KU LEUVEN

FACULTEIT INDUSTRIELE
INGENIEURSWETENSCHAPPEN

TECHNOLOGIECAMPUS GENT

Flexibele uitrol van updates voor Cordova applicaties

Jonas SEMEELEN

Promotor: Vincent Naessens

Co-promotor: Michiel Willocx

Masterproef ingediend tot het behalen van de graad van master of Science in de industriële wetenschappen: Elektronica - ICT

Academiejaar 2017 - 2018

Abstract

De meeste mobiele applicaties die vandaag in de omloop zijn, kunnen enkel geïnstalleerd en geüpdatet worden via een app store. Dit is onpraktisch voor zowel de gebruiker van de app als de ontwikkelaar ervan. Cordova (voormalig PhoneGap) applicaties bezitten de unieke eigenschap te kunnen updaten zonder tussenkomst van app store. Dit is mogelijk doordat de logica van een Cordova applicatie voornamenlijk bestaat uit platformonafhankelijke code zoals Javascript. Deze code kan probleemloos vervangen worden door nieuwe code m.b.v. een update plugin. Op deze manier wordt het mogelijk om een app dynamisch en transparant up te daten waarbij de gebruiker minimale overlast ondervindt.
Beeld je vervolgens een mobiele applicatie in waarbij de functionaliteit van de applicatie afhangt van de gebruiker. Indien er van gebruiker wordt veranderd, moet er nieuwe functionaliteit gedownload en geïnstalleerd worden. Dit proces zou heel moeizaam verlopen bij normale apps indien dit telkens via een app store dient te gebeuren. Cordova apps met hun update plugins zijn hiervoor reeds beter geschikt, maar schieten toch nog te kort op één vlak: de huidige update plugins zijn niet in staat om verschillende updates te verspreiden naar verschillende gebruikers.
Daarom werd er in dit eindwerk een systeem ontworpen rond de CodePush update plugin van Microsoft zodat het wel mogelijk is om verschillende updates te sturen naar verschillende gebruikers. De plugin onderging eerst een uitvoerige analyse om vast te stellen hoe de plugin werkt en om te bepalen welke extra componenten er nog zullen nodig zijn. Vervolgens werden deze ontbrekende componenten geïmplementeerd met als resultaat een veilig systeem dat flexibele updates kan verspreiden naar Cordova applicaties. Ten slotte hoort er bij dit eindwerk nog een prototype applicatie om het geheel te demonstreren.

Trefwoorden: cordova, plugin, update, codepush

Inhoudsopgave

Lijst van figuren

Hoofdstuk 1

Inleiding

1.1 Voorwoord

Als je aan mensen zou vragen hoe ze aan de apps komen op hun GSM, dan zullen de meesten waarschijnlijk antwoorden: "Via de app store."Dit antwoord is niet te verwonderen als je in beschouwing neemt dat Android-, en iOS-gebruikers sinds Maart 2017 uit maar liefst 2.8 respectievelijk 2.2 miljoen apps kunnen kiezen in de app stores.[1] Voor sommigen vormt de app store echter een vervelend opstakel. Zo kost het tijd en geld om een app of een update van een app in de store te plaatsen. Maar gelukkig bestaan er alternatieve manieren om applicaties te ontwikkelen.
De gebruikte technologie in dit eindwerk is Apache Cordova[2]. Dit is een framework om cross-platform mobiele applicaties te ontwikkelen. Bovendien zorgt dit framework ervoor dat de app store kan omzeild worden tijdens het updateproces door gebruik te maken van plugins die instaan voor automatische updates. Dit is een zeer aantrekkelijke eigenschap voor ontwikkelaars, maar ook voor bijvoorbeeld bedrijven die werken met zelfgemaakte applicaties. Het houdt namenlijk in dat updates meteen de gebruikers van de applicatie kunnen bereiken. Dit gemak is jammer genoeg gebonden aan een verhoogd veiligheidsrisico. Aangezien het updateproces niet meer gebeurt via de app store, wordt de update ook niet meer uitgevoerd binnen de veilige omgeving van de app store. De veiligheid hangt nu volledig af van de gebruikte update plugin.
Er bestaan reeds talloze update plugins voor Cordova applicaties, maar deze hebben allemaal beperkingen op vlak van functionaliteit en veiligheid. Daarom werd er in dit eindwerk een systeem ontworpen dat een uitbreiding vormt op de reeds bestaande update plugins. Het systeem is ontworpen met het oog op veiligheid zodat het kan worden gebruikt in situaties waar privacy belangrijk is, of in situaties waar het financiële aspect een grote rol speelt.
Het scenario waarbinnen het resultaat zal gedemonstreerd worden is een thuiszorgsituatie. De functionaliteit van het systeem zal aangetoond worden a.d.h.v. een Cordova applicatie die helpt bij de thuisverzorging van een patient. De mobiele applicatie zal toegang hebben tot een set sensoren die zich bevindt in het huis van de cliënt. Welke sensoren tot die set behoren en welke functies de app moet hebben, hangt af van de noden van de cliënt. Bijgevolg zullen er meerdere versies van de app beschikbaar zijn, elk met een set unieke functies. Dit houdt in dat niet iedere gebruiker dezelfde versie van de app zal hebben. Daarenboven moet er ook voor gezorgd worden dat de juiste updates naar de juiste gebruikers worden gestuurd. Huidige plugins kunnen hieraan niet voldoen en daarom zal er een oplossing voor dit probleem worden geconstrueerd in dit eindwerk.

[1] https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/
[2] https://cordova.apache.org/

1.2 Doelstellingen

Het uiteindelijke doel is de creatie van een systeem dat in staat is om flexibele updates te verspreiden voor Cordova applicaties. Hierbij dient het systeem beveiligd te zijn zodat er geen misbruik mogelijk is en de privacy van de gebruiker geborgen is.

We analyseren bovenstaande paragraaf en onderscheiden volgende elementen:

- *"Het doel is de creatie van een systeem ..."* - Tijdens dit eindwerk worden relaties gevormd tussen losstaande componenten. Dit kunnen reeds bestaande elementen, maar ook specifiek ontworpen elementen zijn. Bij deze elementen denken we aan herbruikbare code zoals beschikbaar via npm[3]. Indien een reeds bestaand element kan gebruikt worden, zal dit verkozen worden boven de eigen creatie van datzelfde element. Bestaande pakketten code hebben enkele voordelen t.o.v. zelf geschreven code. Zo is de kans groot dat het pakket geschreven is door iemand met meer ervaring in het vakgebied. Dit resulteert in efficiëntere code met een betere foutafhandeling. Bovendien beschikt het pakket over de online ondersteuning van de gemeenschap. Omwille van deze redenen zal er tijdens het verdere verloop van het eindwerk zoveel mogelijk gewerkt worden met reeds bestaande codepakketten.

- *"... dat in staat is om flexibele updates te verspreiden ..."* - Met het woord 'flexibel' wordt er bedoeld dat niet elke gebruiker dezelfde update mag/zal ontvangen. Het systeem moet m.a.w. gebruikers van de applicatie kunnen identificeren en vervolgens ervoor zorgen dat de juiste update naar de juiste persoon wordt gepropageerd. Om dit te realiseren, zullen er twee elementen nodig zijn: een element dat instaat voor de identificatie van de gebruiker en een element dat nadien een beslissing maakt op basis van de identiteit van de gebruiker en het gehanteerde beleid.

- *"... voor Cordova applicaties."* - Het Apache Cordova framework speelt een centrale rol in dit eindwerk. Het gerealiseerde systeem is zodanig ontworpen dat enkel Cordova applicaties ervan kunnen gebruik maken. De communicatie tussen de applicatie en de rest van het systeem wordt gerealiseerd door een update plugin. Het gebruik van plugins is eigen aan het Cordova framework. Het zijn de plugins die ervoor zorgen dat de applicatie kan gebruik maken van de functies van het onderliggende besturingssysteem. De update plugin zal als taak hebben de updates te downloaden en te installeren.

- *"Hierbij dient het systeem beveiligd te zijn zodat er geen misbruik mogelijk is ..."* - Misbruik van het systeem treedt op wanner gebruikers toegang hebben tot functionaliteiten waar ze eigenlijk geen recht op hebben. De manier waarop rechten toegewezen worden aan een gebruiker wordt bepaald door diegene die de applicatie distribueert. Zo kan een gebruiker het recht op een bepaalde set functies verwerven door deze simpelweg toegewezen te krijgen of bv. door de betalingsoptie te kiezen die hoort bij de set in kwestie. Indien een gebruiker door manipulatie van het systeem toegang krijgt tot extra functies, is er sprake van misbruik.

- *"... en de privacy van de gebruiker geborgen is."* - Het privacyrecht zegt dat persoonlijke gegevens niet in de handen van een derde partij mogen vallen zonder de toestemming van de personen in kwestie. Het systeem zal gegevens van de gebruikers bevatten, en daarom is het van cruciaal belang dat deze gegevens worden afgeschermd van de buitenwereld. De privacy van de gebruiker kan zelfs nog een stap verder worden doorgetrokken: een aanvaller

[3] https://www.npmjs.com/

mag op geen enkele manier in staat zijn om het mobiel toestel van de gebruiker binnen te dringen gebruikmakend van het systeem.

1.3 Overzicht van de tekst

In de volgende paragrafen wordt een overzicht gegeven hoe de rest van de tekst eruitziet. Zo wordt het voor de lezer duidelijk wat er nog kan verwacht worden van de scriptie.

Hoofdstuk 2 begint met een concretere beschrijving van het scenario. Zo wordt het zeker duidelijk waarvoor het updatesysteem zou kunnen gebruikt worden. De werking en het nut van het systeem zullen ook eenvoudiger te begrijpen zijn voor de lezer als men een praktisch scenario in het achterhoofd houdt. Verder bestaat het hoofdstuk uit een beschrijving van een aanvallersmodel dat een belangrijke rol zal spelen bij de analyse van bestaande Cordova update plugins.

Hoofdstuk 3 gaat van start met een korte inleiding over enkele huidige updatemechanismes. Vervolgens wordt het Cordova framework en het gebruik van plugins nader toegelicht aangezien het begrip van deze elementen van fundamenteel belang is voor de rest van het eindwerk.
Daarna vindt er een gedetailleerde analyse plaats van enkele bestaande update plugins. Dit houdt o.a. in op welke manier ze te werk gaan, hoe ze zijn opgebouwd, wat de veiligheidsrisico's zijn, welke andere elementen er nodig zijn, welke technologieën er gebruikt zijn, enz.
Vervolgens zullen de geanalyseerde plugins getoetst worden aan het aanvallersmodel beschreven in hoofdstuk 2. Na afloop van het kwalitatieve onderzoek en de veiligheidsanalyse wordt er in het besluit een update plugin gekozen die de basis zal vormen van het systeem. Dit vormt de eerste grote stap in het eindwerk.

De beschrijving van het ontwerp in hoofdstuk 4 vertrekt van het geheel en zoomt stap voor stap in op alle aanwezige componenten. Ook de samenhang van de componenten wordt hier verduidelijkt. Het systeem wordt schematisch voorgesteld a.d.h.v. een aantal diagramma. Zo toont een sequentiediagram hoe de communicatie in het systeem verloopt en in welke volgorde de berichten worden verstuurd.
Het hoofdstuk bevat een paragraaf waar de grootste ontwerpsbeslissingen geargumenteerd worden en eventuele alternatieven worden voorgesteld. Merk op dat dit hoofdstuk enkel de structurele beslissingen behandelt, de realisatie ervan is behouden voor hoofdstuk 5. Na het lezen van dit hoofdstuk zou de volledige functionaliteit en werking van het systeem duidelijk moeten zijn voor de lezer.

Hoofdstuk 5 beschrijft de implementatie van het ontwerp uit hoofdstuk 4. Hierbij wordt de oorsprong en gebruikte technologie van elke component besproken. Het zou duidelijk moeten worden voor de lezer welke componenten er zelf geschreven zijn en welke niet. Sommige implementaties worden geïllustreerd a.d.h.v. codevoorbeelden. Er zal blijken dat enkele componenten afkomstig zijn van het internet. De ideale situatie zou inhouden dat deze componenten als *black box* kunnen worden gebruikt. Dit houdt in dat de componenten worden geïntegreerd in het systeem zonder er iets aan te veranderen. Zo kunnen de componenten probleemloos genieten van updates die worden vrijgegeven door de auteur. Dit was jammer genoeg niet overal mogelijk. Toch is het systeem zo ontworpen dat er minimale aanpassingen zijn gemaakt aan de herbruikte componenten.

Het grootste deel van hoofdstuk 6 zal bestaan uit een veiligheidsanalyse van het systeem. In deze analyse wordt het duidelijk waar mogelijke aanvallen kunnen plaatsvinden en wat de geïmplementeerde

oplossing ervoor is. Vervolgens vindt er in dit hoofdstuk een sterkte-zwakteanalyse plaats. Ten slotte eindigt het hoofdstuk met een paragraaf waarin mogelijke uitbreidingen voor het systeem worden besproken. Dit houdt o.a. ook in welke moeilijkheden er zouden optreden indien men het systeem in een andere omgeving zou willen implementeren.

Hoofdstuk 7 handelt over het prototype dat werd ontworpen bij het systeem. Het prototype dient om de functionaliteit van het achterliggend systeem te demonstreren. Het toont aan hoe eenvoudig het flexibele updatesysteem te werk gaat.
Eerst zal de werking van het gemaakte prototype beschreven worden. Dit zal gepaard gaan met screenshots als visuele ondersteuning voor de lezer. Hierna volgen er instructies op welke manier apps moeten gestructureerd worden zodat ze het flexibele updatesysteem correct kunnen gebruiken.

Ten slotte wordt er in hoofdstuk 8 een algemeen besluit gevormd. Om te beginnen wordt er nagegaan worden of er voldaan werd aan de vereisten die werden geformuleerd in hoofdstuk 1.2. Hierbij is extra aandacht besteed aan de security eisen. Vervolgens worden de belangrijkste stappen doorheen het eindwerk nog eens aangekaart en er wordt afgesloten met een bondige formulering van het gerealiseerde systeem en de voordelen die het met zich meebrengt.

Alle code die werd gebruikt in dit eindwerk is te vinden op GitHub via onderstaande link. Bij het eindwerk hoort ook een CD waar ook alle code op te vinden is.

https://github.com/Jonas-Semeelen/Eindwerk

Hoofdstuk 2

Analyse

2.1 Gevalstudie

Dit hoofdstuk begint met een gedetailleerde beschrijving van het scenario waarvoor het systeem is ontworpen. Dit zal het makkelijker maken voor de lezer om later de functionaliteit van het systeem te begrijpen. De lezer moet echter wel in gedachten houden dat het beschreven scenario slechts een van de vele mogelijke toepassingen is voor het systeem. Het scenario is louter een manier om de praktische kant van het systeem weer te geven.
Het gekozen scenario is een niet-professionele thuiszorgsituatie waar er eventueel ook thuisverpleging nodig is. Dit houdt in dat de cliënt wordt verzorgd door een familielid of een naaste omdat professionele hulp niet nodig is of geen optie is. De thuiszorg kan een intense karwei zijn voor de persoon die de rol van verzorger aanneemt. Vaak moeten er dagelijks verscheidene taken ter plaatse worden uitgevoerd. Dit kan gaan van het uitvoeren van eenvoudige klusjes tot verpleegkundige verzorging. Er zijn ook gevallen waarbij er toezicht moet gehouden worden op de cliënt vanwege gezondheidsredenen.
Vele van deze situaties waarbij men momenteel vaak nog persoonlijk moet gaan observeren, zouden kunnen vermeden worden door het installeren van sensoren in het huis van de cliënt. Dit kan de werklast van de verzorger significant verminderen. Het is uiteraard niet de bedoeling om het contact tussen cliënt en verzorger volledig weg te nemen aangezien dit een niet te onderschatten rol speelt in het verzorgingsproces. Het is de bedoeling dat beide partijen voordeel halen uit de invoer van het systeem. Denk aan een situatie waarbij het huis voorzien is van detectoren aan elke deur. Dit zou het bijvoorbeeld mogelijk maken om de toiletbezoeken van de cliënt in kaart te brengen. Indien dit patroon afwijkt van het normale patroon, kan dit op een medisch probleem wijzen. Dit voorbeeld toont aan dat het mogelijk is om een potentieel symptoom op te merken a.d.h.v. de sensoren nog voor de patiënt door heeft dat er iets aan de hand is.
De demo bij dit eindwerk zal gebruik maken van een Cordova applicatie die verbonden is met enkele sensoren zoals licht-, geluid-, temparetuur-, druk- en deursensoren. Dit zijn voorbeelden van types sensoren die je in een thuiszorgwoning zou kunnen terugvinden. Het type en aantal sensoren zal van woning tot woning verschillen naargelang de behoeften van de patiënt. Aangezien de behoeften en sensoren zullen verschillen, zal ook de functionaliteit van de bijhorende applicatie verschillend zijn. Dit resulteert in verschillende versies van dezelfde applicatie. Elke versie zal een andere update moeten ontvangen. Omwille van deze reden zal er nood zijn aan gepersonaliseerde updates in dit scenario.

2.2 Aanvallersmodel

Het aanvallersmodel dat in dit onderdeel wordt besproken, is specifiek voor Cordova update plugins. Het model zal de meest kwetsbare plaatsen van een typische plugin in kaart brengen. In hoofdstuk 3.3 worden twee plugins getest a.d.h.v. dit aanvallersmodel. Om de volgende paragrafen beter te begrijpen, wordt er eerst de algemene werking van een Cordova update plugin gegeven.
Een update plugin zorgt ervoor dat verbeteringen aan een app onmiddellijk de gebruikers bereiken. Dit wordt gerealiseerd door de nieuwe updates op een server te zetten. Elke keer de app opstart, wordt er nagegaan bij de server of er een nieuwe versie beschikbaar is. Indien er een nieuwe versie beschikbaar is en het toestel internetverbinding heeft, dan zal de update worden gedownload. Op deze manier beschikt de gebruiker altijd over de meest recente versie van de applicatie.

We definiëren een eerste soort aanval als een aanval waarbij de aanvaller als doel heeft om kwaadaardige code te uploaden naar de updateserver onder de vorm van een update. Indien de aanval slaagt, zal de geïnfecteerde update automatisch verspreid worden naar alle nietsvermoedende gebruikers van die applicatie. Eenmaal geïnfecteerd, heeft de aanvaller een zekere macht over het toestel van de gebruikers. Met deze macht kan hij bijvoorbeeld persoonlijke informatie zoals contacten uit de contactenlijst proberen stelen. De effectieve macht van de aanvaller hangt echter wel af van de applicatie waarbinnen de kwaadaardige code zit.
Bij een toestel met Android besturingssysteem is elke applicatie namenlijk gebonden aan de *permissions*[4] die werden gedefinieerd in het Android manifest. Een Android app wordt uitgevoerd in een geïsoleerde *sandbox* en de permissions bepalen tot welke functies en resources de app toegang heeft buiten de sandbox. De privacygevoelige permissions worden gecategoriseerd onder 'dangerous permissions' en het zijn net deze resources waar een aanvaller is in geïnteresseerd. De dangerous permissions bevatten nog een extra veiligheidsmaatregel: sinds Android 6.0.0 moet de gebruiker elke keer expliciet toestemming geven wanneer een app wil gebruik maken van resource beschermd door een dangerous permission. Op deze manier kan de gebruiker argwaan krijgen wanneer een app willekeurig toestemming vraagt tot de contactenlijst. Toestellen die Android versie 5.1 of lager gebruiken, beschikken niet over dit beveiligingsmechanisme. Er wordt enkel om de toestemming van de gebruiker gevraagd bij de installatie van de app. Als de app nadien wordt geïnfecteerd, dan kan de kwaadaardige code gebruik maken van alle permissions die de gebruiker bij de installatie van de app heeft goedgekeurd. Indien dit ook dangerous permissions zijn, is de privacy van de gebruiker in gevaar.
De geïnfecteerde update zal zich bovendien razendsnel verspreiden onder de gebruikers aangezien de bron van de infectie zich in de database van de updateserver bevindt. Indien de aanvaller een app met veel permissions en een grote gebruikersbasis weet te infecteren, kan de schade enorm zijn. Omwille van de centrale ligging van de infectie in het systeem en potentiële schade aan de gebruikers, vormt deze soort aanval een enorm risico. Het is daarom absoluut nodig dat er veiligheidsmaatregelen worden getroffen tegen deze aanval. Vanaf nu zal naar deze aanval gerefereerd worden als aanval type 1.

Een tweede type aanval is de klassieke *man-in-the-middle* aanval. Bij deze aanval probeert een aanvaller de communicatie ergens binnen het systeem te onderscheppen. In dit geval is dit de communicatie tussen de plugin en de updateserver. Indien deze communicatie niet geëncrypteerd is, kan de aanvaller de verstuurde data achterhalen en eventueel wijzigen. Als de identiteit van de

[4] https://developer.android.com/guide/topics/permissions/index.html

updateserver niet op correcte wijze wordt gecontroleerd door de update plugin, kan een aanvaller zich uitgeven als de updateserver en schadelijke updates doorsturen naar de app. De schade die de aanvaller kan toebrengen is dezelfde als deze bij de type 1 aanval, maar het is veel moeilijker voor een aanvaller om hetzelfde aantal apps te infecteren. Dit komt omdat de infectie zich nu niet meer even centraal bevindt zoals bij de type 1 aanval.
Om deze aanval te doen slagen, moet een aanvaller een *rogue access point* opzetten om de communicatie tussen de applicatie en het normale Wi-Fi access point te onderscheppen. Vervolgens mag er geen eindpunt authenticatie aanwezig zijn tijdens de communicatie. Als dit mechanisme wel aanwezig is, kan gedetecteerd worden dat de update niet afkomstig is van de updateserver en zal de verbinding met de aanvaller worden verbroken. Al deze obstakels maken de aanval moeilijker uit te voeren en verlagen de totale impact van de aanval.

Hoofdstuk 3

State-of-the-art

In dit hoofdstuk wordt er een overzicht gegeven van de reeds bestaande technologieën die in verband staan met dit eindwerk. De nadruk zal liggen op de update plugins die de dag vandaag gebruikt worden in Cordova applicaties en de implementatie ervan. Nadien vindt er een analyse plaats van twee update plugins. De meest geschikte plugin van de twee wordt nadien gebruikt als basiscomponent in de rest van het eindwerk. Maar eerst wordt in het volgende hoofdstuk een korte vergelijking gemaakt tussen de huidige updatemechanismen als inleiding. Hierbij ligt de focus op het Android besturingssysteem om de vergelijking te maken.

3.1 Huidige updatemechanismen

3.1.1 Klassieke applicaties

Om apps te installeren maakt het Android besturingssysteem gebruik van APK-bestanden. Een APK-bestand wordt gevormd door het gecompileerde Android programma tesamen met alle resources en het manifest-bestand te comprimeren tot één bestand. Merk op dat het hier reeds gaat over een gecompileerd Android programma.
Om een update uit te voeren, moet er een nieuw APK-bestand worden gedownload. Vervolgens wordt er nagegaan of de oude APK kan upgraden naar de nieuwe APK. Dit houdt in dat beide APK's moeten ondertekend zijn door dezelfde sleutel zodat er zekerheid is dat de nieuwe APK van dezelfde auteur is. Verder worden er ook nog enkele zaken uit de manifesten vergeleken zoals de versies van beide APK's zodat er geen oudere versie wordt geïnstalleerd.
Indien alle controles probleemloos zijn verlopen, wordt de database met applicaties geüpdatet zodat er wordt verwezen naar de nieuwe APK. Als laatste stap wordt de oude APK verwijderd.

Het belangrijkste om van het bovenstaande te onthouden is dat er een nieuw gecompileerd programma moet gedownload worden via een APK om een update van een app uit te voeren. Dit downloadproces vereist meestal de tussenkomst van een app store. APK's kunnen ook manueel gedownload worden van andere bronnen, maar dit is niet gebruikelijk.

3.1.2 Cordova applicaties

Om een commerciële Cordova applicatie te installeren zal dit net zoals een klassieke applicatie ook via een app store gebeuren. Indien het gaat over een Android applicatie zal dit eveneens a.d.h.v. een APK zijn. Het grote verschil met een klassieke Android applicatie is dat de functionaliteit van

een Cordova applicatie zich niet bevindt in het gecompileerde programma, maar in de resources. Dit betekent dat er geen hercompilatie nodig is wanneer de functionaliteit van de applicatie wijzigt. De applicatie kan updaten door simpelweg de de code in de resources te wijzigen. In hoofdstuk 3.2.3 wordt dieper ingegaan op dit mechanisme.
Een Cordova app kan m.a.w. updaten zonder hercompilatie en dus ook zonder tussenkomst van een app store. Dit is een zeer aantrekkelijke eigenschap voor applicaties die regelmatig updates ondervinden. De afzijdigheid van een app store maakt het een stuk eenvoudiger om deze updates uit te voeren. Omdat de updates heet van de naald de gebruiker kan bereiken, spreken we van ”hot code updates”.
De vergelijking werd hierboven gemaakt met Android, maar Cordova ondersteunt ook iOS, Windows Phone en nog tal van andere besturingssystemen. Alle ondersteunde systemen zijn te vinden op de website van Cordova[5]. Deze platformen hebben elk hun eigen installatieproces. Maar als de applicatie gemaakt is m.b.v. het Cordova framework, dan genieten ze allemaal van het voordeel van hot code updates.

3.2 Het Cordova framework

3.2.1 Inleiding

Cordova is een framework om cross-platform mobiele applicaties te ontwikkelen. Figuur A.1 op pagina 37 geeft de structuur weer van een Cordova applicatie. Er kunnen drie grote delen worden onderscheiden: de webview, de web app en de plugins. De webview voorziet de applicatie van een gebruikersinterface. Deze interface komt voort uit de HTML en CSS die voorzien is in de web app. De web app bevat ook javascript bestanden, deze zorgen dan weer voor de werking van de app. Zowel HTML, CSS als javascript zijn platformonafhankelijke talen, vandaar de cross-platform eigenschap van Cordova applicaties.
De functionaliteit van een Cordova applicatie komt van de geïnstalleerde plugins. Het zijn de plugins die de native code (platformspecifieke code) bevatten om API calls te maken naar het besturingssysteem van het toestel. Een goede plugin bevat native code voor veel platformen. Op deze manier kan de plugin geïnstalleerd worden bij applicaties voor al deze platformen. Tijdens de installatie van de applicatie op het toestel, wordt dan enkel de relevante native code van de plugins geïnstalleerd.
Een voorbeeld van een populaire plugin is de camera plugin. Een Cordova applicatie zal enkel de camera van een toestel kunnen gebruiken als deze plugin is geïnstalleerd. Als app ontwikkelaar moet je simpelweg de camerafuncties oproepen m.b.v. de scripttaal, waarna de Cordova en besturingssysteem API’s de rest af handelen. De camera plugin behoort tot de set van *core* plugins, dit is een set van 20 plugins dat voorzien is door Cordova zelf. Core plugins hebben als eigenschap dat ze de meeste platformen ondersteunen. Figuur A.1 op pagina 37 in de bijlagen toont 8 plugins die tot deze set behoren. De volledige set core-plugins en de ondersteunde platformen per plugin zijn te vinden in de documentatie[6] op de website van Cordova.
Bovendien is het mogelijk om zelf een plugin te ontwikkelen en toe te voegen aan een Cordova applicatie. Indien gewenst, kan deze ook worden toegevoegd aan de Cordova plugin store[7] zodat de plugin ook beschikbaar is voor andere ontwikkelaars.

[5] https://cordova.apache.org/docs/en/latest/guide/support/index.html
[6] https://cordova.apache.org/docs/en/latest/guide/support/index.html#core-plugin-apis
[7] https://cordova.apache.org/plugins/

3.2.2 Cordova plugins

Elke plugin is opgebouwd volgens een vaste structuur die te zien is op figuur 3.1 pagina 10. De native code van de ondersteunde platformen bevindt zich in de src-map, terwijl de Javascript interface zich in de bin-map bevindt.

Figuur 3.1: Structuur van een Cordova plugin

De interface zorgt voor de communicatie met de native kant waardoor het kan beschouwd worden als het belangrijkste element van de plugin. De interface moet een specifieke syntax volgen om de native functies op te roepen. Cordova voorziet de cordova.exec functie om deze *calls* te maken. De syntax van deze functie is te zien op figuur 3.2 pagina 10. Hieronder volgt een korte uitleg bij de parameters:

- function(winParam): Een callback die wordt uitgevoerd als de exec call succesvol is.
- function(error): Een callback die wordt uitgevoerd als de exec call niet succesvol is.
- service: De naam van de service dat moet worden opgeroepen aan de native kant. Dit stemt overeen met een native bestand uit de src-map.
- action: De naam van de actie die moet worden opgeroepen aan de native kant. Dit stemt overeen met een methode in het bestand van bovenstaande parameter.
- arguments: Array van argumenten om door te geven naar de native omgeving.

```
cordova.exec(function(winParam) {},
             function(error) {},
             "service",
             "action",
             ["firstArgument", "secondArgument", 42, false]);
```

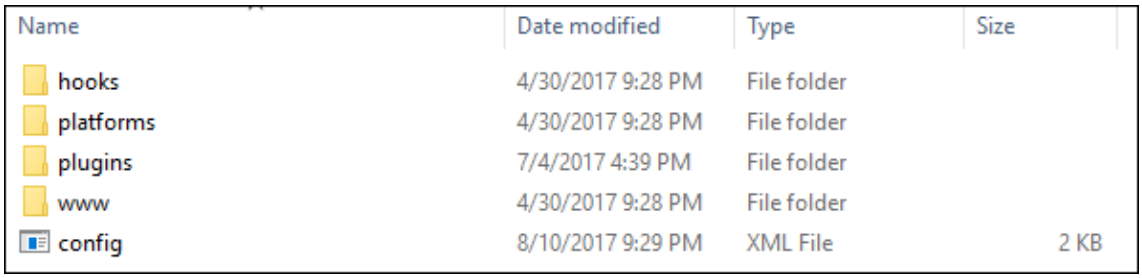
Figuur 3.2: Syntax cordova.exec

3.2.3 Een Cordova applicatie

Nu de lezer wat wijzer is geworden over de plugins, wordt er een stap terug genomen en wordt de volledige Cordova applicatie beschouwd. We beschouwen eerst de *root* van een Cordova project. De structuur van een typisch project is te zien op figuur 3.3 op pagina 11. Hieronder volgt een korte uitleg bij de mappen en bestanden:

- plugins: alle geïnstalleerde plugins bevinden zich in deze map. De bestandsstructuur van elke plugin was al eerder te zien op figuur 3.1 pagina 10. Zoals eerder gezegd bestaan de plugins uit zowel native code van de ondersteunde platformen als Javascript interface bestanden.
- www: bevat alle bestanden die tot de web app behoren. In het begin van hoofdstuk 3.2 werd reeds gesteld dat deze bestanden allemaal geschreven zijn in een platformonafhankelijk taal. Daarenboven bevat deze map alle resources die de app nodig heeft. Een voorbeeld hiervan zijn de afbeeldingen die worden gebruikt in de app.
- platforms: bevat de implementatie van de www-map voor elk ondersteund platform. Alle mappen en bestanden in deze folder worden automatisch aangemaakt door het Cordova framework. Vanuit het standpunt van een ontwikkelaar, is enkel de www-map van belang, het Cordova framework zorgt voor de rest.
- config.xml: configuratiebestand waarin het gedrag van de Cordova applicatie kan worden gedefinieerd.
- hooks: in deze map kan een ontwikkelaar speciale scripten toevoegen. De hooks worden gebruikt om Cordova commando's aan te passen naar de noden van de ontwikkelaar. Dit is echter een geavanceerde ontwikkelaarsoptie en werd niet gebruikt tijdens het verloop van dit eindwerk.

Het belangrijkste om te onthouden is dat een Cordova project naast de platformonafhankelijke code ook native of platformspecifieke code bevat van alle ondersteunde platformen. Tijdens de installatie van de app wordt de www-map tesamen met de juiste platformspecifieke code op het toestel geïnstalleerd.
Herinner u dat alles binnen de www-map deel uitmaakt van de web app uit figuur A.1 pagina 37 en dat alle bestanden in deze mappen platformonafhankelijk zijn. Omdat er geen native code aanwezig is, kan een update eenvoudig uitgevoerd worden door de bestanden te vervangen door een nieuwe versie ervan. Let op dat het hier enkel gaat over www-map. De geïnstalleerde plugins bevinden zich niet in deze map, zie figuur 3.3.
Omdat bestanden en mappen zeer eenvoudig te manipuleren zijn m.b.v. Javascript zijn er een heleboel plugins beschikbaar om Cordova applicaties te updaten. Dit updateproces gebeurt zonder tussenkomst van een app store en zonder dat de gebruiker hiervan enige last ondervindt.



Figuur 3.3: Root van een Cordova project

3.3 Analyse van de plugins

3.3.1 CodePush

De eerste plugin die wordt geanalyseerd is CodePush[8] van Microsoft. CodePush is speciaal ontworpen voor applicatieontwikkelaars zodat ze rechtstreekser in verbinding staan met de gebruikers van hun applicaties. CodePush voorziet drie elementen om dit mogelijk te maken: een cloud service om de apps en updates op te slaan, een command line interface (CLI) om de updates naar de cloud service te pushen en ten slotte een Cordova plugin om de cloud service aan te spreken vanaf het mobiele toestel en updates binnen te halen. Voordat er wordt ingegaan op de werking van de plugin zelf, worden de stappen overlopen die ontwikkelaar moet nemen om gebruik te kunnen maken van de CodePush plugin.

Eerst en vooral moet de ontwikkelaar een CodePush account aanmaken via de CodePush CLI. Het commando *register* opent een browservenster waar de ontwikkelaar zich kan authenticeren a.d.h.v. een Microsoft of een GitHub account. Indien de authenticatie geslaagd is, verschijnt er een eenmalig token in de browser dat moet ingevoerd worden in de CLI. Op deze manier kan de CLI de identiteit bevestigen en een nieuwe sessie starten. De ontwikkelaar kan nu ook gebruik maken van alle andere beschikbare commando's om zijn apps en updates eenvoudig te beheren.
Eenmaal ingelogd kan een nieuwe app worden geregistreerd bij de cloud service. De app wordt permanent gelinkt aan het ingelogde CodePush account en enkel dit account kan veranderingen maken aan de app. Deze instelling kan gewijzigd worden door gebruik te maken van het commando *collaborator*. Dit commando laat toe om projecten te delen met andere ontwikkelaars. Na de registratie van de app wordt er een deployment key toegekend aan de applicatie. De sleutel verschijnt in de CLI en dient gekopieerd te worden naar het config.xml bestand van de applicatie. De sleutel zal een belangrijke rol spelen bij het updaten van de app.
Als een ontwikkelaar een applicatie maakt voor meerdere besturingssystemen, dan moet die app voor elk versie apart worden geüpload naar de cloud service. De cloud service beschouwt de verschillende versies namelijk als verschillende apps. Er zal bijgevolg ook een unieke deployment key per versie worden gegeven. Stel dat een app zowel een Android als een iOS versie heeft, dan moet de applicatie geconfigureerd worden zoals te zien is op figuur 3.4.

```
19      <platform name="android">
20          <preference name="CodePushDeploymentKey" value="YOUR-ANDROID-DEPLOYMENT-KEY" />
21      </platform>
22      <platform name="ios">
23          <preference name="CodePushDeploymentKey" value="YOUR-IOS-DEPLOYMENT-KEY" />
24      </platform>
```

Figuur 3.4: Codefragment config.xml van een applicatie ondersteund door twee platformen

De laatste stap is een update vrijgeven naar de cloud service. Dit gebeurt a.d.h.v. het commando *release*. Het commando heeft twee parameters: *appName* en *updateContentsPath*. Hierbij stelt de parameter 'appName' de naam voor waaronder de app geregistreerd is bij de cloud service. De parameter 'updateContentsPath' is het lokale pad naar de map die de nieuwe versie van de app bevat. Het is belangrijk dat dit pad wijst naar de juiste versie van de app. Zoals eerder gezegd, genereert Cordova een implementatie van de www-map voor elk ondersteund platform. De android versie van de app bevindt zich relatief t.o.v. de root op het pad */platforms/android/assets/www*. Het

[8] https://microsoft.github.io/code-push/

is deze map waar updateContentsPath naar moet wijzen als men de nieuwe android versie van de app wil releasen aan de cloud service.
Voor de release wordt uitgevoerd, wordt er nagegaan of de ingelogde persoon over de rechten beschikt om een update vrij te geven voor de gespecificeerde app. Dit gebeurt a.d.h.v. het logintoken dat lokaal werd opgeslagen. De vrijgave van de update gaat enkel door als de ingelogde persoon in de lijst van collaborators van de app staat. Bovendien gaat de update ook niet door als er geen verschil is tussen de lokale app en de app in de cloud. Dit wordt getest door een hash te nemen van alle lokale files en deze hash te vergelijken met een de hash die hoort bij de app in de cloud.

Indien bovenstaande stappen goed zijn verlopen, kan de app worden geüpdatet m.b.v. de plugin. Om de CodePush plugin te gebruiken, roep je de synchronisatiemethode op in de *handler* van het *deviceready* event zoals te zien is op figuur 3.5 pagina 13. Deze methode is het begin van een lange ketting methodes en callbacks. Eerst verzamelt de plugin een hoop informatie over applicatie. De belangrijkste elementen zijn de deployment key en de packagehash van de files waaruit de app bestaat. Deze informatie wordt vervolgens m.b.v. een HTTP-bericht verstuurd naar de cloud service. Daar wordt de juiste app geïdentificeerd a.d.h.v. de deployment key die werd opgestuurd. Beide hashes worden vergeleken om uit te maken of de cloud service een nieuwere versie van de app bevat. In het geval dat er een update beschikbaar is, bevat het antwoord van de cloud service een URL waarvan de update kan worden gedownload. In het andere geval waar de hashes gelijk zijn er dus geen update beschikbaar is, wordt er een leeg antwoord teruggestuurd.

```
29      onDeviceReady: function() {
30          codePush.sync();
31          this.receivedEvent('deviceready');
32      },
```

Figuur 3.5: De synchronisatiemethode wordt opgeroepen na de deviceready event

Om de nieuwe update de downloaden, maakt de CodePush plugin gebruik van een andere plugin, nl. de FileTransfer[9] plugin. Deze plugin beschikt over zowel een download als een uploadfunctie, maar de plugin wordt echter enkel gebruikt om bestanden te downloaden. De downloadmethode neemt enkele parameters aan zoals de download-URL en het pad waar het gedownloade bestand moet geplaatst worden en maakt vervolgens gebruik van de Android API's om de bestandoverdracht te vervolledigen.
De update wordt gedownload onder de vorm van een gezipt bestand. Als laatste stap wordt dit bestand geünzipt en is de nieuwe versie klaar voor gebruik. De nieuwe versie van de app treedt echter pas in gebruik nadat de app wordt herstart. Dit komt doordat de CodePush plugin altijd de vorige versie van de app bijhoudt in het geval dat de nieuwe app zorgt voor een crash. Zo kan er een *rollback* gebeuren naar de vorige versie en beschikt de gebruiker altijd over een werkende app.

Er dient echt wel een zeer belangrijke opmerking te worden gemaakt. De zaken die kunnen worden geüpdatet zijn beperkt tot de mappen en bestanden die zich in de www-folder bevinden. Veranderingen aan de app die te maken hebben met platformspecifieke code, zoals het toevoegen van een nieuwe plugin aan de applicatie, kunnen niet worden uitgevoerd door de CodePush plugin. Hiervoor zal een app store toch tussen beide moeten komen om de app te updaten. Bovendien geldt

[9] https://github.com/apache/cordova-plugin-file-transfer

deze beperking niet enkel voor de CodePush plugin, maar is elke Cordova update plugin hieraan onderdanig.
Nu de werking van de CodePush plugin duidelijk is voor de lezer, wordt de plugin in de volgende paragrafen kritisch besproken. Bij deze bespreking zal de nadruk liggen op het security aspect.

Eerst en vooral wordt de plugin geanalyseerd op basis van resistentie tegen het aanvallersmodel gedefinieerd in hoofdstuk 2.2.
De type 1 aanval werd gedefinieerd als een aanval waarbij de aanvaller een geïnfecteerde update op de updateserver probeert te zetten. Eerder in dit hoofdstuk werd het voor de lezer duidelijk dat interactie met de cloud service van CodePush gebeurt via een CLI. Om een update voor een bepaalde app vrij te geven, moet er ingelogd zijn met het juiste account. Enkel het account dat de app heeft geregistreerd en de accounts die expliciet zijn toegevoegd als collaborator hebben het recht om updates vrij te geven.
Stel: de aanvaller heeft een applicatie op het oog met veel permissions en een groot aantal gebruikers. De enige manier om een type 1 aanval succesvol uit te voeren is door de GitHub of Microsoft logingegevens van een collaborator van de app te achterhalen. Enkel met deze gegevens kan een aanvaller inloggen op een account dat over de rechten beschikt om een update pushen naar de cloud service. Bovendien is er geen manier om te achterhalen wie behoort tot de collaborators van een bepaalde app, tenzij de ontwikkelaars van de app hier zelf voor uitkomen.
Al deze zaken maken het extreem moeilijk voor een aanvaller om een type 1 aanval uit te voeren. Bovenstaand veiligheidsmechanisme heeft enkel effect indien de communicatie tussen de CLI en de cloud service geëncrypteerd is. Indien dit niet het geval is, hoeft een aanvaller slechts één maal mee te luisteren op de verbinding wanneer een collaborator inlogt in de CLI. Vanaf dat moment beschikt de aanvaller over de logingegevens en kan hij zelf inloggen met de gegevens.

In het algemeen kan een type 2 MITM aanval voorkomen worden door een vorm van authenticatie en *tamper detection* te gebruiken over de onbeveiligde verbinding. CodePush implementeert deze twee door de communicatie tussen de plugin en de cloud service over een HTTPS-verbinding te doen verlopen. Het protocol zorgt ervoor dat bij elke nieuwe verbinding de eindpunten zich moeten authenticeren en dat de communicatie tussen beide wordt versleuteld. Zo wordt het onmogelijk voor een aanvaller om mee te luisteren op de verbinding, en zal het gedetecteerd worden als de aanvaller data probeert te injecteren in de verbinding.
Bij dit onderdeel dient er wel een belangrijke opmerking te worden gemaakt. Tijdens de analyse van de CodePush plugin werd er een veiligheidsrisico ontdekt. Het risico ontstaat tijdens de download van de update. Kort samengevat wordt er een parameter doorgegeven van de CodePush plugin naar de FileTransfer plugin waardoor alle certificaten blindelings worden vertrouwd. De parameter heeft als default waarde 'false' in de FileTransfer plugin, maar ze krijgt expliciet de waarde 'true' door de call vanuit de CodePush plugin. De parameter dient als optie voor ontwikkelaars, omdat die vaak gebruik maken van zelf getekende certificaten tijdens het ontwikkelen en testen van een applicatie. Normaal gezien weigert het besturingssysteem een zelf getekend certificaat omdat het niet ondertekend is door een geldig certificaatautoriteit. Maar door deze parameter op *true* te zetten, worden alle certificaten, ook de zelf getekende, aanvaard. Hierdoor is de verbinding vatbaar voor een MITM aanval ook al gebeurt de communicatie via HTTPS. De *vulnerability* zorgt ervoor dat de certificaten die zorgen voor authenticatie waardeloos zijn. In de volgende paragraaf wordt de *threat* in detail gedocumenteerd zodat de lezer deze zelf kan controleren. De relevante bestandsnamen en regels code kan de lezer terugvinden in bijlage B.

De manier waarop CodePush gebruik maakt van de FileTransfer plugin is te zien op figuur B.1 in

bijlage B. In het codefragment valt te zien dat de vijfde en laatste parameter van deze call 'true' is. Merk op dat dit de standaard instelling is bij de installatie van de CodePush plugin.
Vervolgens volgt er een analyse van de download-methode in de FileTransfer plugin, deze is te zien op figuur B.2 in bijlage B. In de parameters van de functie vinden we als vijfde parameter 'trustAllHosts' terug. Het is deze parameter die de waarde 'true' meekrijgt bij de call door de CodePush plugin. In het commentaar op lijn 175 staat dat de waarde van deze parameter per *default* 'false' is. Toch roept CodePush deze methode op met de trustAllHosts parameter op true zodat de default waarde wordt overschreven. Dit resulteert in een situatie waar de certificaten niet meer worden gecontroleerd. Hoe dit juist komt wordt duidelijk in figuur B.3 bijlage B.
In de figuur valt de zien hoe er een HTTPS verbinding wordt opgezet waarbij iedereen wordt vertrouwd. Dit gebeurt door een socketFactory aan te maken die alle certificaten vertrouwt. De code hiervan is te zien op figuur B.4 bijlage B. In het commentaar bij de methode wordt duidelijk gezegd dat deze code expliciet dient voor ontwikkelaars die gebruik maken van zelf getekende certificaten op hun webserver voor ontwikkelingsdoeleinden. Het is m.a.w. niet de bedoeling dat deze code wordt uitgevoerd in een echte situatie.

Het is onwaarschijnlijk dat de ontwikkelaars van de CodePush plugin deze ontwikkelaarsoptie met opzet op 'true' zetten. Het is waarschijnlijk onopgemerkte implementatiefout zijn, maar één ding is zeker: het vormt een enorm veiligheidsrisico indien deze code ongewijzigd wordt gebruikt in applicaties. Het maakt elke applicatie kwetsbaar voor een MITM aanval.
De threat is nog steeds aanwezig in versie 1.9.6-beta. De threat werd gedocumenteerd a.d.h.v codefragmenten van de Android API's, maar omdat de oorzaak van de threat zich bevindt in de Javascript interface van de plugin, zijn alle ondersteunde platformen vatbaar voor de *vulnerability*.
Het is onduidelijk hoeveel apps gebruik maken van de CodePush plugin. Op de website van CodePush[10] worden er al zeker 75 apps als showcase voorgesteld. Het is mogelijk dat deze apps de vulnerability bevatten omdat er geen instructies of documentatie is gegeven omtrent de trustAllHosts-optie.
Ten slotte wordt er aangeraden dat de vulnerability eenvoudig wordt opgelost door de ontwikkelaars van de CodePush plugin door de trustAllHosts parameter op lijn 75 figuur B.1 bijlage B op 'false' te zetten. Op deze manier is de plugin veilig voor gebruik.

3.3.2 Dynamic update

De tweede plugin is cordova-plugin-dynamic-update[11] geschreven door Lee Crossley. In essentie vervult deze plugin dezelfde functie als de CodePush plugin. Bij het opstarten van de app wordt nagegaan bij een server of er een update beschikbaar is. Vervolgens wordt de gezipte update gedownload en uitgepakt. Er bestaat echter wel een gigantisch verschil tussen deze plugin en de CodePush plugin. Dit kan het eenvoudigst geïllustreerd worden a.d.h.v. de totale grootte van beide plugins: de dynamic update plugin is 9.58KB groot terwijl de CodePush plugin maar liefst 3.48MB groot is. Dit is een verschil van ongeveer factor 370. Dat getal valt de verklaren doordat de CodePush plugin beschikt over een heleboel extra functionaliteiten in vergelijking met de dynamic update plugin. Bovendien maakt CodePush gebruik van 8 andere plugins om al deze extra functies te implementeren terwijl de dynamic update plugin slechts bestaat uit 1 Javascript bestand.
Deze extreme vergelijking tussen de twee update plugins wordt gemaakt om aan de lezer duidelijk te maken dat er een heleboel dergelijke update plugins beschikbaar zijn in de plugin store. De

[10] https://microsoft.github.io/code-push/
[11] https://github.com/leecrossley/cordova-plugin-dynamic-update

complexiteit van deze plugins ligt ruwweg tussen de dynamic update plugin die enkel voorziet in basisfunctionaliteit en de CodePush plugin die een arsenaal aan extra functies en ondersteuning met zich meebrengt.

Vervolgens gaan we de veiligheid van de dynamic update plugin na en wordt aanvallersmodel uit hoofdstuk 2.2 toegepast.
Type 1 aanvallen kunnen niet besproken worden omdat de plugin geen ondersteuning heeft om updates te uploaden naar de update server. De plugin vereist enkel dat de URL van de update server hardgecodeerd wordt in de code. Dit geeft de ontwikkelaar de vrijheid over de implementatie van de server. Dit kan zowel een voordeel als een nadeel zijn. Sommige ontwikkelaars maken liever hun eigen server dan de server van een ander te gebruiken en dan kan dit als een voordeel worden beschouwd. Vele anderen beschouwen dit echter als een nadeel omdat dit voor extra werk zorgt.
De bescherming tegen de type 2 aanval hangt af van de updateserver die wordt gebruikt. Indien dit een HTTPS-server is, dan is deze plugin even goed beveiligd tegen een MITM aanval als de CodePush plugin. Dit komt omdat beide plugins gebruik maken van dezelfde onderliggende Android API's. Deze zorgen ervoor dat het certificaat van de updateserver wordt gecontroleerd en dat de communicatie geëncrypteerd is.

3.4 Besluit

Cordova applicaties bezitten het voordeel t.o.v. klassieke applicaties dat ze updates kunnen uitvoeren zonder tussenkomst van een app store. Deze hot code updates zorgen ervoor dat updates onmiddellijk de gebruiker bereiken. Dit is zeer voordelig voor applicaties die vaak updates ondergaan.
De hot code updates zijn echter wel beperkt tot het updaten van de platformonafhankelijke code. Voor Cordova applicaties is dit zeer ruim want praktisch de volledige applicatie is hieruit opgebouwd. Enkel de plugins van een applicatie kunnen niet worden geüpdatet. Deze bestaan namelijk deels uit native code en om deze code te updaten, is er een hercompilatie nodig. Om plugins te wijzigen, toe te voegen of te verwijderen zal er dus toch tussenkomst van een app store nodig zijn.

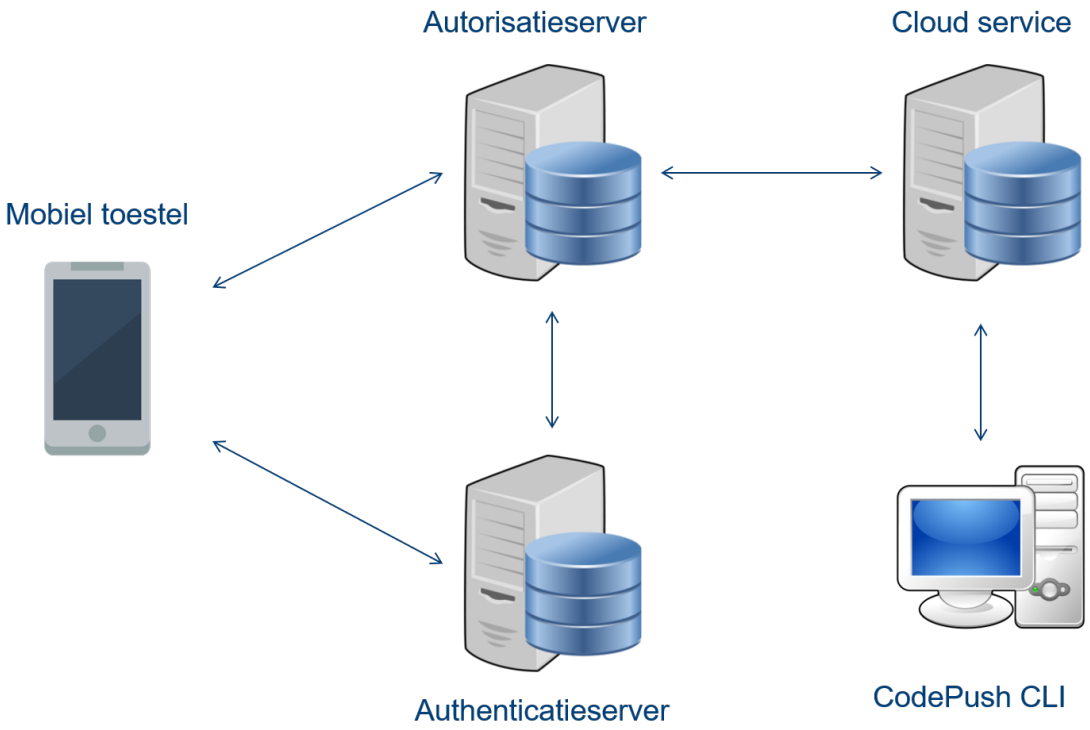
Er zijn reeds een hele hoop update plugins ter beschikking, elk met z'n eigen set functionaliteiten. Het is aan de ontwikkelaar om te kiezen welke functies hij ter beschikking wilt. In het algemeen is elke update plugin veilig voor gebruik als de verbinding tussen plugin en updateserver beveiligd is en als de update op een veilige manier op de server wordt geplaatst.
Om aan de praktische doeleinden van dit eindwerk te voldoen, wordt er in het verdere systeem gebruik gemaakt van de CodePush plugin. De cloud service is ideaal voor de opslag van apps en updates. Daarenboven is de command line interface om de apps en updates te beheren een zeer praktische tool die bijdraagt aan de totale waarde van het systeem.

Toch kan de plugin niet op zichzelf voldoen aan de gestelde eisen om gepersonaliseerde updates te verspreiden. Daarom zal de CodePush plugin uitgebreid worden met enkele elementen zodat het een heus updatesysteem wordt. Het volgende hoofdstuk geeft het ontwerp weer van dit systeem.

Hoofdstuk 4

Ontwerp

Dit hoofdstuk behandelt het ontwerp van het systeem. Zowel het volledige systeem als alle onderdelen afzonderlijk worden besproken. Nadien worden de ontwerpbeslissingen besproken in een apart hoofdstuk. In figuur 4.1 wordt er een globaal overzicht gegeven van de structuur van het systeem. Het systeem kan opgedeeld worden in 5 grote onderdelen: een authenticatieserver, een autorisatieserver, de cloud service, de CLI en het mobiel toestel dat gebruik maakt van het systeem.



Figuur 4.1: Algemene structuur van het systeem

4.1 De afzonderlijke componenten

In dit onderdeel worden alle componenten nader toegelicht, maar de effectieve implementatie ervan wordt pas in hoofdstuk 5 gegeven. Op figuur 4.1 pagina 17 kan men zien dat de authenticatieserver zowel met het mobiele toestel als met de autorisatieserver in verbinding staat. Dit komt omdat de authenticatieserver een duale rol speelt in het systeem. Enerzijds zal de gebruiker van de app zich moeten authenticeren bij de server door in te loggen. Anderzijds zal de autorisatieserver bij elke binnenkomende request de authenticatieserver raadplegen om de afzender van de request te authenticeren.
De rol van updateserver in het systeem wordt vervuld door de cloud service van CodePush, zie hoofdstuk 5.1. In hoofdstuk 2 werd enkel de term updateserver gebruikt. Nu vastligt dat er verder wordt gebouwd op de CodePush plugin, zullen de termen cloud service en updateserver door elkaar worden gebruikt, maar ze stellen dezelfde component voor. Deze server bevat de apps en updates. Daarenboven bevat de server een heleboel functies waarmee ontwikkelaars apps en updates kunnen beheren op de server. Ontwikkelaars kunnen deze functies gebruiken door de CodePush CLI te gebruiken.
Vervolgens is er de autorisatieserver die dienst doet als proxyserver voor de updateserver. De autorisatieserver zal bepalen welke requests doorgestuurd worden naar de updateserver en welke niet. Deze server bevat ook de logica om te bepalen welke gebruiker welke update ontvangt. Dit mechanisme wordt in het volgende onderdeel uitgelegd tesamen met de volledige werking van het systeem.
De communicatie tussen de mobiele toestellen van de gebruiker en de autorisatieserver wordt geregeld door de CodePush plugin.
Dit is genoeg informatie voor de lezer om de rest van het ontwerp te begrijpen. In hoofdstuk 5 wordt de uitwerking van het ontwerp gegeven.

4.2 Het volledige systeem

De belangrijkste functie van het systeem is de juiste updates verspreiden naar de juiste gebruikers. Figuur A.2 op pagina 38 geeft dit proces weer in een sequentiediagram. Het sequentiediagram maakt duidelijk tussen welke servers er gecommuniceerd wordt en in welke volgorde dit gebeurt. De volle pijlen stellen synchrone berichten voor terwijl de smalle pijlen asynchrone berichten voorstellen. Op de figuur zijn er 10 communicatiestappen te zien. Hieronder volgt een beschrijving van elke stap.

1. De eerste maal dat de app wordt opgestart, moet de gebruiker zich authenticeren op de applicatie. Vervolgens stuurt de app deze logingegevens op naar de authenticatieserver ter controle. Daar worden de gegevens vergeleken met de gegevens uit de database. Indien de logingegevens geldig zijn, wordt er een token toegekend aan de gebruiker. Dit token wordt opgeslagen in de database van de authenticatieserver en wordt opgestuurd naar de app. De app slaat het token op in het geheugen van het toestel waardoor de gebruiker zich niet telkens opnieuw moet inloggen. Hetzelfde token zal ook meegestuurd worden bij elke request naar de autorisatieserver.

2. Als de applicatie beschikt over een accesstoken, stuurt de CodePush plugin een updateCheck-bericht om na te gaan of de app up-to-date is. Dit werd al eerder behandeld in hoofdstuk 3.3.1 tijdens de bespreking van de CodePush plugin. Het enige verschil is dat alle berichten

nu niet rechtstreeks naar de updateserver gaan, maar onderschept worden door de autorisatieserver.

3. De autorisatieserver controleert het token van elk binnenkomend bericht bij de authenticatieserver. Bij een geldig token, zal de authenticatieserver het corresponderende user id terugsturen naar de autorisatieserver. De autorisatieserver bevat een database met een tabel dat de user id van een gebruiker linkt aan een bepaald privilegeniveau. De database bevat nog een tweede tabel dat elk privilegeniveau linkt aan een bepaalde deployment key. De deployment keys kunnen nu dynamisch toegevoegd worden aan de berichten op basis van de identiteit van de gebruiker. Op deze manier wordt het mogelijk om per gebruiker te bepalen welke update hij of zij ontvangt. Dit is de kern van het flexibele updatesysteem.

4. Als de request zowel geauthenticeerd als geautoriseerd is, dan wordt de request met zijn oorspronkelijke parameters en de dynamisch toegevoegde deployment key doorgestuurd naar de updateserver. In de updateserver wordt vervolgens bepaald of er een update beschikbaar is. In het geval er een beschikbaar is, dan bevat het antwoord van de updateserver een URL vanwaar de update kan gedownload worden.

5. Dit bericht zal opnieuw langs de autorisatieserver gaan, maar deze keer zullen er geen controles gebeuren en zal het bericht simpelweg doorgestuurd worden naar het mobiele toestel.

6. Eenmaal dit bericht aankomt, beschikt de plugin over de download URL. De plugin zal nu het download proces initialiseren door verbinding te maken met de URL.

7. De downloadrequest wordt opnieuw onderschept door de autorisatieserver zoals in stap 3 en opnieuw wordt de identiteit van de afzender gecontroleerd bij de authenticatieserver.

8. Eenmaal geauthenticeerd, wordt de request doorgespeeld naar de updateserver.

9. De updateserver maakt een zip-bestand van de update en stuurt deze terug.

10. Alle data gaat opnieuw via de autorisatieserver, maar net als in stap 5 gebeurt er opnieuw geen controle. Ten slotte bereikt de update het mobiele toestel en kan de update geïnstalleerd worden.

4.3 Ontwerpbeslissingen

In dit onderdeel worden de belangrijkste ontwerpsbeslissingen nader toegelicht. De eerste grote beslissing is de invoer van de autorisatieserver als proxyserver. In hoofdstuk 3.3 werd het duidelijk voor de lezer dat de CodePush plugin rechtstreeks in verbinding staat met de cloud service. Deze structuur kon echter niet voldoen aan de eisen om verschillende updates naar verschillende gebruikers te verspreiden. Dit komt doordat de CodePush plugin een unieke deployment key toekent aan apps en deze key gebruikt om de juiste update in de cloud service te identificeren. Dit heeft als beperking dat elke app slechts één key heeft en dus enkel de update kan binnenhalen die geassocieerd is met deze deployment key. Door een autorisatieserver in te voegen, is het mogelijk om de deployment key dynamisch toe te voegen en zo verschillende updates per app mogelijk te maken. Dit is de kern van het flexibele updatesysteem. Vervolgens worden de verschillende versies elk als een app geregistreerd bij de updateserver. Zo beschikt elke versie over een unieke deployment key. Vanuit het perspectief van de updateserver staan er enkel apps op de server. De server zelf

houdt niet bij welke versies tot dezelfde app behoren. Dit mechanisme wordt volledig voorzien door de autorisatieserver. Bovendien is de autorisatieserver transparant voor de update plugin en zijn er minimale veranderingen gemaakt aan de CodePush plugin om de communicatie via de autorisatieserver te doen verlopen. Op deze manier kan de plugin nog steeds updates ondergaan zonder dat er veranderingen nodig zijn aan het systeem.
Door de centrale ligging en functie van de autorisatieserver kan deze beschouwd worden als zowel een *policy descision point (PDP)* als een *policy enforcement point (PEP)* in het netwerk. Een PEP is een element in een netwerk dat beslissingen uitvoert op basis van een gegeven policy. Een voorbeeld van zo'n dergelijke beslissing is een gebruiker toegang weigeren tot een bepaalde resource. In tegenstelling tot de PEP die de beslissingen uitvoert, is de PDP het element die de beslissingen maakt. In het netwerk van het flexibele updatesysteem doet de autorisatieserver dienst als beide. De server gaat bij elke request na wie de afzender is en beslist op basis van de identiteit van de afzender wat er met de request zal gebeuren.

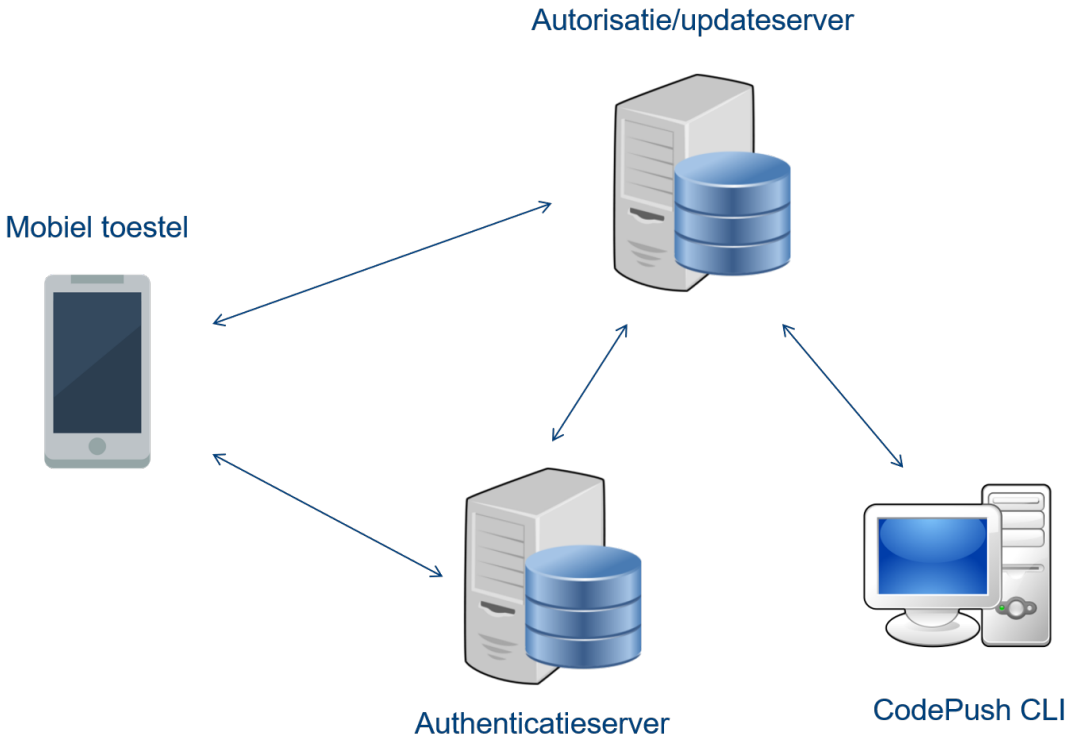
De tweede structurele beslissing is de scheiding van authenticatie- en autorisatieserver. Deze beslissing is voornamelijk gebaseerd om de privacy van de gebruiker te garanderen. Zo bevat de autorisatiesever geen enkele gevoelige data van de gebruiker. Deze data wordt enkel bewaard in de database van de authenticatieserver. Dit maakt het ook mogelijk om de authenticatieserver extra te beveiligen indien dit gewenst is.

Een derde beslissing is het gebruik van tokens als authenticatiemiddel. Alhoewel het gebruik van tokens niet van structureel belang is, heeft het wel een invloed op de communicatie binnen het systeem. Bovendien wordt het systeem er een stuk gebruiksvriendelijker door. Het zou namenlijk onhandig zijn als de gebruiker zich bij elke request zou moeten authenticeren door het invoeren van zijn logingegens. Het gebruik van een token laat toe dat de gebruiker zich eenmalig authenticeert, waarna de token kan herbruikt worden als identiteitsbewijs bij de volgende momenten waar authenticatie vereist is.
De tokens worden automatisch gegenereert door de authenticatieserver a.d.h.v. een *random token generator* telkens een gebruiker zich aanmeldt bij de applicatie. De tokens worden in een database opgeslagen en hebben een beperkte geldigheidsduur. Als een gebruiker zich uitlogt bij de app, dan wordt het accesstoken zowel uit het geheugen van de app als uit de database verwijderd.

4.4 Alternatief ontwerp

Een ander ontwerp dat werd overwogen is te zien in figuur 4.2 pagina 21. Het verschil tussen dit alternatieve ontwerp en het gekozen ontwerp is afwezigheid van de extra autorisatieserver.
Dit ontwerp werd tijdelijk overwogen omdat het systeem ook is uitgetest a.d.h.v. een lokale kopie van de CodePush cloud service, zie hoofdstuk 5.1.1 pagina 22. Bij dit ontwerp zou de updateserver zodanig aangepast zijn dat de server ook over de nodige autorisatiefuncties beschikt. De totale complexiteit van dit ontwerp is lager omdat het systeem nu slechts bestaat uit twee servers.
Dit weegt echter niet op tegen de aanpassingen die nodig zijn aan de updateserver en aan de database ervan. De grote hoeveelheid modificaties zou ervoor zorgen dat de server niet meer kan genieten van toekomstige updates en *bugfixes* die ervoor worden vrijgegeven. Na een update zouden er opnieuw veel aanpassingen moeten gebeuren zodat de server weer correct functioneert. Daarom werd dit ontwerp al snel overtroefd door het finale ontwerp waarbij de autorisatie gebeurt op een aparte server. Op deze manier moet de updateserver geen veranderingen ondergaan en kan de server gebruikt worden als black box. Zo kunnen nieuwe versies van de updateserver

probleemloos in het systeem worden geïntegreerd op voorwaarde dat de functionaliteit en interface van de server niet gewijzigd is.
Bovendien kan de cloud service van CodePush uiteraard niet aangepast worden waardoor dit ontwerp op niets slaat indien men wenst gebruik te maken van de echte cloud service.



Figuur 4.2: Alternatief ontwerp van het systeem

Hoofdstuk 5

Realisatie

Dit hoofdstuk focust op de implementatie van het ontwerp dat in het vorige hoofdstuk uitvoerig werd besproken. Eerst wordt elke onderdeel van het systeem afzonderlijk besproken. Daarna worden de andere gebruikte technologieën zoals de communicatie nader toegelicht.

5.1 De servers

De drie servers zijn geschreven in Javascript en worden uitgevoerd in de run-time omgeving van Node.js. Daarenboven werd er ook gebruik gemaakt van het Express[12] framework. Dit framework voorziet eenvoudige en flexibele functionaliteiten voor Node.js web applicaties. De belangrijkste functie die Express met zich meebrengt is het gebruik van *routes* en de *middleware* die eraan kan gekoppeld worden. Deze termen worden duidelijk na het lezen van de volgende paragrafen.
Hieronder volgt er een beschrijving van de afzonderlijke servers. Deze beschrijving houdt onder meer in van waar de broncode van de servers afkomstig is. De server kan zelf geschreven zijn of kan als geheel van het internet komen. Indien afkomstig van het internet, is het de bedoeling om de server als black box te gebruiken. Maar zoals zal blijken in de volgende paragrafen was dit niet 100% realiseerbaar.

5.1.1 Updateserver

In hoofdstuk 3.3 werd ervoor gekozen om een systeem te bouwen rond de CodePush plugin. We weten reeds dat de plugin gebruik maakt van de cloud service van Microsoft om updates te verspreiden. Het systeem is ontworpen zodat deze cloud service kan dienen als updateserver.
Om beter inzicht te krijgen in de werking van de cloud service werd ook gewerkt met een quasi-exacte kopie[13] van deze cloud service. De server bevat dezelfde functionaliteiten als de cloud service en is perfect compatibel met de CodePush plugin. Het codepakket van deze server zal ook te vinden zijn op de GitHub link die bij dit eindwerk hoort.

5.1.2 Authenticatieserver

Om de authenticatieserver te implementeren werd gebruik gemaakt van een Oauth2 server[14] met een Express wrapper. Oauth is een open standaard dat gebruikt wordt voor autorisatie. Via Oauth

[12] https://expressjs.com/
[13] https://www.npmjs.com/package/code-push-server
[14] https://www.npmjs.com/package/node-oauth2-server

kan de gebruiker aan een programma, website of server toegang geven tot zijn of haar gegevens die opgeslagen zijn bij een derde partij. Het Oauth protocol wordt in dit systeem op een lichtjes alternatieve manier gebruikt. Zo kan de gebruiker zich eenmalig authenticeren bij de Oauth2 server, waarna deze een token ontvangt dat de applicatie kan hergebruiken om zich te authenticeren in de toekomst.
In hoofdstuk 4.1 werd reeds gezegd dat de authenticatieserver een dubbele rol speelt in het systeem. De server moet in staat zijn om tokens te genereren en om tokens te controleren op geldigheid. De eerste functionaliteit is standaard aanwezig op de Oauth2 server, maar de tweede functie om tokens te controleren echter niet. Daarom werd de server aangepast zodat het ook deze tweede functie bevat. Figuur 5.1 op pagina 23 toont de twee routes die gebruikt worden door het systeem. Routes bepalen de manier waarop de server reageert op binnenkomende request. De grant()-methode en alle functies die de methode gebruikt, waren reeds aanwezig op de server. De check()-methode werd zelf toegevoegd om de functionaliteit van de server te vervolledigen.

```
17  // Handle token grant requests
18  app.all('/oauth/token', app.oauth.grant());
19
20  //Handle accesstoken checks from the authorization server
21  app.get('/oauth/accesstoken', app.oauth.check());
```

Figuur 5.1: Twee Express routes in de authenticatieserver

5.1.3 Autorisatieserver

De autorisatieserver is in tegenstelling tot de andere twee servers voor het grootste deel zelf geschreven. Er werd opnieuw gebruik gemaakt van een Node.js server met Express framework zodat het eenvoudig is om met de server te communiceren. Figuur A.3 op pagina 39 geeft het index.js bestand van de autorisatieserver weer. Het indexbestand is de kern van elke Node.js applicatie. In de code vallen duidelijk drie routes te onderscheiden: updateCheck, downloadUrl en reportStatus. Enkel via deze routes kan gecommuniceerd worden met de server. Berichten naar andere URL's zullen een foutmelding opgooien bij de verzender.
Elke route zal eerst de methode middleware.checkToken uitgevoerd worden. Deze methode haalt het accesstoken uit het bericht en controleert de validiteit ervan bij de authenticatieserver. Enkel als het accesstoken geldig is, zal de volgende functie uitgevoerd worden. Deze functies verschillen van route tot route. Zo zal er bij de updateCheck ook een deployment key gekozen worden alvorens het bericht wordt doorgestuurd naar de updateserver. Dit gedrag werd reeds beschreven in hoofdstuk 4.2.

5.2 Communicatie

Om de communicatie binnen het systeem te beveiligen, gaan alle berichten over HTTPS. Node.js beschikt standaard over de functionaliteit om een HTTPS server te starten. Om dit correct te doen werken, is het belangrijk om ervoor te zorgen het certificaat van de server en de eventueel bijhorende certificaatketting correct worden geconfigureerd op de server. In een realistische omgeving zal de server een uniek URL-adres hebben en zal het certificaat getekend zijn door een erkende certificaatautoriteit (CA). Omdat mobiele toestellen standaard een lijst met de bekendste CA's bevatten, zal het besturingsysteem geen probleem hebben om in dit scenario de certificaatketting

te valideren. Een lokale opstelling daarentegen zorgt voor enkele complicaties, zie hoofdstuk 6.3 pagina 31.

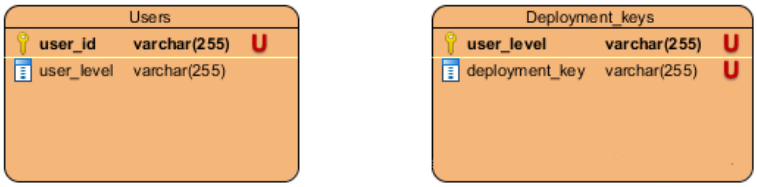
Er werd reeds aangehaald dat alle servers worden uitgevoerd als Node.js webapplicaties en voorzien zijn van het Express framework. Hieronder volgt een voorbeeld hoe de communicatie verloopt via het Express framework.
Stel dat de autorisatieserver zich bevindt op *https://mijnautorisatieserver.com*, dan kan de route in figuur 5.2 aangesproken worden door een GET-bericht te sturen naar *https://mijnautorisatieserver.com/welkom?clientUniqueId=abc123*. De parameters van het bericht zullen beschikbaar zijn via het 'req' object. Vervolgens kunnen deze gebruikt worden in de functie die bij de route hoort en kan een antwoord gestuurd worden via het 'res' object. Dit wordt geïllustreerd in figuur 5.2 waar de clientUniqueId uit de URL wordt gehaald en wordt vergeleken met een bepaalde waarde. De uitkomst van deze conditie bepaalt welk antwoord terug wordt gestuurd. De communicatie kon ook geïmplementeerd worden zonder het Express framework, maar vanwege de eenvoudigheid van het framework en het feit dat dit framework reeds werd gebruikt op de updateserver en authenticatieserver, is ervoor gekozen om ook de autorisatieserver te voorzien van het framework. Op deze manier gebeurt alle communicatie binnen het systeem op dezelfde manier.

```
24  app.get('/welkom', function(req, res) {
25
26      var clientUniqueId = req.query.clientUniqueId;
27
28      if(clientUniqueId=='abc123'){
29        res.send("Welkom Jonas");
30      }
31      else{
32        res.send("Onbekend ID");
33      }
34
35  });
```

Figuur 5.2: Voorbeeld van een route met het Express framework

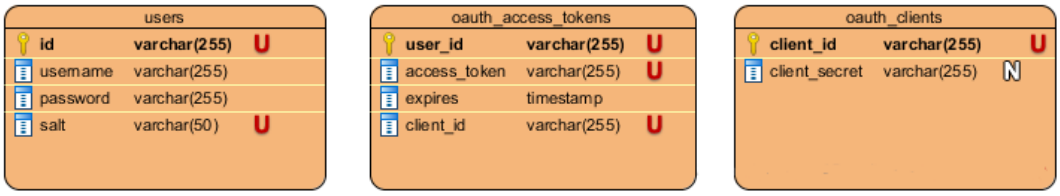
5.3 Databases

Bij elke server hoort een database om de nodige gegevens op te slaan. In de volgende paragrafen worden de drie databases besproken. Bij elke database is er een schematische voorstelling en korte uitleg gegeven.
De eenvoudigste database is deze van de autorisatieserver, ze bestaat uit slechts twee tabellen. De schematische voorstelling is terug te vinden op figuur 5.3 op pagina 25. Het nut van de eerste tabel is om aan elke gebruiker een priviligeniveau toe te kennen. Dit niveau kan bv. afhangen van de betalingsoptie van de gebruiker. Wie meer betaalt, krijgt een hoger privilegeniveau toegewezen en heeft dus ook recht op meer functionaliteit van de applicatie. In de tweede tabel wordt elk priviligeniveau gelinkt aan een deployment key. Merk op dat er in deze tabellen geen persoonlijke informatie over de gebruikers wordt opgeslagen. De gebruikers worden geïdentificeerd door een uniek user_id. Dit user_id wordt enkel in de authenticatie database gelinkt aan een persoon en zijn/haar privégegevens.



Figuur 5.3: Schema autorisatie database

De authenticatie database is iets groter, maar is nog altijd vrij beperkt qua complexiteit. Ze bestaat uit drie tabellen die te zien zijn op figuur 5.4 pagina 25. In de eerste tabel *users* worden de gegevens van de gebruiker opgeslagen. De belangrijkste kolommen zijn de *id*, *username* en het *password*. Dit zijn de gegevens die nodig zijn om een gebruiker te authenticeren. Deze tabel zou eenvoudig kunnen worden uitgebreid met extra gegevens van de gebruikers naargelang de noden van de applicatie waarvoor het systeem wordt gebruikt. De tweede tabel slaat accesstokens op en linkt ze aan user_id's van een gebruiker en aan een client_id. Het client_id dient om mobiele toestellen te identificeren. De functionaliteit van het client_id wordt echter niet gebruikt in het systeem. De Oauth2 server bevatte een functie om slechts een bepaalde set toestellen te accepteren, maar deze werd overschreven zodat alle toestellen worden aanvaard. Ten slotte heeft elk accesstoken een vervaldatum. Deze bepaalt hoe lang het accesstoken kan gebruikt worden door de applicatie. Het SQL schema dat bij dit server pakket hoort, bevat echter wel een groot veiligheidsrisico: het wachtwoord van de gebruikers werd in leesbare tekst opgeslagen in de tabel 'users'. Daarom werd de tabel aangepast door de kolom 'salt' toe te voegen. Ook de kolom 'password' werd aangepast en bevat nu een hash van de de waarde 'username:password:salt'. Het algoritme dat werd gebruikt om de hash uit te voeren is SHA-256[15]. Op deze manier worden de wachtwoorden van de gebruikers nooit blootgesteld aan het oog van iemand met toegang tot de database.



Figuur 5.4: Schema authenticatie database

Ten slotte is er de database van de lokale updateserver, niet te verwarren met de database van de cloud service. Al kan verwacht worden dat deze er gelijkaardig uitziet. De database is veel groter en complexer dan de andere twee. Dit is te verwachten aangezien de database alle applicaties en daarbij ook alle updates van elke applicatie bevat. Verder bevat de database ook gegevens over de applicatieontwikkelaars.
Omdat een schematische voorstelling van de database niet overzichtelijk is, is deze niet gegeven. Figuur A.4 op pagina 39 in de bijlage geeft echter wel een duidelijk en eenvoudig overzicht van alle tabellen in de database. De meeste tabellen staan in functie van de CLI voor de ontwikkelaars. In hoofdstuk 3.3 werden enkele functies van de CLI reeds kortstondig besproken. De CLI heeft echter een arsenaal aan functies die de ontwikkelaar kan gebruiken. Voor een gedetailleerde beschrijving van alle functies wordt verwezen naar de documentatie van CodePush[16].

[15] https://en.wikipedia.org/wiki/SHA-2
[16] https://microsoft.github.io/code-push/docs/cli.html

Hoofdstuk 6

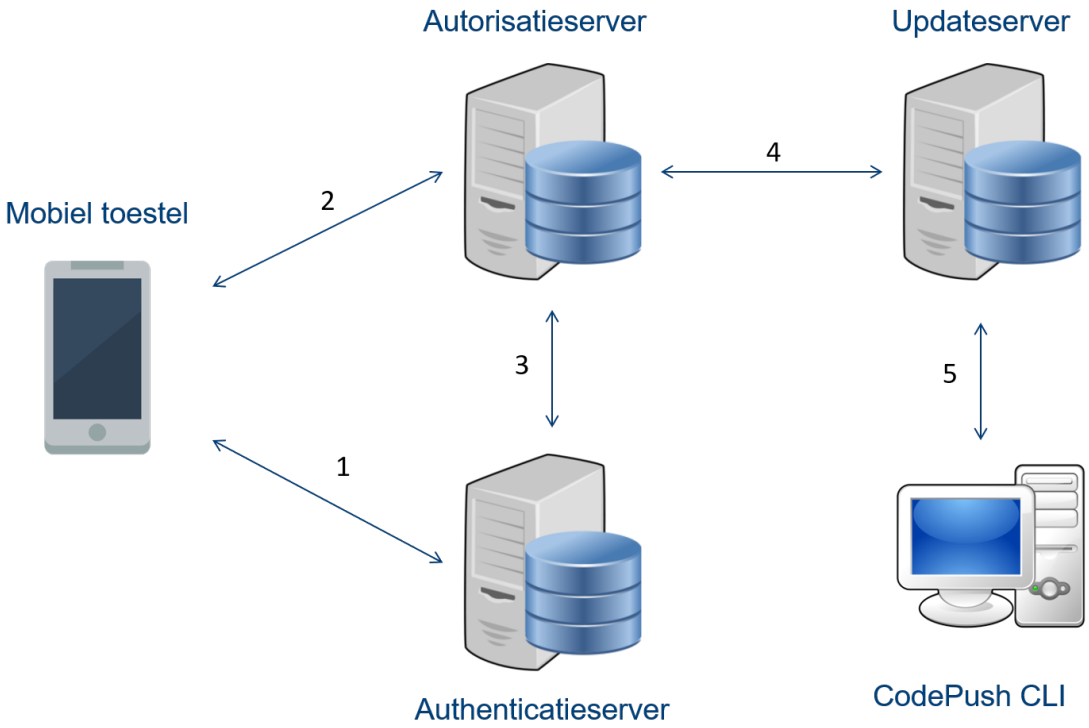
Evaluatie

Dit hoofdstuk bestaat uit drie onderdelen. Het eerste onderdeel is een veiligheidsanalyse van het volledige systeem. Daarna volgt er een sterkte-zwakteanalyse over het geheel. En het hoofdstuk wordt afgesloten door een aantal algemene opmerkingen i.v.m. het resultaat.

6.1 Veiligheidsanalyse van het systeem

Dit onderdeel bevat een uitvoerige analyse van de veiligheid van het systeem. Eerst zullen de mogelijke veiligheidsrisico's per verbinding worden vastgesteld, om daarna de geïmplementeerde veiligheidsmaatregelen te geven. Zo zal het duidelijk worden tegen welke aanvallen het systeem beveiligd is, en tegen welke het nog vatbaar is. Om de analyse te visualiseren wordt er opnieuw gebruik gemaakt van de globale structuur van het systeem. Deze is nogmaals gegeven in figuur 6.1 pagina 27. Op de figuur heeft elke verbinding een cijfer gekregen. Hieronder worden de verbindingen in deze volgorde besproken.
De eerste verbinding is die tussen het mobiel toestel en de authenticatieserver. Communicatie tussen deze elementen neemt plaats wanneer de gebruiker zich inlogt op de applicatie. De applicatie zal vervolgens de logingegevens verifiëren bij de authenticatieserver en er wordt een accesstoken toegekend aan de gebruiker.
Omdat er persoonlijke logingegevens worden verstuurd, kan een aanvaller proberen meeluisteren op het kanaal om de gegevens te achterhalen. Indien hij hierin slaagt, kan hij zich voordoen als de gebruiker zonder dat het systeem dit kan opmerken. Zo heeft de aanvaller mogelijks toegang tot functies van de applicatie waar hij geen recht op heeft.
Dit soort misbruik kan vermeden worden indien de aanvaller de logingegevens van de gebruikers niet kan achterhalen.

Eens de applicatie beschikt over een geldig accesstoken, kan er verbinding worden gemaakt met de autorisatieserver (verbinding twee op de figuur) om na te gaan of er een update beschikbaar is. Bij elke request naar de autorisatieserver wordt het accesstoken in een autorisatie header toegevoegd.
Er kan hier dezelfde logica worden gevolgd als bij de eerste verbinding. Indien er een aanvaller meeluistert, kan hij het accesstoken achterhalen en zelf gebruiken. Opnieuw zou de aanvaller in staat zijn zich voor te doen als een andere gebruiker en dus in staat zijn misbruik te maken van het systeem.
Deze verbinding houdt echter nog een veel groter risico in. De verbinding is kwetsbaar voor een MITM-aanval. Deze aanval is te vergelijken met de type 2 aanval besproken in hoofdstuk 2.2. Een



Figuur 6.1: Globale structuur van het systeem

aanvaller zou zich kunnen voordoen als de autorisatieserver en wachten tot een applicatie verbinding wil maken. Op deze manier kan hij de applicatie laten geloven dat er een update beschikbaar is en vervolgens een geïnfecteerde update doorsturen naar de applicatie. De volgende keer dat de app wordt opgestart, wordt het kwaadaardige script uitgevoerd en is het toestel geïnfecteerd.
Vanuit het standpunt van de gebruiker is dit een veel ernstiger probleem dan het misbruik maken van de logingegevens. Het toestel van de gebruiker kan namenlijk zeer gevoelige informatie bevatten dat niet in de handen mag vallen van personen met slechte bedoelingen.
Dit probleem kan verholpen worden door de aanwezigheid van eindpunt authenticatie.

Verbinding 3 zorgt ervoor dat de autorisatieserver de accesstokens van binnenkomende requests kan verifiëren bij de authenticatieserver. De autorisatieserver stuurt de accesstokens via verbinding 3 op naar de authenticatieserver ter controle. Ook hier kan een aanvaller meeluisteren om accesstokens te achterhalen. Deze verbinding houdt echter een veel groter risico in dan de vorige: indien een aanvaller hier meeluistert, zal hij niet enkel het accesstoken van één gebruiker te weten komen, maar zal hij uiteindelijk de accesstokens van alle gebruikers kunnen achterhalen. Dit komt omdat de autorisatieserver het accesstoken van elke binnenkomde request (dus van alle gebruikers) controleert bij de authenticatieserver. Dit kan leiden tot een grootschalig misbruik van het systeem.
Er is hier een duidelijke nood aan encryptie zodat een aanvaller niet kan meeluisteren.

Verbinding 4 is onderdanig aan dezelfde MITM-aanval als verbinding 2. Een aanvaller kan de autorisatieserver om de tuin leiden door zich voor te doen als de updateserver en een geïnfecteerde update terugsturen naar de autorisatieserver als deze een update verwacht. Vervolgens wordt de update doorgestuurd naar het mobiel toestel, waarna deze wordt geïnstalleerd. Als verbinding 2

beveiligd zou zijn, dan zou het mobiel toestel de binnenkomde data blindelings vertrouwen. Dit is echter niet veilig omdat de geïnfecteerde update wordt geïnjecteerd in verbinding 4 zonder dat het mobiel toestel dit kan detecteren.
Om een MITM aanval onmogelijk te maken, moeten alle verbindingen tussen het mobiel toestel en de updateserver beveiligd zijn. Op deze manier kan een aanvaller nergens een valse update injecteren.

Verbinding 5 wordt gebruikt om de apps en updates in de database van de updateserver te beheren. De belangrijkste functie is het pushen van updates naar de updateserver. De werking hiervan werd reeds uitgelegd in hoofdstuk 3.3.1. Daar werd aangetoond dat dit proces enkel veilig is als de communicatie over een beveiligd kanaal gebeurt zodat een aanvaller zich niet kan voordoen als ontwikkelaar.

6.1.1 besluit

Omdat er bij elke verbinding wel een mogelijk veiligheidsrisico optreedt, was het noodzakelijk om het volledige systeem te beveiligen door overal gebruik te maken van HTTPS. Dit zou er namelijk voor zorgen dat alle communicatie wordt geëncrypteerd zodat aanvallers niet meer kunnen meeluisteren op de verbindingen. Bovendien biedt HTTPS ook eindpunt authenticatie waardoor MITM aanvallen onmogelijk worden.
In hoofdstuk 3.3.1 werd duidelijk dat de CodePush plugin reeds HTTPS gebruikt om de communicatie tussen de componenten te beveiligen. Er restte dus enkel nog de toegevoegde servers te beveiligen. Dit werd gerealiseerd door beide servers te configureren als HTTPS-servers. Dit was eenvoudig te doen m.b.v. de standaard functies van Node.js. Op deze manier gebeurt alle communicatie automatisch via een beveiligde HTTPS-verbinding. Bijgevolg zijn alle communicatiekanalen tussen de servers en de applicatie beveiligd en is er geen misbruik mogelijk.

6.2 Sterkte-zwakteanalyse

6.2.1 Sterktes

Aangezien doorheen het hele project rekening is gehouden met de veiligheid van de afzonderlijke componenten en het hele systeem, verdient het veiligheidsaspect vernoemd te worden in dit onderdeel van de analyse.
Het project ging van start met de zoektocht naar een geschikte update plugin. De plugin moest voldoen aan bepaalde functionele voorwaarden en moest bovendien ook goed beveiligd zijn op zichzelf zodat het kon gebruikt worden in het systeem. De CodePush plugin was hiervoor de ideale kandidaat. Waar de plugin mankeerde aan functionaliteit, werden twee extra servers ingevoerd om de ontbrekende functies aan te vullen. Daarenboven werd dit gerealiseerd zonder in te boeten aan de veiligheid van het volledige systeem.

Het tweede aspect dat thuishoort in dit onderdeel is de eenvoudigheid van het systeem. Hiermee worden twee zaken bedoeld.
Ten eerste wijst dit op de eenvoudige interacties tussen de elementen in het systeem. Zo gebeurt alle communicatie a.d.h.v. eenvoudige HTTP-berichten via het Express framework. Bovendien zijn er maar een beperkt aantal berichten waarop de servers reageren. Deze zijn gedefinieerd door de routes van het Express framework. Op deze manier volgt het systeem altijd hetzelfde eenvoudige patroon. Dit patroon kan u nogmaals vinden op figuur A.2 in de bijlage op pagina 38.

Ten tweede kan men hieronder de graad van herbruikte elementen verstaan. Er werd vooropgesteld dat het beter is om bestaande componenten te gebruiken i.p.v. zelf componenten te creëeren indien dit mogelijk is. In hoofdstuk 5 werd het reeds in detail duidelijk op welke manier de elementen werden geïmplementeerd. Hieronder worden alle componenten en hun implementatie nog eens kort opgesomd.

- Update plugin: CodePush update plugin
- Authenticatieserver: Oauth2 server
- Autorisatieserver: eigen creatie
- Updateserver: CodePush cloud service

De autorisatieserver vormt het centrale element van het systeem. Door deze zelf te schrijven, was het mogelijk om voor de andere componenten bestaande code te gebruiken. Op deze manier is er zo goed mogelijk voldaan aan het voornemen om bestaande componenten te herbruiken om het systeem op te bouwen.

Als derde kan aangehaald worden dat de ingevoerde autorisatie- en authenticatieserver *stateless* zijn. Dit houdt concreet in dat er nergens bepaalde toestanden worden opgeslagen. Zo wordt er bv. niet bijgehouden over welke appversie de gebruikers beschikken. De plugin gaat simpelweg na of er een update beschikbaar is a.d.h.v. een bericht naar de autorisatieserver telkens de app opstart. Dit proces gebeurt compleet statenloos.
Dit houdt in dat de meerdere servers probleemloos kunnen samengeclusterd worden om de schaal van het systeem uit te breiden. Dit is een niet te onderschatten voordeel als de last die het systeem ondervindt te groot wordt.

De gevolgen van het gebruik van het flexibele updatesysteem zijn ook te merken aan de kant van de client. We hebben te maken met applicaties waarbij bepaalde functies vergrendeld zijn voor sommige gebruikers. Welke functies vergrendeld zijn voor welke gebruiker hangt bv. af van de betalingsoptie die de gebruiker heeft gekozen. Iemand die meer betaalt, zal logischerwijs meer functies ter beschikking hebben.
De huidige applicaties van dit type realiseren dit door de functies te vergrendelen aan de kant van de client (m.a.w. in de app zelf). Dit zorgt voor extra complexiteit voor de app ontwikkelaar. Door gebruik te maken van het flexibele updatesysteem, verdwijnt deze extra last voor de ontwikkelaar. Het updatesysteem zorgt ervoor dat enkel de functies waarop een gebruiker recht heeft, worden geïnstalleerd op het toestel. Er hoeft geen code meer vergrendeld te worden in de app zelf. Op deze manier kan een ontwikkelaar zich maximaal focussen op de frontend ontwikkeling van zijn applicatie en hoeft hij zich niet bezig te houden met authenticatie en autorisatie.
De 'betalende' functies onthouden aan de serverkant brengt nog een tweede voordeel met zich mee. Zo wordt het onmogelijk voor *hackers* om de applicatie op het toestel te kraken met als doel toegang te krijgen tot de betalende functies van de app. De functies kunnen niet gekraakt worden omdat ze simpelweg niet aanwezig zijn in de applicatie.

Ten slotte is er nog een voordeel dat te maken heeft met de manier waarop updates worden uitgerold. Het uitrollen van de laatste update voor een app is altijd een riskant gebeuren. Men weet namelijk nooit 100% zeker op voorhand of de nieuwe versie stabiel is en niet zal crashen. Daarom kan het belangrijk zijn om de laatste versie van een app eerst te laten testen door een groep gebruikers alvorens de update globaal te verspreiden.

Het flexibele updatesysteem biedt exact deze mogelijkheid en nog meer. Er kan per gebruiker bepaald worden welke update hij of zij ontvangt. Zo kan de database van de authenticatieserver uitgebreid worden met gegevens van de gebruiker zoals het besturingssysteem op zijn/haar toestel, versie van dat besturingssysteem, geografische locatie, etc. Op deze manier wordt het eenvoudig om testgroepen te maken voor bepaalde updates. Het wordt zo ook mogelijk om bepaalde updates net niet uit te rollen naar bepaalde gebruikers omdat bv. de update niet compatibel is met de versie van het besturingssysteem dat ze gebruiken.
Het flexibele updatesysteem maakt m.a.w. een gestructureerde uitrol van updates mogelijk waar dit eerder niet mogelijk was.

6.2.2 Zwaktes

De eerste zwakte die wordt beschouwd, is er een van structurele aard en heeft betrekking tot de autorisatieserver. Zoals eerder gezegd, draait de autorisatieserver in de run-time omgeving van Node.js. Node.js maakt gebruik van *single-threading* met *non-blocking I/O calls* waardoor er duizenden verbindingen tegelijkertijd mogelijk zijn zonder een context switch. Dit design is bedoeld voor applicaties waar er nood is aan een hoge graad van *concurrency*. Op zich lijkt hier nog altijd niets mis mee, de server zal nog steeds goed functioneren ookal wordt het aantal gelijktijdige requests zeer hoog. Het probleem houdt zich echter schuil in de aard van de requests. Stel er is een nieuwe update beschikbaar voor een applicatie die zich op de updateserver bevindt. Wanneer de gebruikers die app opstarten, zal er een updateCheck-bericht gestuurd worden naar de server om na te gaan of er een nieuwe update is. Deze berichten vormen geen probleem aangezien dit korte berichten zonder *payload* zijn. Maar aangezien er wel degelijk een update beschikbaar is, zal er voor elke gebruiker ook een update volgen. En omdat alle communicatie via de autorisatieserver gaat, zullen ook alle downloads via deze server gaan. Hier kan de bandbreedte van de autorisatieserver voor een *bottleneck* zorgen. Dit fenomeen zal echter enkel voorvallen als het totaal aantal gebruikers van alle applicaties op de updateserver zeer groot wordt en/of als er meerdere updates tegelijk worden vrijgegeven.
Dit probleem kan opgelost worden door een *handoff* mechanisme te implementeren. Dit houdt in dat de download van de updates niet meer via de autorisatieserver gaat, maar rechtstreeks tussen de cloud service en het mobiel toestel gebeurt. Op deze manier wordt de grootste last van de autorisatieserver weg genomen. Het probleem kan natuurlijk ook opgelost worden door een extra autorisatieserver te plaatsen. In het onderdeel 'Sterktes' werd reeds gezegd dat de servers stateless zijn en dat ze probleemloos kunnen geclusterd worden.

Als tweede zwakte kan aagehaald worden dat het niet gelukt is om de herbruikte componenten volledig als black box te implementeren. Bij elke component was er wel een kleine aanpassing nodig waardoor de componenten niet volledig probleemloos kunnen genieten van de updates en bugfixes die de ontwikkelaars ervoor vrijgeven. Zo werd de CodePush plugin aangepast zodat het accesstoken in een header wordt meegestuurd bij elk bericht dat wordt verzonden. Daarnaast werd er bij de authenticatieserver een methode toegevoegd om de accesstokens te controleren. Ook al zijn dit maar kleine aanpassingen, ze maken het een pak lastiger om de herbruikte te updaten.

Ten derde past de graad van complexiteit ook in dit onderdeel. Het gewone Cordova updatesysteem met enkel de updateplugin werd uitgebreid met twee extra servers. Dit verhoogt de complexiteit van het geheel aanzienlijk. In hoofdstuk 4.4 werd er een systeem voorgesteld met slechts één extra server, maar de lagere complexiteit van dit systeem woog niet op tegen de problemen en nadelen die het met zich meebracht.

Ten slotte is er nog een zwakte die inherent is aan Cordova applicaties. Cordova apps kunnen op meerdere platformen gebruikt worden omdat ze opgemaakt zijn uit niet-gecompileerde platformonafhankelijke code zoals Javascript. Het is dan ook niet extreem moeilijk om als hacker de broncode van een Cordova applicatie te achterhalen. Op deze manier zou een hacker eenmalig kunnen betalen voor de duurste versie van de app zodat deze legaal wordt gedownload naar zijn mobiel toestel. Daarna kan de hacker de broncode achterhalen en aanpassen waardoor hij het loginmechanisme kan omzeilen. Nu is de hacker is staat alle functies van de app te gebruiken zonder problemen. Daarenboven is hij nu ook in staat om zijn gemodificeerde versie van de app te verspreiden.
Dit is een probleem dat optreedt bij alle Cordova applicaties en kan niet verholpen worden. Het gebruik van tijdsgelimiteerde accesstokens beperkt echter wel de schade die dit met zich mee brengt. Nadat een accesstoken is verlopen, moet er namelijk opnieuw worden ingelogd. Maar omdat het loginmechanisme werd omzeild, is dit niet meer mogelijk. De gekraakte app kan dus geen updates meer ontvangen via het flexibele updatesysteem omdat na verloop van tijd niet meer beschikt over een geldig accesstoken. De gekraakte versie zal echter wel ongelimiteerd kunnen worden gebruikt.

6.3 Opmerkingen

Dit onderdeel bevat een aantal algemene opmerkingen i.v.m. het systeem.
Ten eerste dient er wat gezegd te worden over de implementatie van het systeem. Zoals gezegd in hoofdstuk 5.1.1 kan het systeem ook gebruik maken van de kopie van de cloud service. Deze server kan via Node.js eenvoudig lokaal gestart worden. Zo kan de volledige opstelling lokaal draaien door Node.js te gebruiken. Dit is echter niet hoe het systeem in een realistische omgeving zou gebruikt worden, maar omdat er toch veel lokaal is gewerkt om te servers te analyseren, volgt er hieronder toch wat uitleg.
Lokaal werken maakt het handig om het systeem te onderzoeken, maar brengt enkele beperkingen met zich mee. Als de servers als HTTPS-servers worden geconfigureerd, dan moeten de servers beschikken over de juiste certificaten. Daarbij moet er gewerkt worden met zelfgetekende certificaten omdat de servers lokaal draaien en dit had twee gevolgen:

1. Er moet een eigen certificaatautoriteit gecreëerd en geïnstalleerd worden op de mobiele toestellen die wensen gebruik te maken van het systeem.

2. De ontwikkelaarsfuncties van de updateserver werken niet. Dit komt omdat de server maar een schijnbare kopie is van de cloud service van CodePush en dus geen perfecte kopie is. Indien je de updateserver wil gebruiken, kan je de server niet configureren als HTTPS-server en zal sommige communicatie dus niet beveiligd zijn. Dit vormt echter geen probleem aangezien volledig lokaal wordt gewerkt.

De tweede beperking houdt in dat het mobiel toestel met hetzelfde WiFi-netwerk moet verbonden zijn als de computer waarop de servers draaien. Anders zal de applicatie geen verbinding kunnen maken met de lokale ip-adressen van de servers.
Dit zijn problemen die niet voorkomen in een realistisch scenario waar alle servers geregistreerd zijn bij een Internet Service Provider en voorzien zijn van een certificaat dat ondertekend is door een erkende certificaatautoriteit.

Naast de implementatie dient er ook iets gezegd te worden over de herbruikbaarheid van het systeem. Met andere woorden, wat zijn de complicaties als het systeem zou gebruikt worden in een ander scenario? Bij een verandering van scenario moeten drie elementen aangepast worden:

- Uiteraard zal er zelf een nieuwe applicatie moeten geschreven worden. Het prototype dat bij dit eindwerk hoort, dient enkel ter demonstratie.
- De inhoud van de update database. De applicatie met zijn eventueel verschillende versies moet worden gepusht naar de updateserver m.b.v. de CLI.
- De inhoud en eventueel de structuur van de authenticatie database. De gebruikers en hun gegevens moeten manueel toegevoegd worden aan deze database.

Hoofdstuk 7

Prototype

7.1 Inleiding

Het prototype is een Cordova applicatie dat instaat bij een thuiszorgsituatie. Er zijn meerdere versies van de app beschikbaar met elk een toenoemende graad van functionaliteit. Het doel van het prototype is om aan te tonen dat het flexibele updatesysteem in staat is om de juiste updates naar de juiste gebruikers te sturen. Bovendien toont het ook de eenvoudigheid aan om een andere versie van de app te installeren wanneer er van gebruiker wordt veranderd of als een gebruiker beslist te upgraden naar een andere versie.

7.2 De applicatie

Het eerste scherm dat de gebruiker te zien krijgt, is een loginscherm. Dit scherm is voor alle versies van de app hetzelfde. In dit scherm dient de gebruiker zich te authenticeren a.d.h.v. zijn logingegevens. Deze gegevens worden vervolgens gecontroleerd bij de authenticatieserver. Na een geslaagde authenticatie bevindt de gebruiker zich op een scherm waar een overzicht te zien is van alle sensoren die in verbinding staan met de applicatie. Hier treden er reeds verschillen tussen de verschillende versies. Men kan de menubalk zichtbaar maken door naar rechts te vegen vanaf de linkerkant van het scherm. Welke andere schermen er te bereiken zijn via de menubalk hangt ook af van de versie van de app. Figuur A.6 op pagina 41 in de bijlage toont hoe de schermen van de versies verschillen t.o.v. elkaar. Bovendien bevat de expert-versie van de app nog extra schermen zoals degene die te zien zijn in figuur A.7 op pagina 42 in bijlage A.

7.3 De sensoren

Ter volledigheid dient er ook iets gezegd te worden over de sensoren die in verbinding staan met het prototype. De opstelling is gemaakt met een LoRaWAN Rapid Development Kit.[17] Een set sensoren en een LoRa Radio Module werden verbonden met een SODAQ Arduino[18] module. Dankzij de radio module kan de Arduino microprocessor gegevens van de verbonden sensoren doorsturen naar het Internet of Things (IoT) netwerk van Proximus. Eens doorgestuurd, zijn de gegevens beschikbaar via de AllThingsTalk Maker RESTful API[19]. Ten slotte kan de app de data van de

[17] http://www.allthingstalk.com/lorawan-rapid-development-kit
[18] https://www.arduino.cc/
[19] http://docs.allthingstalk.com/developers/api/rest/

sensoren verkrijgen door gebruik te maken van deze API.
Al deze zaken gebeuren op een relatief eenvoudige manier. Zowel de code om de data van de Arduino naar het IoT-netwerk te sturen, als de berichten van de app naar de API zijn eenvoudig gestructureerd. Het enige waarbij moet opgelet worden is de *duty cycle* waarmee de Arduino de gegevens doorstuurt naar het IoT-netwerk. Om het IoT-netwerk van proximus niet te veel te belasten, dient de *duty cycle* van de doorgestuurde data rond de 5% te liggen. Omwille van deze reden, stuurt de Arduino de data van een sensor slechts om het half uur door naar het IoT-netwerk. Zowel de code van alle versies van het prototype als de code die op de Arduino staat is te vinden op de GitHub link die bij dit eindwerk hoort.

7.4 Zelf een applicatie maken

Indien een ontwikkelaar een applicatie wil maken die gebruik maakt van het flexibele updatesysteem, dan zijn er enkele zaken die hij in achting moeten nemen.
De app moet op een bepaalde manier gestructureerd worden zodat het login- en updateproces correct gebeurt. Figuur A.5 op pagina 40 in de bijlage geeft a.d.h.v. een flowchart weer wat de algemene structuur van zo'n applicatie moet zijn. Enkel als deze structuur wordt gevolgd, zal de app op de juiste manier het updatesysteem gebruiken. De ontwikkelaar heeft zelf de vrijheid om deze structuur te programmeren. Het prototype is enkel een voorbeeld van hoe dit kan geïmplementeerd worden. Hieronder volgt een korte uitleg van hoe de belangrijkste elementen uit figuur A.5 op pagina 40 in de bijlage werden geïmplementeerd.
Om het accesstoken op te slaan, werd gebruik gemaakt van een opslag API speciaal voor Cordova applicaties genaamd *LocalStorage*.[20] Met LocalStorage kunnen sleutelwaardeparen eenvoudig worden opgeslagen. LocalStorage kan enkel strings opslaan en heeft slechts een capaciteit van 5MB. Dit vormt echter geen probleem om het accesstoken op te slaan. Daarenboven is de LocalStorage API zeer eenvoudig om te gebruiken. Er waren nog andere opslag API's beschikbaar, maar deze hadden een veel grotere complexiteit en dat was niet nodig voor enkel de opslag van het accesstoken.
De updateCheck wordt uitgevoerd door de CodePush plugin. Tijdens deze check wordt ook de validiteit van het accesstoken nagegaan. Indien deze niet meer geldig is, moet opnieuw worden ingelogd. Het loginmechanisme bestaat uit een asynchroon bericht met de logingegevens naar de authenticatieserver. Pas wanneer een nieuw accesstoken wordt teruggestuurd, zal het updateproces verdergaan.
De initialisatie en installatie van de download worden beide uitgevoerd door de CodePush plugin. Hier kan kan een ontwikkelaar niet veel aan veranderen. Men kan wel parameters meegeven aan de CodePush commando's om de installatieopties te configureren. Alle opties zijn te vinden in de documentie[21] van CodePush. De optie die hier dient de besproken worden is de *installMode*. Deze optie kreeg als parameter 'IMMEDIATE' zodat de app na het updaten onmiddellijk heropstart. Dit is belangrijk zodat de gebruiker altijd de juiste versie van de app gebruikt.

[20] https://cordova.apache.org/docs/en/latest/cordova/storage/storage.html
[21] https://microsoft.github.io/code-push/docs/cordova.html

Hoofdstuk 8

Besluit

De doelstelling van dit eindwerk was de ontwikkeling van een systeem dat in staat is om flexibele updates te verspreiden voor Cordova applicaties. De flexibele updates zijn nodig voor applicaties die bestaan uit meerdere versies. Hierbij is het belangrijk is dat de gebruikers zeker beschikken over de juiste versie en bijgevolg ook updates krijgen voor enkel die versie. Daarenboven moest het updatesysteem een correct gebruik van de applicatie garanderen waarbij geen misbruik mogelijk is.

Hieronder worden alle stappen van het eindwerk en de belangrijkste mechanismes nogmaals bondig opgesomd.
In de eerste stap werd er een kwalitatief onderzoek gedaan naar Cordova update plugins. Dit onderzoek zou bepalen op welke plugin werd verdergebouwd. De CodePush plugin van Microsoft kwam als meest gekwalificeerde plugin uit het onderzoek. Maar de plugin was echter niet voldoende om aan de doelstelling te voldoen. Er ontbraken nog elementen om de updates te personaliseren.
Om de updates bij de juiste gebruikers te krijgen, was er nood aan authenticatie langs de kant van de gebruiker. Om dit te implementeren is gebruik gemaakt van een authenticatieserver die los staat van de andere servers. De authenticatieserver is verbonden met een database dat de (login)gegevens van alle gebruikers bevat. Bovendien wordt er ook gebruik gemaakt van een authenticatiemechanisme met accesstokens zodat de gebruikers een minimale last ondervinden van het authenticatieproces.
Naast authenticatie was er ook nood aan autorisatie zodat de juiste gebruiker de juiste updates ontvangt. Hiervoor werd een aparte autorisatieserver ingeschakeld. De database van de autorisatieserver linkt gebruikers aan een bepaald privilegeniveau. Bij elke request dat de autorisatieserver onderschept, wordt de identiteit van de requester bij de authenticatieserver gecontroleerd. Nadat de identiteit correct is vastgesteld, kan de autorisatieserver bepalen welke update de gebruiker dient te ontvangen. Dit gebeurt door de juiste deployment key dynamisch toe te voegen aan de request en deze door te sturen naar updateserver. In de updateserver wordt de juiste update geselecteerd a.d.h.v. de deployment key uit de request. De updateserver antwoordt ten slotte met een download-URL waarvan de update kan worden gedownload.
Indien het bovenstaande en het besluit uit hoofdstuk 6.1 in beschouwing worden genomen, dan kan er besloten worden dat er werd voldaan aan de doelstellingen uit hoofdstuk 1.2.

Het resultaat van dit eindwerk is een veilig en flexibel updatesysteem dat geschikt is om gepersonaliseerde updates te verspreiden naar Cordova applicaties. Alhoewel dit systeem een hogere graad van complexiteit heeft t.o.v. het klassieke updatesysteem met enkel de update plugin, brengt

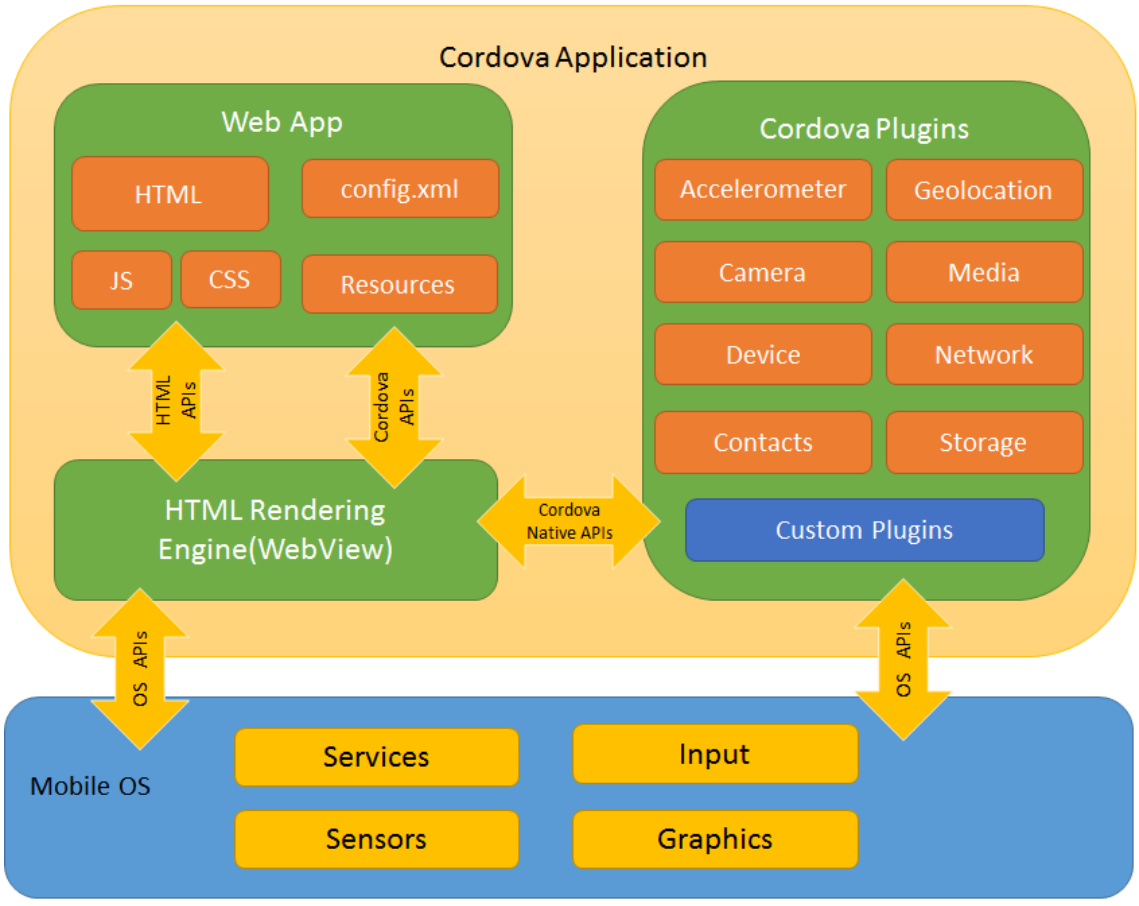
het flexibele updatesysteem toch een aantal voordelen met zich mee. Deze voordelen kunnen samengevat worden als volgende:

- Het updateproces is overal beveiligd.
- Het is een eenvoudig systeem: de communicatie is gestructureerd en er is een hoge graad aan herbruikte code.
- De extra servers zijn statenloos zodat het systeem heel goed schaalbaar is.
- De ontwikkeling van bepaalde types applicaties wordt eenvoudiger voor de ontwikkelaar.
- Het updatesysteem laat toe om updates op een gecontroleerde manier uit te rollen.

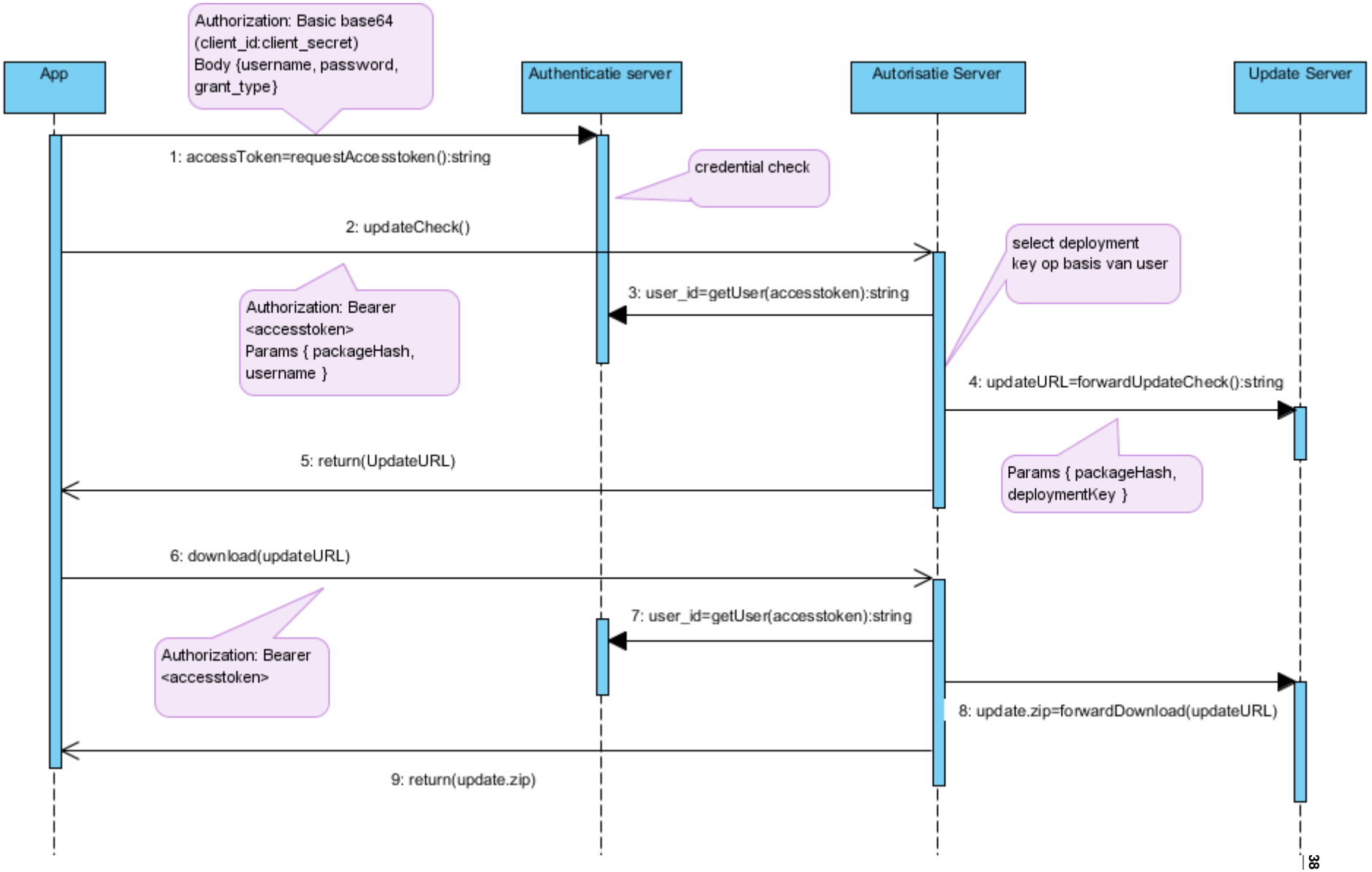
Deze voordelen maken het flexibele updatesysteem een solide alternatief voor de bestaande updatemechanismes.

Bijlage A

Extra figuren



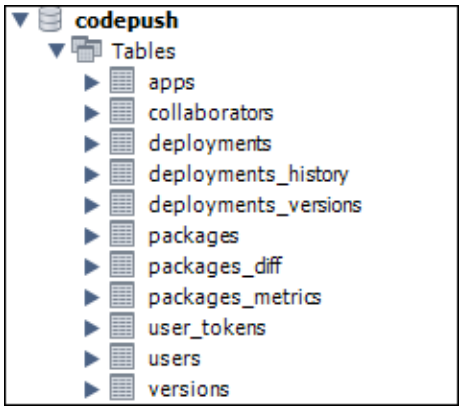
Figuur A.1: Structuur van een Cordova applicatie



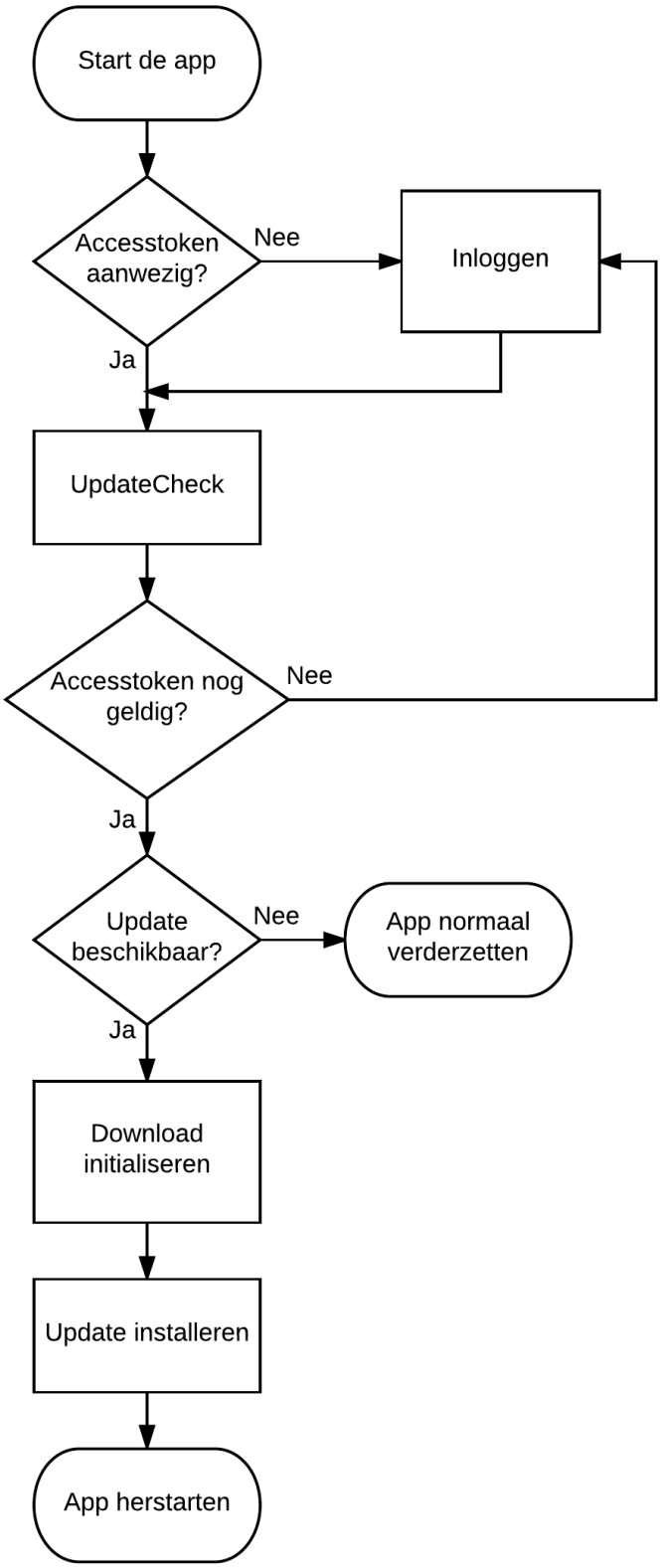
Figuur A.2: Sequentiediagram van het updateproces

```
var express = require('express');
var bodyParser = require('body-parser');
var request = require('request');
var XMLHttpRequest = require("xmlhttprequest").XMLHttpRequest;
var fs = require('fs');
var config = require("config.js"); //configuratiebestand waar de ip-adressen van de servers instaan
var https = require('https');

// Configuratie van de HTTPS server
var privateKey  = fs.readFileSync('auto_server.key');
var certificate = fs.readFileSync('auto_server.crt');
var passPhrase = 'abc123';
var sub_ca = fs.readFileSync('sub-ca.crt');
var options = {ca: sub_ca, key: privateKey, cert: certificate, passphrase: passPhrase};

var middleware = require('./lib/middleware');
var queryStringify = require('./lib/queryStringify.js');

var app = express();
var httpsServer = https.createServer(options, app);

app.use(bodyParser.urlencoded({ extended: true }));

app.use(bodyParser.json());

app.all('/updateCheck', middleware.checkToken, function(req, res) {...
});


app.all('/:downloadUrl', middleware.checkToken, function(req, res) {...
});


app.post('/reportStatus/:option', middleware.checkToken, function(req, res){...
});


httpServer.listen(3002);
console.log("[Index] authorization server listening on port 3002");
```

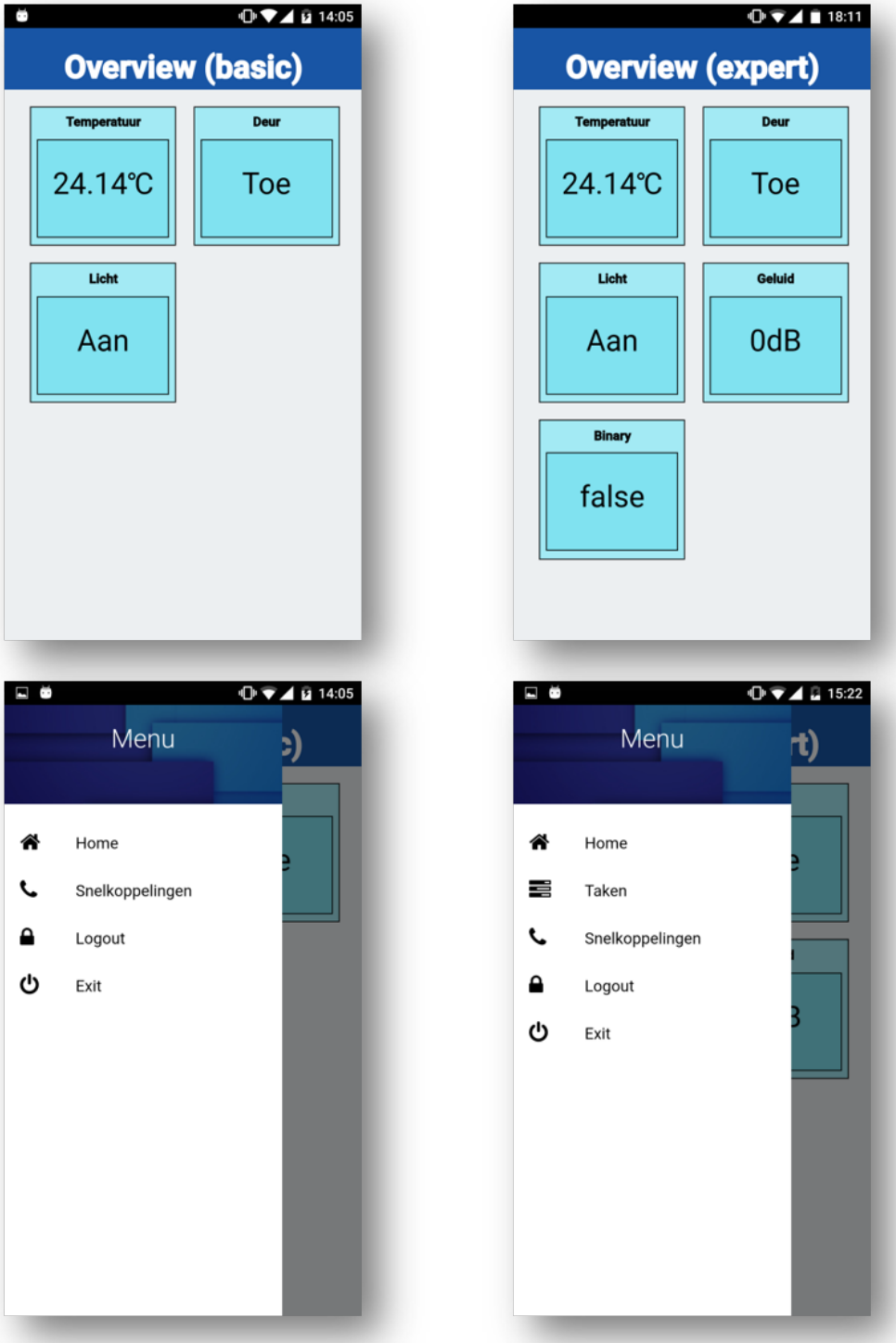
Figuur A.3: Index.js van de autorisatieserver



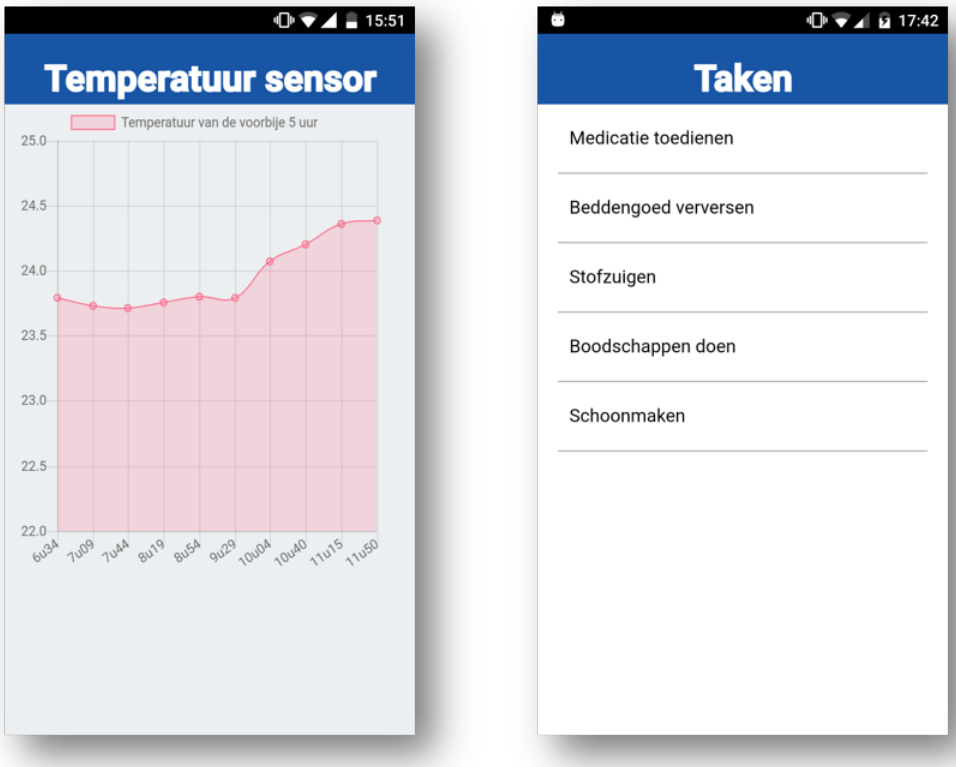
Figuur A.4: Overzicht database updateserver



Figuur A.5: Vereenvoudigd UML-diagram van het updateproces van de applicatie



Figuur A.6: Verschil tussen basic (links) en expert (rechts) versie van het prototype



Figuur A.7: Extra schermen van de expert-versie van het prototype

Bijlage B

Codefragmenten threat CodePush plugin

```
72                  this.currentFileTransfer.download(this.downloadUrl, cordova.file.dataDirectory + LocalPackage.DownloadDir +
73                      "/" + LocalPackage.PackageUpdateFileName, downloadSuccess, downloadError, true);
```

Figuur B.1: Codefragment remotePackage.js van de CodePush plugin

```
169 /**
170  * Downloads a file form a given URL and saves it to the specified directory.
171  * @param source {String}          URL of the server to receive the file
172  * @param target {String}         Full path of the file on the device
173  * @param successCallback (Function}  Callback to be invoked when upload has completed
174  * @param errorCallback {Function}    Callback to be invoked upon error
175  * @param trustAllHosts {Boolean} Optional trust all hosts (e.g. for self-signed certs), defaults to false
176  * @param options {FileDownloadOptions} Optional parameters such as headers
177  */
178 FileTransfer.prototype.download = function(source, target, successCallback, errorCallback, trustAllHosts, options) {
179     argscheck.checkArgs('ssFF*', 'FileTransfer.download', arguments);
180     var self = this;
181
182     var basicAuthHeader = getBasicAuthHeader(source);
183     if (basicAuthHeader) {...
190     }
191
192     var headers = null;
193     if (options) {...
198     }
199
200     if (cordova.platformId === "windowsphone" && headers) {
201         headers = convertHeadersToArray(headers);
202     }
203
204     console.log(headers);
205
206     var win = function(result) {...
227     };
228
229     var fail = errorCallback && function(e) {...
232     };
233
234     exec(win, fail, 'FileTransfer', 'download', [source, target, trustAllHosts, this._id, headers]);
235     //                                          args(0) args(1) args(2)        args(3)   arsg(4)
236 };
```

Figuur B.2: Codefragment filetransfer.js van de FileTransfer plugin

```
841                     if (useHttps && trustEveryone) {
842                         // Setup the HTTPS connection class to trust everyone
843                         HttpsURLConnection https = (HttpsURLConnection)connection;
844                         oldSocketFactory = trustAllHosts(https);
845                         // Save the current hostnameVerifier
846                         oldHostnameVerifier = https.getHostnameVerifier();
847                         // Setup the connection not to verify hostnames
848                         https.setHostnameVerifier(DO_NOT_VERIFY);
849                     }
```

Figuur B.3: Codefragment 1 filetranser.java van de FileTransfer plugin

```
621     /**
622      * This function will install a trust manager that will blindly trust all SSL
623      * certificates.  The reason this code is being added is to enable developers
624      * to do development using self signed SSL certificates on their web server.
625      *
626      * The standard HttpsURLConnection class will throw an exception on self
627      * signed certificates if this code is not run.
628      */
629     private static SSLSocketFactory trustAllHosts(HttpsURLConnection connection) {
630         // Install the all-trusting trust manager
631         SSLSocketFactory oldFactory = connection.getSSLSocketFactory();
632         try {
633             // Install our all trusting manager
634             SSLContext sc = SSLContext.getInstance("TLS");
635             sc.init(null, trustAllCerts, new java.security.SecureRandom());
636             SSLSocketFactory newFactory = sc.getSocketFactory();
637             connection.setSSLSocketFactory(newFactory);
638         } catch (Exception e) {
639             LOG.e(LOG_TAG, e.getMessage(), e);
640         }
641         return oldFactory;
642     }
```

Figuur B.4: Codefragment 2 filetranser.java van de FileTransfer plugin

Bijlage C

Wetenschappelijke samenvatting - Extended abstract

Hot code updates in Cordova applications

Jonas Semeelen[1]

[1] *Univeristy of Leuven, Ghent, Belgium*

January 11, 2018

In this dissertation, an alternative and flexible update system was developed for Cordova applications. To achieve this, a qualitative study was done on Cordova update plugins. The study determined the most suitable plugin which was then used as a foundation for the rest of the update system. The functionality of the plugin was expanded by adding an authorisation and authentication server to the system. This made it possible to identify users and thus send them personalised updates.

1 Introduction

Imagine a mobile application which has a wide variety of resources at its disposition. And let's say these resources are wireless sensors in a homecare environment. In order for the app to be able to receive data from the sensors, it will need specific code for each different type of sensor. Now try to imagine all of the different and unique homecare settings. Bear in mind that each homecare setting is designed accordingly to the needs of the resident. This results in a huge amount of different settings.

Before moving on, a bit more context is provided in order for the reader to be able to fully understand the goal of the thesis. The previously stated application is meant for relatives or acquaintances of the patient in need of homecare. The app will assist in monitoring the patient when the caretaker is unable to be physically present at the house. For instance, a trip sensor will alert the caretaker immediately if the patient has tripped somewhere in the house.

Of course the app and the use of the sensors won't come freely. A caretaker will have to pay in order to use the app's functionality. However, the caretaker will only want to pay for the functionality needed by his or her patient. Since the different amount of homecare settings is so high, the app provides different kinds of packages from which the caretaker can choose. This way the caretaker will only pay for the necessary set of functions.

Now we don't want the app itself to contain all of the functionality of all the different available packages. This would make the app too large and de app users would often have to download updates for pieces of code they can't even use because they don't pay for the package. We also don't want to distribute the different packages as seperate applications. This would only complicate matters if a caretaker decides to upgrade to another package. This would require the installation of a whole new app.

Conclusion: we want one flexible and lightweight application which functionality is based on the user. The actual goal of the thesis is to develop a system which can provide update support for said type of application. The newly developed updatesystem will be accompanied by a prototype homecare application in order to demonstrate the update system.

1.1 What to expect from this paper

This paper will discuss all of the steps taken to accomplish the goal. The first chapter is about Cordova and its update plugins. These are two key technologies used during the thesis. They get a whole chapter because the update plugin will be the foundation of the whole system. Next, the design of the system will be thoroughly discussed. This is followed by the implementation of each component and how these implementations were chosen. Finally, an analysis is made of the accomplished system.

2 Process

This chapter will discuss the same steps that are taken in the thesis. At the end of each step, a summary is given about the contribution of the step in the grand picture. After reading this chapter, it should be clear what the system can do and how it's made.

2.1 The update plugin

As you may or may not know, Cordova[1] is a framework which allows the creation of cross-platform applications using standard web technologies such as Javascript, HTML and CSS. Cordova uses plugins as an interface between the Javascript and the native components. Because of the application's hybrid properties, the actual app solely consists out of platform independent code. To update the app, this code can simply be replaced by new code because there are no native components present, thus no recompilation of code is required.

This property makes it possible to update Cordova apps without the interference of an app store. We call these kind of updates 'hot code updates'. There are currently a lot of plugins available in the Cordova plugin store which can perform hot code updates. Some of the existing plugins are written by professionals and continue to be enhanced up until this day. It was then decided that an existing plugin would be used in the system instead of a custom designed one. This way, the update plugin will be able to benefit from continuous updates and bugfixes released by its authors.

After carefully analyzing several plugins, the CodePush plugin by Microsoft came out on top. This plugin offers secure communication with a cloud service where the apps and updates are stored. The plugin grants a unique deployment key to each application that is newly registered in the cloud service. When an app contacts the cloud service to check for updates, the cloud service will use this deployment key to search for the correct application in its database. On top of this, the plugin also comes with a command line interface (CLI) to manage the apps in the cloud service.

Although CodePush seems like a flawless update plugin, a security vulnerability was found during its analysis. When the connection is made with the cloud service right before the download, a parameter is wrongfully given the value *true*. This sets in motion a chain of method calls resulting in a situation where every server certificate is blindly accepted. This makes the user's device vulnerable to a man-in-the-middle attack. The vulnerability is carefully documented in the thesis and a solution to the vulnerability has been suggested.

2.2 The design

After the choice of the update plugin, it was possible to design the rest of the system around the plugin since the plugin itself can't fulfill the needs of a flexible update system. The additional components are supposed to complete the missing functionality. In order to determine which update should be sent to which app user, two additional elements are needed: an authentication and an authorisation element. Using these elements it is possible to identify the requester of the update and authorise the request based on the identity.

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

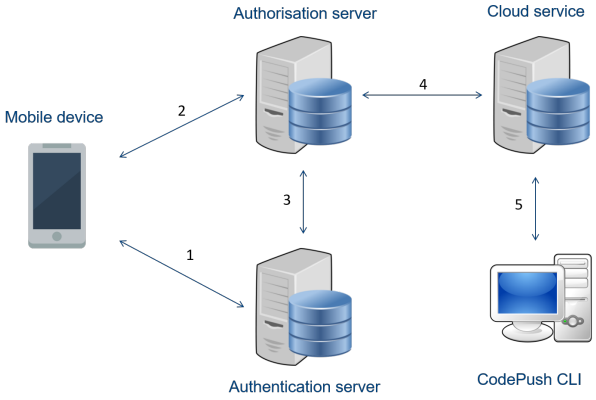


Figure 1: *Design of the system*

Several designs were considered but the design given in figure 1 proved to be the best. The biggest design decision that was made is the use of separate servers for the authentication and authorisation component. In the previous subsection it was already stated that the CodePush plugin comes along with a cloud service and an interface to use this cloud service. The main reason for the introduction of the extra servers is the preservation of the CodePush web server as a whole. By keeping this server unmodified it can be used as a black box and it can keep on receiving updates and bugfixes by the Microsoft development team. A second argument for using this design is the central location of the authorisation server. All the requests from the apps will be intercepted by this server. This allows the authorisation server to be a policy enforcement point (PEP). This a point in a network which executes decisions based on a certain policy. This part is the core of the whole flexible update system.

With each incoming request, the identity of the sender is extracted. Based on this identity, a deployment key is dynamically added to the request before forwarding it to the cloud service. By dynamically adding the deployment key instead of having it statically linked to the application, it is possible to dsitribute updates in a flexible manner. Because the decision policy is based on the user's identity, it is important that every request is properly authenticated. This is done by the authentication server.

The authentication server has two jobs. The first one is to authenticate users when they log in to the application. If the login is succesful, an access token will be granted. This token is used by the app in all subsequent requests made to the authorisation server. There, the token is extracted from the request and is sent to the authentication server for validation. This allows the PEP to do its job correctly.

2.3 The implementation

This subsection will clarify how each component of the system was implemented. It became clear in subsection

2.1 that the system will use the CodePush plugin. If the plugin is used alongside its intended cloud service and CLI, three out of five elements on figure 1 are already explained. Only the authorisation and authentication server remain.

By placing the authorisation server in the center of the system, it was possible to use an existing server which provides authentication mechanisms. The server that is used is an Oauth2 server[2]. Oauth is an open standard for authorisation. It allows users to grant permission to third parties to access their private data. This Oauth2 server already owns the functions needed to authenticate users. Because of this the Oauth2 server can also be used as a black box, just like the cloud service.

The authorisation server acts as an interface between the other components in the network which allows those components to be used as a black box. This was only possible by custom creating the authorisation server. Because of this, the authorisation server doesn't benefit from constant updates and bugfixes like the other elements. But this little disadvantage does not outweigh the benefits of the other components.

The authorisation server is a Node.js[3] web application which uses the Express[4] framework. The combination of these two technologies results in a lightweight and flexible authorisation server. The complexity of the server is relatively low since its only three jobs are verifying accesstokens, adding deployment keys to requests and forwarding these requests to the cloud service.

3 Analysis

3.1 Risks

The analysis of the system focuses mostly on the security aspect. The updatesystem has to be safe to use by apps which handle sensitive information. For this reason, every part of the system is checked to see if there are any vulnerabilities. The next paragraph will discuss the possible attacks, where they might happen in the system and what the risk involved is. Each connection in figure 1 is given a number to make the discussion easier.

The first connection is used to validate the user's login data. If an attacker is listening on this connection, he might find out the username and password. He will then be able to impersonate the user and fool the system.

The second connection is used whenever the update plugin sends a request to the authorisation server. This request contains the user's access token which is used for authentication purposes. If an attacker gets hold of this token, he can impersonate the user even without his or her login data. This is not quite as bad as the previous issue since the validity of an access token has a limited duration. This connection is also the prime target for a man-in-the-middle (MITM) attack. An attacker might try to inject a malicious update into this connection when the user is performing an update.

The third connection is susceptible to the same type of accesstoken-related risk, but on a larger scale. This connection is used when the authorisation server sends an access token to the authentication server to check its validity. This means that every token of every user gets sent over this connection. This could result in a large scale abuse of the system if the tokens have a long validity duration.

The fourth connection can also be targeted by a MITM attack. It's, just like connection two, a point in the system where malicious code might be injected.

Last but not least, the dangers to the fifth connection. This connection is used to upload and manage apps and updates in the cloud server. If an attacker were able to inject code into this connection, or if an attacker got hold of the logindata of an app developer, he would be able to directly upload malicious code into the cloud service. This would result in a global distribution of the infected update to all app users. This simply cannot happen.

3.2 Security measures

The previous section listed numerous security risks which could happen if the system isn't properly secured. Luckily measures were taken in order to secure the system and its users. In fact, there's one measure that would provide security versus all the risks from the previous section. This measure is to ensure all communication between all the components happens by HTTPS. Attackers would not be able to listen to the connections because now all data sent is encrypted and malicious code injections will be detected by the servers and the mobile device since HTTPS supports tamper detection.

To enforce this measure, all the servers in the system are configured as HTTPS servers. This entails secure communication on all connections.

4 Conclusion

The goal was to create a flexible update system for Cordova applications. In the first step, a qualitative research was done about Cordova update plugins. This research resulted in the qualification of the CodePush update plugin as the foundation of the system. Next, two servers were added for authentication and authorisation purposes. The correct combination of these elements results in a system which can provide individualised updates to Cordova app users.

[2] https://www.npmjs.com/package/node-oauth2-server
[3] https://nodejs.org/en/
[4] https://expressjs.com/

Bijlage D

Poster

KU LEUVEN

Jonas Semeelen
Academiejaar 2017-2018
Master of Science in de Industriële Wetenschappen: Electronica - ICT

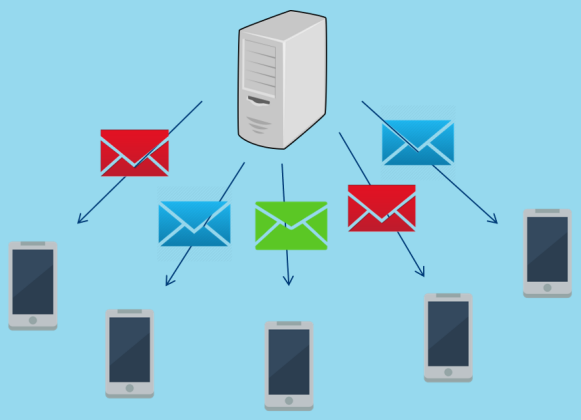
Hot code updates for Cordova applications

Goal

Creation of an updatesystem which is able to distribute personalized updates for Cordova applications

Why?

- Server-side restriction of app content
- Controlled distribution of new updates



APACHE CORDOVA™ + 101 011 1110

CodePush

First stage

Extensive qualitative research on the Cordova framework and update plugins

Result

CodePush update plugin as the foundation of the system

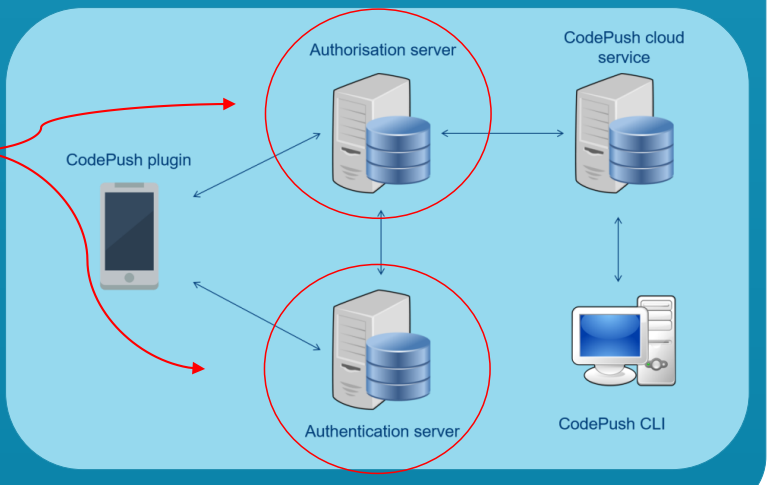
Second stage

Expansion of the CodePush plugin's functionality by adding two additional servers

Result

A flexible updatesystem for personalized updates which is:

- Scalable
- Secure
- Easy maintainable



11 Januari 2018

Promotor: Vincent Naessens
Co-promotor: Michiel Willocx

FACULTEIT INDUSTRIELE INGENIEURSWETENSCHAPPEN
TECHNOLOGIECAMPUS GENT
Gebroeders De Smetstraat 1
9000 GENT, België
tel. + 32 92 65 86 10
fax + 32 92 25 62 69
iiw.gent@kuleuven.be
www.iiw.kuleuven.be