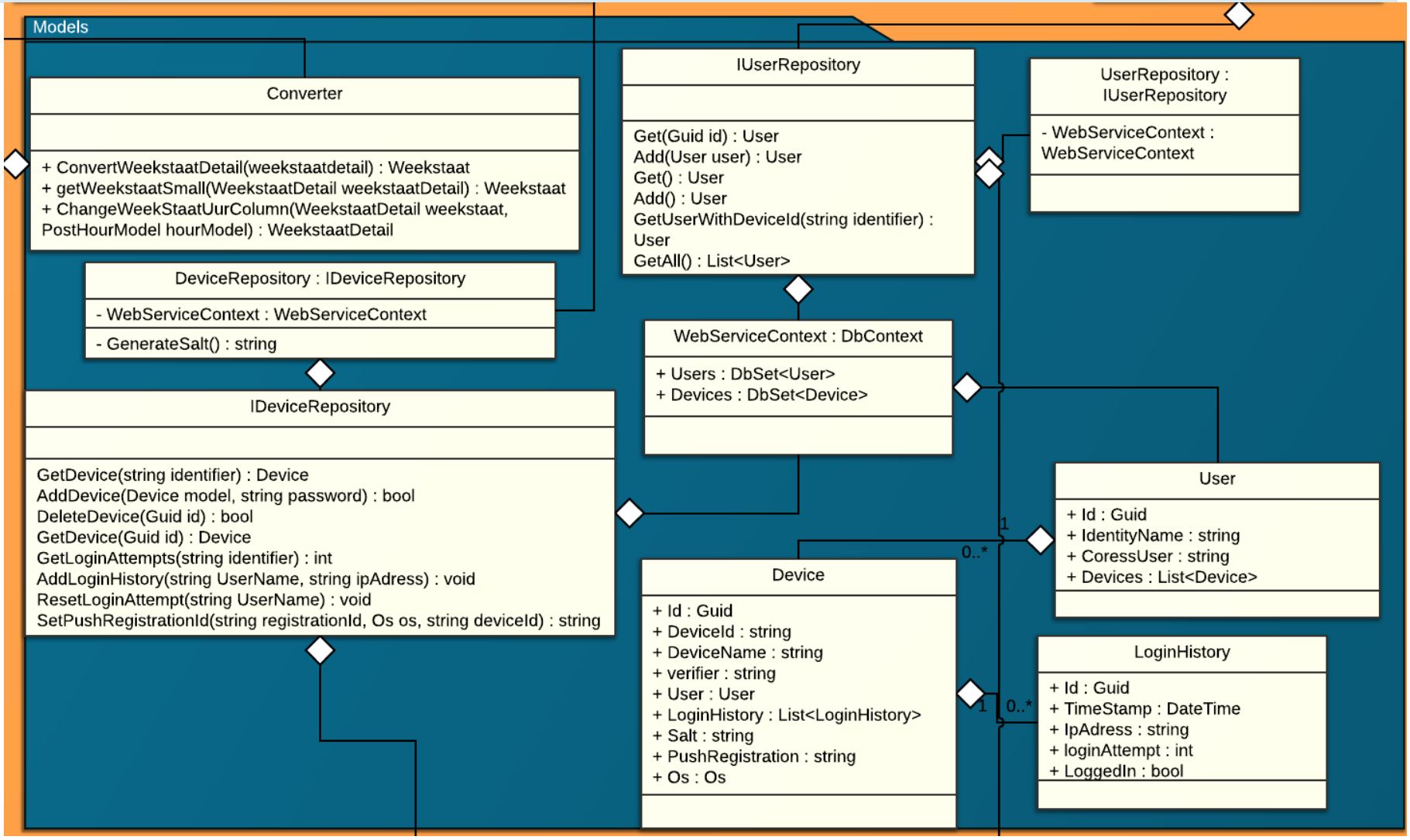
De berekeningen worden door een SRP6 implementatie gedaan wat in de Bouncy castle library zit. Bouncy castle staat bekend als crypto library voor java en C#. Het is open source waardoor de berekeningen gecontroleerd kunnen worden. Dit is door Allard Soeters gedaan.

Bij het verzenden van informatie van de webservice naar de x-platform applicatie of andersom zal het bericht geencrypt worden. De encryptie zal doormiddel van een AES-GCM encryptie gaan. Dit wordt via het bouncycastle library gedaan.

2.4 X-platform model



2.5 Webservice model



In het model van het model wordt de data opgehaald en de data wordt geconverteerd. In dit klasse diagram vinden drie verschillende gebeurtenissen plaats. Namelijk de database connectie, het repository pattern wordt gebruikt en er is een converter klasse.

2.5.1 Database

De database connectie maakt gebuik van de DbContext van het Entity framework. Het Entity Framework wordt via code first geïmplementeerd. User, Device en LoginHistory zijn klasses die omgezet worden naar database tabellen.

Het repository pattern wordt hierbij gebruikt omdat de business logica goed gescheiden moet worden van de database connectie. Dit is omdat er dan makkelijker de business logica getest kan worden maar ook omdat dit een geordend project oplevert wat beter uitbreidbaar is.

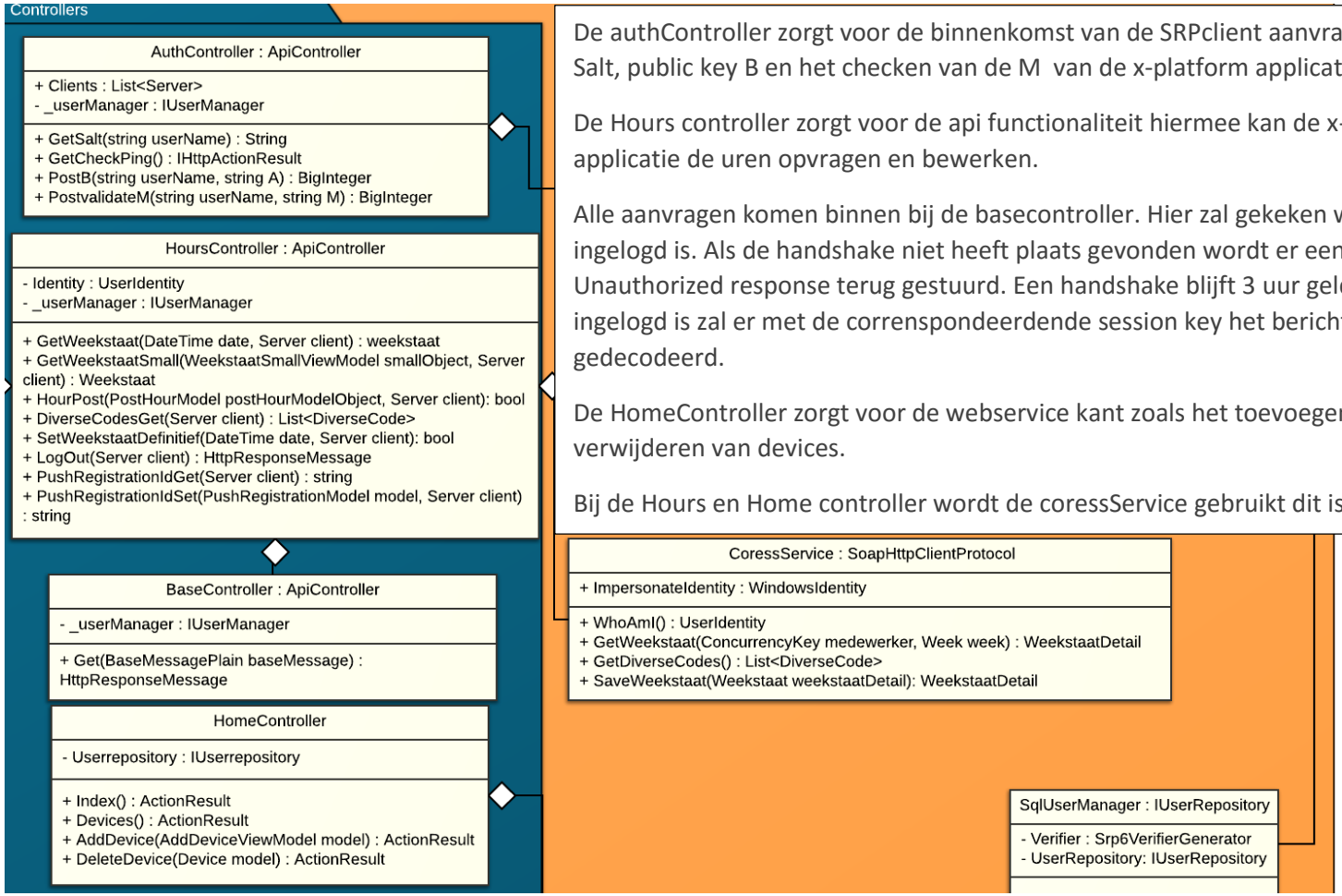
2.5.2 Converter

De converter klasse zorgt ervoor dat de data vanaf de interne webservice omgezet wordt zodat de x-platform applicatie gebruik kan worden hieronder staat in een schema hoe deze conversie gaat.

Interne coress	X-platform Applicatie	Opmerkingen
WeekstaatDetail	**Weekstaat**	
Medewerker Medewerker	*DateTime* start	
Week Week	*DateTime* Eind	
decimal ContractUren	*List<weekstaatDag>* weekstaatdagen	
WeekstaatDetailStatus StatusMedewerker		
WeekstaatDetailStatus StatusSsm	**WeekstaatDag**	
WeekstaatDetailStatus StatusFiatteur	*DateTime* date	
WeekstaatKmRow Kilometers	*bool* weekstaatPlichtig	Uit Medewerker
List<WeekstaatUrenRow> GewerkteUren	*WeekstaatDetailStatus* StatusMedewerker	
List<WeekstaatUrenRow> Overuren	*List<WeekstaatUurColumn>* Kilometers	
WeekstaatUrenRow TijdVoorTijdUren	*List<WeekstaatUurColumn>* GewerkteUren	
List<WeekstaatUrenRow> OrtUren	*List<WeekstaatUurColumn>* OverUren	
List<WeekstaatUrenRow> StandbyUren	*List<WeekstaatUurColumn>* OrtUren	
IList<KeyValuePair<WeekstaatRowType, WeekstaatUrenRow>> AllUren	*List<WeekstaatUurColumn>* StandbyUren	
string Opmerkingen		
DateTime? LastModified	**WeekstaatUurColumn**	
	string Code	
	bool IsDiverseCode	
WeekstaatUrenRow	*decimal* vlaue	
decimal[] _values		
string Code		
bool IsEmpty		
bool IsDiverseCode		

De interne coress levert per week alle gegevens. Voor de mobiele app is er alleen per dag informatie nodig. Als de dag opgevraagd wordt in de x-platform app zal er aan de interne coress de hele week gevraagd worden. Deze week wordt dan geconverteerd naar 1 dag met de juiste klasses.

2.6 Webservice controllers



De authController zorgt voor de binnenkomst van de SRPclient aanvragen zoals de Salt, public key B en het checken van de M van de x-platform applicatie.

De Hours controller zorgt voor de api functionaliteit hiermee kan de x-platform applicatie de uren opvragen en bewerken.

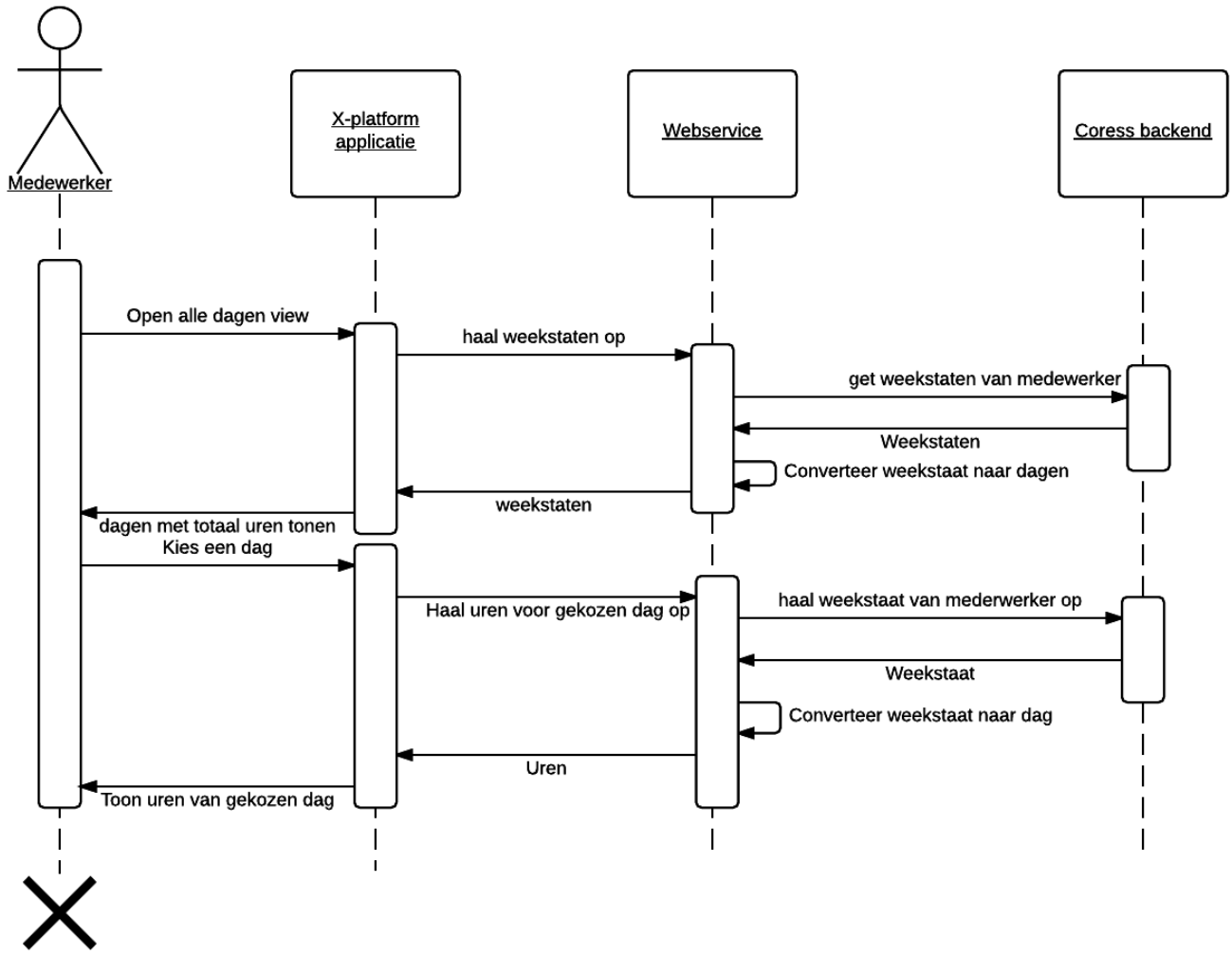
Alle aanvragen komen binnen bij de basecontroller. Hier zal gekeken worden of er ingelogd is. Als de handshake niet heeft plaats gevonden wordt er een Unauthorized response terug gestuurd. Een handshake blijft 3 uur geldig. Als er ingelogd is zal er met de correnspondeerdende session key het bericht worden gedecodeerd.

De HomeController zorgt voor de webservice kant zoals het toevoegen en verwijderen van devices.

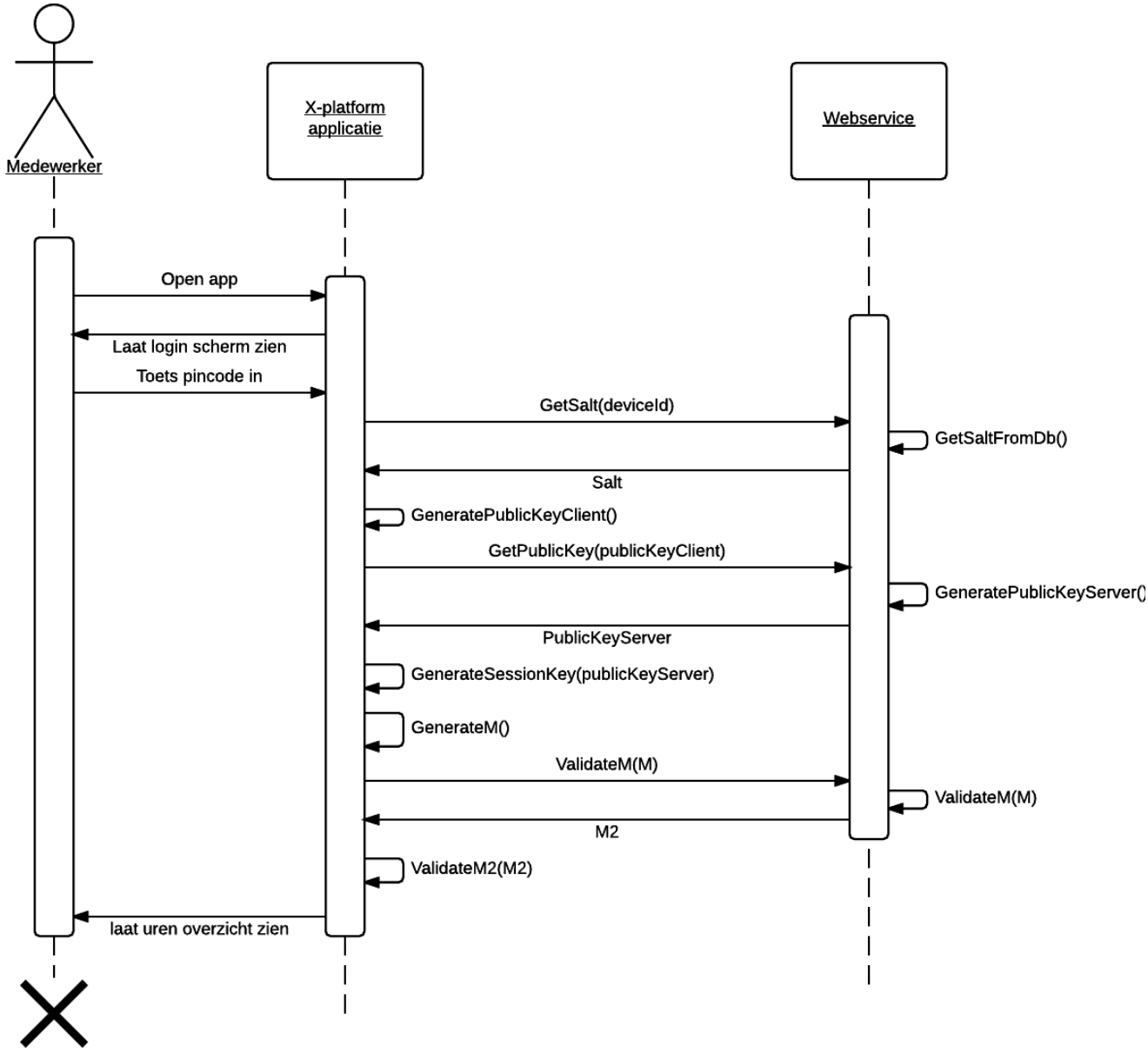
Bij de Hours en Home controller wordt de coressService gebruikt dit is de service

3. Sequence diagrammen

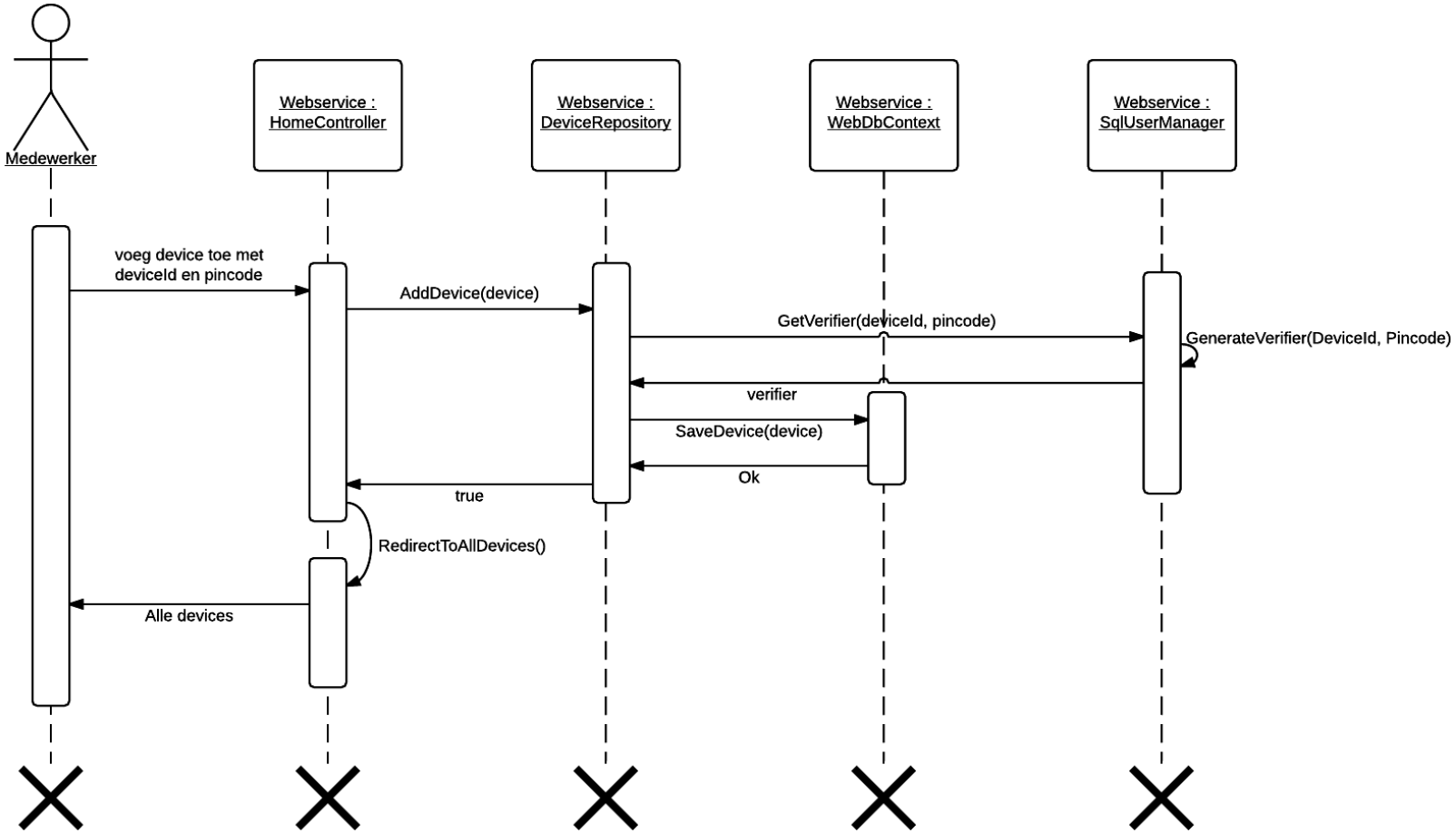
1.1 ingevulde uren bekijken



1.2 Inloggen


Medewerker
X-platform applicatie
Webservice
Open app
Laat login scherm zien
Toets pincode in
GetSalt(deviceId)
GetSaltFromDb()
Salt
GeneratePublicKeyClient()
GetPublicKey(publicKeyClient)
GeneratePublicKeyServer()
PublicKeyServer
GenerateSessionKey(publicKeyServer)
GenerateM()
ValidateM(M)
ValidateM(M)
M2
ValidateM2(M2)
laat uren overzicht zien

1.3 Device toevoegen sequence diagram



Bijlage F – Security

Mobile Coress

Allard Soeters

Auteur	Allard Soeters
Studentnummer	11114762
Studie	Informatica
School	Haagse Hogeschool
Eerste examinator:	G.A. Mijnarends
Tweede examinator:	H.G.J. Bechet
Opdrachtgever:	R. Flohil
Afstudeerbegeleidster:	P. Hijl
Technisch Begeleider:	E. van Alebeek
Business Unit manager:	H. Brands
Plaats	Zoetermeer
Datum	24-08-2015
Bedrijf	Info Support B.V.

Inhoudsopgave

1. Analyse

1.1 Assets (valuables)

De beveiliging van software is belangrijk omdat er assets gestolen kunnen worden. Helaas zijn de assets niet alleen de software en de data van de software. Als er geen goede beveiliging is kunnen de assets ook buiten de software liggen bijvoorbeeld de systemen van de gebruikers die op het zelfde netwerk zitten als de software. Daarom moet er goed bepaald worden wat er gestolen kan worden. Een hacker zal er alles aan doen om deze assets te stelen als het de moeite waard is.

De data van Coress is het eerste wat de hacker tegen kan komen. In deze data staan gegevens van klanten en projecten. Daarbij horen het aantal uren dat een medewerker aan een project heeft gewerkt. De klantgegevens en hoeveel uur er besteed word aan welke projecten is erg interessant voor de concurrenten. Daarnaast zou het niet goed zijn voor de reputatie van Info Support als de klant gegevens door iedereen te bekijken zijn.

Als een hacker buiten de software gaat kijken en het netwerk ingaat zal hij andere applicaties tegenkomen. De mobiele webservice staat in de DMZ. In de DMZ staat de publieke website en andere publieke applicaties. Hij kan de informatie aan kunnen passen van deze publieke applicaties. Als de hacker van de DMZ naar het interne netwerk kan komen zal hij alle applicaties en alle assets van heel Info Support kunnen vinden. Dit valt buiten de scope van het project en hier zal dan ook geen aandacht aan besteed worden.

De scope van dit project zal de data van Coress zijn die verkregen kan worden via de mobiele applicatie en de mobiele webservice. De connectie van de mobiele webservice en de interne Coress webservice valt ook buiten de scope omdat hier nog geen manier voor gevonden is om deze te authentiseren.

1.2 Threats

Een threat kan van iedere hoek komen. Een concurrent die een hacker inhuurt om gegevens te stelen. Maar ook een scrip kid die een aantal pogingen doet om wat informatie te stelen. Een script kid heeft meestal geen baat bij de informatie die hij steelt maar deze probeert hij dan te verkopen aan de hoogste bieder.

Er bestaan ook ethische hackers en deze zullen je systeem proberen te hacken om deze bij de eigenaar aan het licht te brengen. De hacker probeert het internet veiliger te maken. Een ethische hacker is strafbaar zodra hij begint met hacken. Omdat hij niet misbruik maakt van de vulnerabilities die hij vind zijn deze hackers handig voor de eigenaar. Daarom kan de eigenaar een responsible disclosure opstellen waarin staat dat iedereen bepaalde hacks mag uitvoeren om lekken aan te tonen. Zodra er misbruik gemaakt wordt van een vulnerability zal de eigenaar meestal aangifte doen. Omdat niet alle applicaties een responsible disclosure hebben zijn er niet veel ethische hackers. Het levert een ethische hacker ook niks op, steeds meer bedrijven geven een beloning zodra er een vulnerability is gevonden om deze hackers aan te trekken.

1.3 Vulnerabilities

Een mobiele applicatie die zijn gegevens ophaalt via het internet zal nooit 100% veilig zijn. Er zullen altijd ergens gaten zijn en deze kunnen altijd gevonden worden. Dit zal zo zijn zolang het internet en hackers bestaan. Hierdoor zullen alle gaten die er bekend zijn om deze te dichten. Bij de mobiele applicatie zijn dat de volgende onderdelen:

1. Authenticatie
2. Local storage
3. Jailbreak
4. Authorization
5. session hijacking

1.4 Risks

Het risico dat een vulnerability gehackt wordt kan berekend worden door de waarschijnlijkheid maal de impact die de vulnerability heeft. Het kan zijn dat er een grote kans is dat de vulnerability benut wordt maar dat er nauwlijks impact heeft, dit betekent dat het risico niet hoog is.
Hieronder in een tabel de waarschijnlijkheid en de impact voor de verschillende vulnerabilities.

	Waarschijnlijkheid	Impact	Totaal	Rank
Authentication	9	8	72	1
Authorisation	7	10	70	2
Session hijacking	8	5	40	2
Injection	3	8	24	4
Jailbreak	9	2	18	5
Local storage	9	1	9	6

1.4.1 Authentication

1.4.1.1 Waarschijnlijkheid:

Wat een hacker wilt is het zo onopvallend mogelijk het hele systeem ontdekken en stelen. Als de authenticatie op een manier omzeild kan worden zal dit doen sneller bereikt worden. Het is een grote kans dat de hacker als eerst zich op de authenticatie richt.

1.4.1.2 Impact:

Als de authenticatie omzeild kan worden dan kan de hacker zich voordoen als iemand anders. En het hele systeem kan bekijken.

Het kan voorkomen dat de hacker wel de authenticatie kan omzeilen maar zich niet voor kan doen als iemand anders. Hij moet hiervoor dan nog de authorization omzeilen. Dit zorgt ervoor dat er geen 10 punten gegeven zijn maar 8.

1.4.2 Authorisation

1.4.2.1 Waarschijnlijkheid:

Om de authorisation te omzeilen moet eerst de authenticatie worden omzeild.

1.4.2.2 Impact:

Als de authorisation omzeild wordt in combinatie met de authentication kan de hacker het hele systeem in.

1.4.3 Session hijacking

1.4.3.1 Waarschijnlijkheid:

Als er een sessie wordt overgenomen kan de hacker voor de duur van de sessie zich voordoen als de overgenomen persoon. Dit is natuurlijk voor de hacker fijn want het in onopvallend.

1.4.3.2 Impact:

De impact is niet heel groot omdat de hacker maar een bepaalde tijd heeft om gebruik te maken van de hack, totdat de sessie verloopt.

1.4.4 Jailbreak

1.4.4.1 Waarschijnlijkheid:

Een jailbreak(iOs, windows phone) of je device rooten(Android) is simpel te doen en vergt niet veel moeite. Bij nieuwere devices zal dit nog niet meteen werken. Met een jailbreak kan de source code ontdekt worden.

1.4.4.2 Impact:

De impact indien de applicatie goed in elkaar zit, zal niet groot zijn. Als de source code ontdekt word kan de hacker alleen de client side alle mogelijke hacks makkelijker vinden.

1.4.5 Injection

1.4.5.1 Waarschijnlijkheid:

Injection is een manier om gegevens in de database te manipuleren. Het is een lastigere manier om gegevens te ontedekken. En Lastig om alles in het systeem te doen.

1.4.5.2 Impact:

Als er injection gebruikt wordt kan de hacker alles met de data van de applicatie doen. Met SQL injection kan de data alleen bekeken worden. Andere injections zouden ook kunnen schrijven maar deze komen minder vaak voor.

1.4.6 Local storage

1.4.6.1 Waarschijnlijkheid:

Als er data lokaal opgeslagen opgeslagen wordt kan deze makkelijk worden opgehaald. Daar is meestal geen jailbreak of iets dergelijks voor nodig. De kans is groot dat de lokaal opgeslagen data wordt gestolen.

1.4.6.2 Impact:

Omdat er lokaal niet veel opgeslagen is en met deze gegevens niks mee gedaan kan worden is de impact nihil.

1.5 Measurements

Voor de volgende Vulnerabilities zijn er voor dit project maaregelen genomen:

1. Authenticatie
2. Session hijacking

Voor de andere vulnerabilities zal indien nodig nog maaregelen genomen moeten worden. Een aantal vulnerabilities zijn al aangepakt maar worden niet besproken, zoals SQL injection. De ORM mapper, entity framework zorgt er al voor dat er geen SQL injection meer kan plaatsvinden.

De mobiele applacatie kent maar één rol en dat is de rol van een weekstaat plichtige medewerker. Er wordt geen autorisatie uitgevoerd door de mobiele applicatie of webservice. De autorisatie vind plaats in de interne Coress webservice.

1.5.1 Authenticatie

Het SRP-6a protocol is gekozen om ervoor te zorgen dat er authenticatie goed uitgevoerd wordt en dat dit gat gedicht is. Voor de keuze en uitwerking van SRP zie bijlage.

1.5.2 Session hijacking

Om ervoor te zorgen dat de verbinding veilig is en dat er geen Man in the middle de sessie over kan nemen en gegevens kan stelen is er voor twee methodes gekozen. Er wordt gebruik gemaakt van SSL, dit zorgt ervoor dat er een veilige geëncrypte verbinding opgezet wordt tussen de webservice en de mobiele applicatie.

De tweede methode is een extra encryptie laag tussen de code van de webservice tot de code van de mobiele applicatie. Deze berichten worden geëncrypt met de sessie sleutel die verkregen wordt door het SRP protcol. Dit zorgt ervoor als er bug in SSL gevonden is dat de hackers door nog een laag moeten hacken.

2. Tests

2.1 Authenticatie

Uit de owasp testing guide v4 zijn de volgende test gekomen die gebruikt kunnen worden:
https://www.owasp.org/images/5/52/OWASP_Testing_Guide_v4.pdf

1. Testing for Credentials Transported over an Encrypted Channel (OTG-AUTHN-001)
2. Testing for Weak lock out mechanism (OTG-AUTHN-003)
3. Testing for Bypassing Authentication Schema (OTG-AUTHN-004)

2.1.1 OTG-AUTHN-001

2.1.1.1 Uitvoering

In deze test wordt er gekeken of de username en password gemakkelijk onderschept kan worden. Dit kan gebeuren als de verbinding unencrypted is. Er zijn twee verschillende soorten testen; blackbox en de whitebox test.

In de blackbox test zal er gebruik worden gemaakt van een web proxy. Met deze proxy kan de data onderschept worden die de applicatie stuurt naar de webservice en terug. Bij iedere request mbt inloggen kan er dan gekeken worden of er gegevens onderschept kunnen worden.

De web proxy(WebScarab) zal iedere request ontvangen zodra de applicatie eentje stuurt. Deze request zal geïnspecteerd worden. Als er iets van een username of password voorkomt zal dit genoteerd worden. Als een username voorkomt zal dit niet een groot probleem zijn maar als een password erin voorkomt zal dit een groot probleem zijn.

In de whitebox test zal er gekeken worden in de code. In de code za er gecontroleerd worden of er encryptie wordt toegepast. En of de gebruikersnaam en wachtwoord uberhaupt over de verbinding gestuurd wordt. Of dat er op een andere manier authenticatie wordt uitgevoerd.

2.1.1.2 Resultaten

Blackbox:
Er vind encryptie plaats. Er is niks gevonden.

Whitebox:
Er vind een dubbele encryptie laag plaats, als de SSL laag gehackt wordt dan zal er altijd nog een tweede laag. In deze laag zit er wel het device id van het apparaat. Met het device id kan niks kwaads meegedaan worden, het device id kan je vergelijken met een emailadres waar je mee moete inloggen.
Als de tweede laag gedecrypt wordt zal er geen password gevonden worden. Het authenticatie protocol wat gebruikt wordt is SRP en met dit protcol wordt geen password verstuurd. Zie voor meer uitleg over SRP in de bijlage.

2.1.2 OTG-AUTHN-003

2.1.2.1 Measurement vooraf

Er is een lock-out mechanisme gebouwd in de webservice. Als een device id 10 keer foutief heeft aangemeld dan kan hij 20 minuten het niet nog een keer proberen. Dit wordt allemaal server side geregeld.

2.1.2.2 Uitvoering

Voor het testen of er een brute-force of dictionary attack uitgevoerd kan worden zal er gekeken worden of er een lock-out mechanisme inzit.
Dit zal op de volgende manier gaan.

1. Voer het verkeerde wachtwoord in voor 10 keer.
2. Voer het juiste wachtwoord in om te kijken of er ingelogd kan worden
3. Zodra er niet ingelogd kan worden betekent dit dat er gewacht moet worden
4. Wacht 20 minuten.
5. Log in met het juiste wachtwoord om te laten zien dat er na 20 minuten weer ingelogd kan worden.

2.1.2.3 Resultaten

Stap	Gelukt
1	Ja
2	Ja
3	Ja
4	Ja
5	Ja

2.1.3 OTG-AUTHN-004

2.1.3.1 Uitvoering

Om te testen of er een manier is om de authenticatie te omzeilen of te spoofen dat er een authenticatie plaats gevonden heeft zijn er verschillende manieren.

1. Direct page request (forced browsing)

Door verschillende resources op te vragen bij een webserver kan het voorkomen dat er niet gecheckt word of de gebruiker geauthentiseerd is. De developer zou dan denken dat de gebruiker/hacker altijd dezelfde sequence naloopt.

2. Parameter modification

Als er in de request een parameter met betrekking tot de authenticatie zit kan je deze misschien veranderen zodat de webservice denkt dat de request door een geauthentiseerde gebruiker is gestuurd.

Door in de request parameters te zoeken naar een

3. Session ID prediction

Een session id wordt soms meegestuurd om de juiste gegevens op te halen. De sessie id kan ook vervalst worden. Als er een session id wordt geraden van een geauthentiseerde gebruiker dan kan deze sessie over genomen worden.

Door in de request parameters, cache of cookies te kijken kan er misschien een sessie-id gevonden worden. Als dit het geval is kan deze sessie-id veranderd worden door gebruik te maken van de web proxy.

4. Injection

Bij een request kan er een parameter mee worden gestuurd waarbij in code deze naar de sql server wordt gestuurd. En de code verwacht misschien een boolean terug. Zodra er dan in deze parameter een sql code staat wat een true oplevert zal er ook een true terug gegeven worden. Hierdoor kan de webservice denken dat het een juist wachtwoord was.

2.1.3.2 Resultaten

Direct page request:

Geteste url	Moet ingelogd zijn
https://172.1.1.1:5000/api/hours/	Ja
https://172.1.1.1:5000/api/Home/Index/	Ja
https://172.1.1.1:5000/api/Home/LoginHistories/	Ja
https://172.1.1.1:5000/api/Home/ [POST AddDeviceViewModel]	Ja
https://172.1.1.1:5000/api/Home/ [POST Device]	Ja
https://172.1.1.1:5000/api/Auth/{userName}/Salt/	Nee
https://172.1.1.1:5000/api/Auth/checkPing/	Nee
https://172.1.1.1:5000/api/Auth/{userName}/B/	Alleen als salt nog niet is opgeroepen
https://172.1.1.1:5000/api/Auth/{userName}/validateM/	Alleen als salt en B nog niet zijn aangeroepen

Parameter modification:
De parameters die gestuurd worden zijn encrypted. Die encryptie kan alleen uitgevoerd worden als er ingelogd is. Als er ongeencrypte parameters gestuurd worden wordt er een unauthorized bericht terug gestuurd.

Session ID prediction:
Er is geen sessie Id gevonden. Wel is er een device id gevonden maar als deze veranderd wordt kan er niks gedaan worden.

Injection:
Zie Parameter modification.

2.2 Session hijacking

Om de verbinding te testen moet er een man in the middle attack uitgevoerd worden. Als de attack slaagt dan moeten er andere measurements genomen worden.

Door middel van een man in the middle attack kunnen alle gegevens die de client naar de server stuurt en andersom onderscheppen. Hierdoor kan het sessie id of andere waardes die aan de sessie gekoppeld zit gestolen worden. Met deze waarde(s) kan de client na gedaan kan worden.

Via de onderstaande tets zal er nagegaan worden of er een sessie over genomen kan worden

1. Testing for Session Management Schema (OTG-SESS-001)
2. Testing for Session Fixation (OTG-SESS-003)

2.2.1 OTG-SESS-001

2.2.1.1 Uitvoering

In deze test wordt er getest of er een sessie token gecreëerd wordt en als deze gecreëerd wordt of deze dan makkelijk te voorspellen zijn. Als sessie token makkelijk te voorspellen is dan kan er een sessie gemakkelijk worden gekaapt.

Door te zoeken in de requests van de applicatie naar de server kan er gekeken worden of er een token bijgehouden wordt.

2.2.1.2 Resultaten

Er is geen sessie Id gevonden. Wel is er een device id gevonden maar als deze veranderd wordt kan er niks gedaan worden.

2.2.2 OTG-SESS-003

2.2.2.1 Uitvoering

In deze test wordt gekeken of de sessie token wel steeds hernieuwd wordt tijdens het opnieuw inloggen of wanneer de sessie is verlopen. Dit zorgt ervoor dat de sessie token ook echt gekoppeld zit aan een sessie.

Als de sessie token ontdekt is moet deze vergeleken worden met de sessie token die verkregen wordt als er opnieuw aangemeld is. Zodra er afgemeld is moet er een API call worden gemaakt met de laatste sessie token. Dit zou een 401 error code(not authorized) of iets dergelijks op moeten leveren.

2.2.2.2 Resultaten

Sessie code na inloggen	
Sessie code na uitloggen	Het deviceId
Sessie code na 2e x inloggen	Hetzelfde deviceId
Resultaten na inloggen	Hetzelfde deviceId

Het blijkt dat de deviceId steeds opnieuw gebruikt kan worden. Maar dat er wel ingelogd moet zijn. Na 3 uur te wachten en nog een keer proberen met dezelfde deviceid een api call te maken kan dit gewoon. Zodra er ingelogd is blijft de sessie actief.

Dit is op te lossen door na drie uur de sessie te resetten.
Na de oplossing waren de problemen opgelost.

3. Bronnen

1. https://www.owasp.org/index.php/REST_Assessment_Cheat_Sheet
2. http://www.infoq.com/news/2010/03/REST_security
3. https://www.owasp.org/index.php/REST_Security_Cheat_Sheet