# S3 – Securely Sharing Selfies

Student: Gert-Jan den Besten
Studentnr.: 835333020

Datum scriptie: 30 oktober 2016

Begeleider: dr. ir. H.L. (Hugo) Jonker
Begeleider: dr. ir. H.J.M. (Harrie) Passier
Examinator: prof. dr. T.E.J. (Tanja) Vos

T61327 - Afstudeerproject bachelor informatica

Open Universiteit Nederland, faculteit informatica

# Inhoudsopgave

# 1 Samenvatting

In deze scriptie wordt het S3 (Securely Sharing Selfies) systeem gepresenteerd. Het S3-systeem heeft als doel Context Driven Privacy bij het delen van selfies te faciliteren. Binnen de OU wordt onderzoek naar Context Driven Privacy uitgevoerd. Het systeem is gerealiseerd ter ondersteuning van dit onderzoek.

Omdat de term Context Driven Privacy impliceert dat men o.a. moet weten wat context nu eigenlijk is en dat men vervolgens iets met context kan doen, is een klein literatuuronderzoek uitgevoerd naar hoe men context kan bepalen en klassificeren. Dit literatuuronderzoek, plus de resultaten daarvan, is als een domeinanalyse binnen deze scriptie opgenomen.

Het S3-systeem bestaat uit twee componenten, een Android app (S3App) en een secure server (S3Server). S3App faciliteert het maken van encrypted selfies, waarbij de gebruiker zelf policies kan definiëren waaraan voldaan moet worden als men een selfie wil bekijken. S3Server beheert de sleutels die voor het ontsleutelen van betreffende selfies nodig zijn.

Het leeuwendeel van deze scriptie wordt gevormd door de beschrijving van het ontwerp, de gemaakte keuzes en de realisatie van de S3App en S3Server componenten. Deze beschrijving is aangevuld met suggesties voor zowel uitbreiding als verbetering van deze componenten. Deze suggesties hebben niet alleen betrekking op security, ook m.b.t. de softwareontwikkeling worden de nodige opmerkingen gemaakt.

Bij de realisatie van het S3-systeem is gebruik gemaakt van diverse -vaak impliciete- technieken, die of gangbaar zijn in de Java-wereld, of op dit moment in opmars zijn. Hier ligt een duidelijke link naar het vakgebied Software Engineering. Gemaakte keuzes worden in de lopende tekst toegelicht. De toegevoegde bijlagen bestaan uit meer gedetailleerde beschrijvingen -plus verwijzingen- m.b.t. deze technieken.

# 2 Introductie

*“Information Technology is considered a major threat to privacy because it enables pervasive surveillance, massive databases and lightning-speed distribution of information across the globe.”*
[Nissenbaum, 2010]

Mocht bovenstaand citaat in 2010 nog door sommigen als overdreven worden ervaren, dan is in 2013 -dankzij de onthullingen van Edward Snowden- daar definitief een einde aan gekomen. Uit die onthullingen bleek o.a. dat de National Security Agency (NSA) niet alleen ambassades en regeringsleiders van bevriende mogendheden afluistert en bespioneert [Webref: The Guardian, 2013-1]. Ook kwam een groots ingezette aanval op Belgacom door zowel de NSA als het Britse GCHQ [Webref: The Guardian, 2013-2] aan het licht.

Dit kan gevolgen hebben voor het vertrouwen van burgers in de overheden en bedrijven: Zijn onze privacygevoelige gegevens wel in veilige en kundige handen?

Daarnaast is de smartphone niet meer uit het dagelijkse leven weg te denken. Deze smartphones beschikken naast een forse verwerkingscapaciteit, ook over diverse sensoren waarmee o.a. de actuele locatie van de gebruiker kan worden vastgesteld. Anno 2016 is het heel gebruikelijk dat de gemiddelde smartphone continu met het internet verbonden is. Deze combinatie van factoren vormt ook een risico voor de privacy van smartphonegebruikers.

Het doel van het S3-project is ondersteuning te bieden aan onderzoek dat binnen de OU wordt uitgevoerd op het gebied van Context Driven Privacy. Daartoe is een Android app met bijbehorende server gerealiseerd. Deze app kan selfies en- en decrypten, waarbij de gebruiker zelf van toepassing zijnde policies kan definiëren.

De doelgroep van deze scriptie wordt gevormd door zowel de opdrachtgever als medestudenten. Vanwege de toegepaste technieken en het aantal gebruikte frameworks wordt, t.b.v. de studenten, de nodige aandacht aan implementatiedetails besteed. In de bijlagen worden veel implementatiedetails nog eens nader toegelicht. Daarbij wordt uitgegaan van basis Javakennis op het niveau van de OU-cursussen “Objectgeoriënteerd programmeren in Java” (1 en 2). Ook wordt basiskennis m.b.t. Servlets en Tomcat verondersteld. Hiervoor volstaat kennis op het niveau van de cursus “Webapplicaties: de serverkant”.

## 2.1 Formulering van de opdracht en vaststellen projectscope

De initiële opdrachtformulering voor dit project luidt:
*“Doel van dit project is om een Android app te ontwikkelen die foto's maakt en laat uitwisselen tussen gebruikers, op zo'n manier dat de ontvanger de foto alleen kan zien als aan de voorwaarden is voldaan (policy compliance). Verder delen van een foto kan alleen als de zender dat expliciet heeft toegestaan. Tegelijk met het maken van een foto wordt een policy-klasse bepaald aan de hand van de context (tijd, datum, nabije WiFis, nabije GSM antennes, etc.).*
*In eerste instantie gaat het om een grove klassering (prive, werk, publiek), die, afhankelijk van de voortgang, verder verfijnd kan worden (strand, bos, stappen, vakantie, werkbezoek, etc.).”*

Dit project is in eerste instantie bedoeld voor een team van twee studenten. Het uitvoerende team bestaat echter uit één persoon, zodat slechts 400 uur voor dit project beschikbaar is. Daarmee is oplevering van alle gewenste producten niet haalbaar. In een vroeg stadium is daarom, in overleg met de opdrachtgever, besloten dat:

1. een Android app (S3App) met een secure server (S3Server) wordt opgeleverd;
2. één policy voor tijdstip van verloop en één policy m.b.t. de vastgestelde locatie wordt ondersteund;
3. geen uitwisselfunctionaliteit voor selfies geimplementeerd wordt;
4. een aanvallermodel wordt opgesteld.

Door deze afspraken, uitgangspunten, het eigen moeten maken van het ontwikkelen onder Android en de hoeveelheid te realiseren (software)producten, heeft de huidige implementatie van het S3-systeem meer het karakter gekregen van het leggen van een goede basis, waaraan andere studenten tijdens vervolgopdrachten verder kunnen werken.

## 2.2 Routekaart

In hoofdstuk 3 wordt ingegaan op de onderzoekscontext waarbinnen dit project is uitgevoerd. Hoofdstuk 4 vormt de domeinanalyse, waarin wordt nagegaan hoe context globaal bepaald en geklassifeerd kan worden. De beschrijving van de opgeleverde softwarecomponenten is opgenomen in hoofdstuk 5.

Hoofdstuk 6 geeft een reflectie op het uitgevoerde project en hoofdstuk 7 belicht de persoonlijke ervaringen, zoals deze tijdens de projectuitvoering zijn opgedaan. Deze twee hoofdstukken zijn beide vanuit een persoonlijk perspectief geschreven.

Tenslotte bevat hoofdstuk 8 de conclusies en de aanbevelingen.

# 3 Onderzoekscontext

Het S3-project is uitgevoerd om ondersteuning te bieden aan onderzoek op het gebied van Context Driven Privacy. Naast de term Context Driven Privacy wordt in de meeste publicaties de term Context-Aware Privacy gebruikt. In het onderzoek binnen de OU richt men zich o.a. op de vraag hoe selfies veilig gedeeld kunnen worden, waarbij de publicerende partij restricties op kan leggen m.b.t. de omstandigheden waaronder betrokken afbeelding getoond mag worden.

In dit hoofdstuk wordt een korte beschijving gegeven van het onderzoek m.b.t. Context-Aware Privacy in het algemeen. Daarna volgt een samenvatting van reeds uitgevoerd onderzoek naar encryptie van te sharen content.

## 3.1 Context Driven Privacy / Context-Aware Privacy

Het onderzoek naar Context-Aware Privacy wordt binnen diverse wetenschapsdisciplines uitgevoerd.
Zo kiezen Reed et al. [2016] voor een sociologische benadering in hun onderzoek naar self-disclosure gedrag van Facebookgebruikers, waarbij men de invloed van de nationaliteit en cultuur van betreffende gebruikers analyseert.
Vanuit de psychologie wordt onderzoek gedaan naar hoe gebruikers mogelijke inbreuken op hun privacy ervaren, hoe zij daarmee omgaan en of zij daar maatregelen voor nemen en/of van daartoe geboden functionaliteit gebruik maken [Joinson et al, 2010] en [Taddei en Contena, 2013].

Binnen de informatica wordt het onderzoek naar Context-Aware Privacy vanuit de techniek benaderd. Vanwege deze technische achtergrond hebben de publicaties vaak een meer formeel wiskundige insteek. Zo moet eerst duidelijk worden, wat precies onder context wordt verstaan (paragraaf 4.2 gaat hier nader op in). Pas als het begrip context duidelijk is, kan men onderzoeken hoe context-awareness gemodelleerd kan worden [Neovius en Sere, 2009]. In zijn dissertatie onderzoekt Ahmed [2010] diverse methoden van Context-Aware access control, waarbij hij een nieuw acces-control model voorstelt.

Op grond van laatstgenoemde dissertatie kan men stellen, dat Context-Aware Privacy neerkomt op het bieden van toegang tot bepaalde (privacygevoelige) content en systemen, afhankelijk van de context waarbinnen de gebruiker zich op dat moment bevindt.

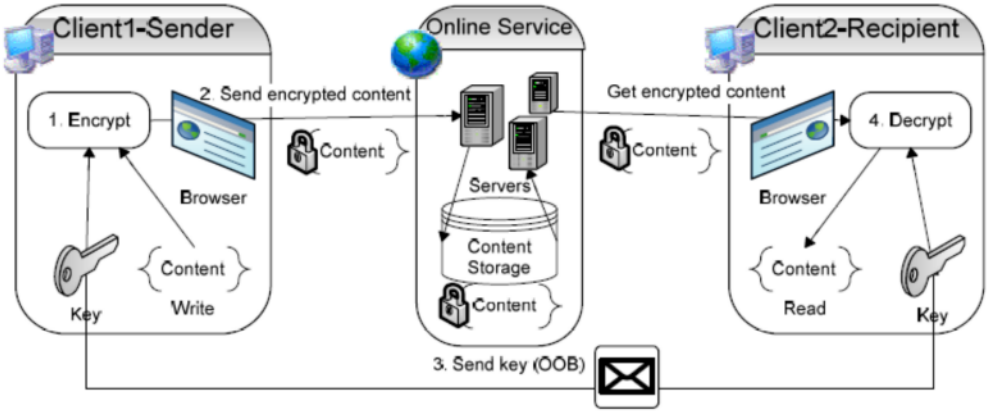
## 3.2 Veilig delen van content

Voor het aspect veilig delen van content worden, als voorbeeld, twee systemen beschreven die in publicaties zijn voorgesteld, te weten: BlogCrypt [Paulik et al., 2010] en P3 [Ra et al., 2013].

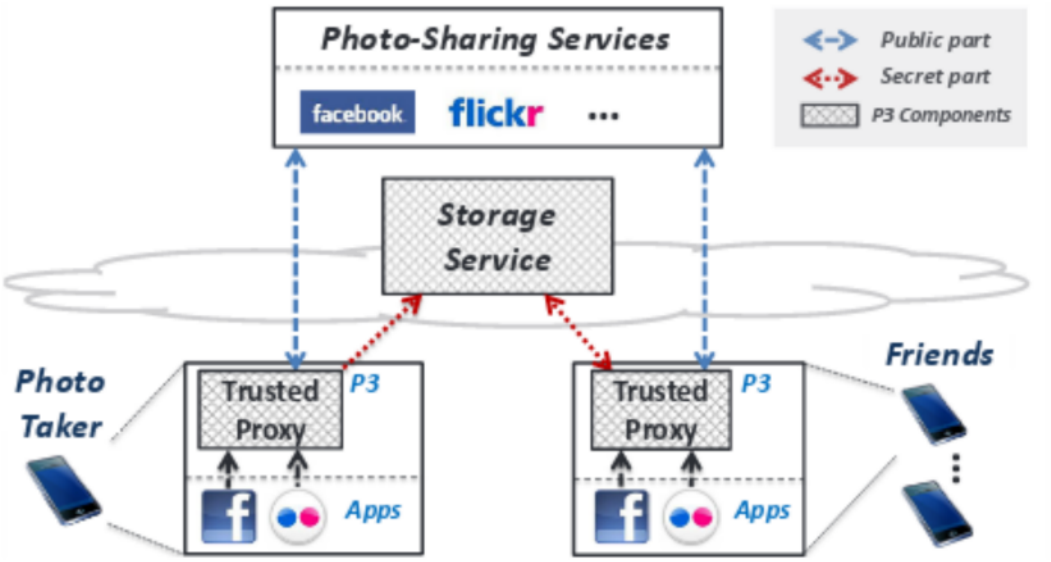
Met BlogCrypt stellen de onderzoekers een methode voor, om m.b.v. symmetrische encryptie blogteksten met een vertrouwelijk karakter te kunnen publiceren op publiek toegankelijke blogging websites. Het idee hierachter is, dat de publicerende partij de tekst voor de publicatie versleutelt. Daarna kunnen alleen diegenen die over de benodigde symmetrische key beschikken, de tekst weer reconstrueren en lezen. Afbeelding 1 geeft dit principe weer. Paulik et al. [2010] geven hierbij wel aan, dat het keymanagement nog een probleem is. In afbeelding 1 is te zien, dat men voorlopig uit is gegaan van keydistributie via email.

P3 [Ra et al., 2013] is een soortgelijk systeem als BlogCrypt, maar het richt zich op het encrypten van (gedeelten) van foto’s. Men maakt daarbij onderscheid tussen publieke en private delen van foto’s. Uit afbeelding 2 blijkt dat men de publieke gedeelten vrij via social media deelt, terwijl de geheime gedeelten -via trusted proxies- op een speciale P3 storage server worden geplaatst en geshared.
Net zoals bij BlogCrypt gaat men bij P3 uit van symmetrische encryptie, waarbij men geen uitspraken m.b.t. de keydistributie doet.

Zowel BlogCrypt als P3 nemen echter niet de gebruikerscontext in beschouwing. De gebruikercontext is echter wel medebepalend voor wanneer een decryptie-actie veilig is.




*Afbeelding 1: Voorgesteld model voor BlogCrypt [Paulik et al., 2010]*




*Afbeelding 2: Architectuur P3-systeem [Ra et al., 2013].*

## 3.3 De meerwaarde van het S3-project

Voor het S3-project is ervoor gekozen om naast de S3App ook een S3Server te realiseren.
De app neemt de foto's en versleutelt deze m.b.v. een zelf gegenereerde symmetrische sleutel. De server doet dienst als opslag voor deze gegenereerde keys. Een opgevraagde key wordt alleen door de server verstrekt als vanuit de app aan eventueel vooraf gedefinieerde contextgebaseerde policies is voldaan.

S3 gaat op twee punten verder dan BlogCrypt en P3: S3 neemt de gebruikerscontext in acht en biedt een basale vorm van keymanagement.

Op moment van schrijven van deze scriptie werkt de opdrachtgever aan het wetenschappelijke artikel, waar het S3-project een onderdeel van vormt.

# 4 Domeinanalyse - Context globaal bepalen en klassificeren

## 4.1 Introductie

Deze analyse bevat de resultaten van een klein literatuuronderzoek. Eerst wordt in paragraaf 4.2 de context van deze analyse beschreven en definities van Context en Context-Awareness gegeven. Paragraaf 4.3 gaat in op het meten en bepalen van contextwaarden. Paragraaf 4.4 beschrijft het modelleren van context en paragraaf 4.5 vervolgt met een beschrijving van services waarmee men contextgegevens kan verrijken. Men spreekt daarom ook wel van context augmentation services. Tenslotte volgt paragraaf 4.6 met de conclusies.

### 4.1.1 Verantwoording

Deze domeinanalyse is het eindproduct van de tweede iteratiefase binnen het project “Securely Sharing Selfies”. Het gaat in op het onderwerp “Context bepalen & klassificeren”, met als uitbreidingsonderwerp “Inhaken op location/context augmentation services”.

### 4.1.2 Vraagstelling

De hierboven genoemde onderwerpen worden als onderstaande vragen verwoord:

**Vraag 1:**
*“Wat kan men, m.b.v. algemeen beschikbare smartphones, aan omgevingswaarden bepalen en hoe kan men daaruit meer abstracte contexten vaststellen?”*

**Vraag 2:**
*“Hoe kan men contextgegevens verrijken m.b.v. informatie/functionaliteit van derde partijen?”*

## 4.2 De begrippen context en context-awareness

Sinds 2005 zijn de smartphones in opmars. De geboden mogelijkheden en verwerkingscapaciteit nemen ieder jaar toe. Moderne smartphones hebben meerdere sensoren, waarmee metingen m.b.t. de directe omgeving mogelijk zijn. Vervolgens kan de programmatuur op de smartphone hierop reageren. Het principe dat altijd en overal computerfunctionaliteit voor gebruikers beschikbaar is, duidt men aan met de term ubiquitous computing. Hierbij spelen de begrippen context en context-awareness een grote rol.

Eerst moet dus vastgesteld worden wat onder het begrip context wordt verstaan. Uit de publicaties van Riahi en Moussa [2015] en van Hoyos et al. [2013] blijkt, dat de door Dey [2001] voorgestelde definitie van context onder onderzoekers het meest gangbaar is. Deze definitie luidt:
*“Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves”*

Hoewel deze definitie algemeen wordt geaccepteerd, is deze nog heel breed. Er bestaat nog steeds geen consensus over wat het begrip context nu eigenlijk precies inhoudt [Hoyos et al., 2013]. Op dit moment bestaan twee hoofdstromingen. Één groep voegt de notie van social context toe en houdt rekening met de emoties en de gemoedstoestand van de gebruiker. Andere auteurs baseren hun werk op Dey's definitie en beschouwen de context als de verzameling van alle externe applicatieparameters die het applicatiegedrag beïnvloeden en definiëren daarbij nieuwe gezichtspunten op de applicatiedata en -services [Riahi en Moussa, 2015].

In deze domeinanalyse wordt uitgegaan van de laatste denkwijze, ofwel:
*“De context is de verzameling van alle externe applicatieparameters die het applicatiegedrag beïnvloeden.”*

In het spraakgebruik wordt flexibel met deze definitie omgegaan. Volgens deze definitie zijn zowel de tijd als de locatie parameters binnen de context. Maar vaak zal -ook in deze analyse- over de (sub)contexten tijd en locatie worden gesproken.

Binnen de context kan men entiteiten onderscheiden. Zo zijn artsen en patiënten personen in de context van een ziekenhuis. Maar binnen diezelfde context zijn artsen ook werknemers. Hoe en welke entiteiten binnen de context worden onderscheiden, hangt af van het doel van de applicatie. Men kan deze entiteiten als subcontexten beschouwen. Deze domeinanalyse maakt regelmatig gebruik van deze mogelijkheid.

Dey geeft tevens een definitie van context-awareness [Dey, 2001]:
*“A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user’s task.”*

Deze domeinanalyse sluit zich bij deze definitie aan.

## 4.3 Meten van context

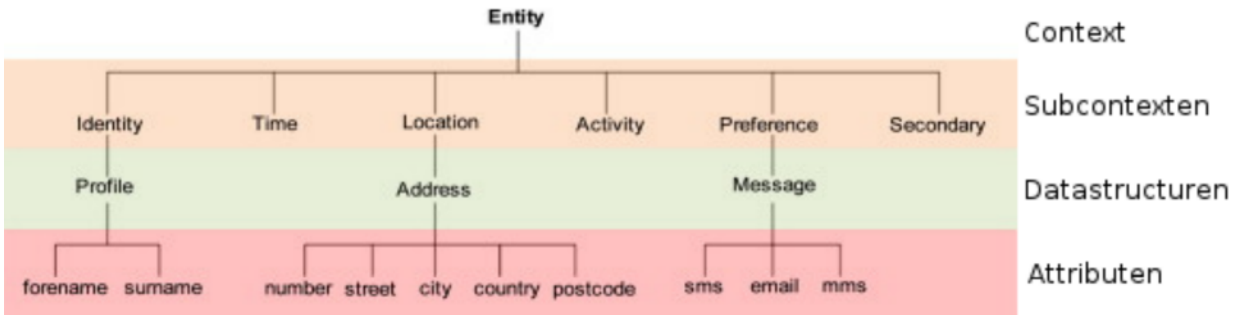
### 4.3.1 Beperkingen aan contextbepaling

In de praktijk zal het doorgaans niet mogelijk zijn, een context op basis van gemeten waarden van slechts één sensor te bepalen.

*Voorbeeld “Weertype”:*
Stel dan men geinteresseerd is in het weertype van de locatie waar een smartphonegebruiker zich bevindt. De context “weertype” kent de domeinwaarden: arctisch, koud, gematigd, subtropisch en tropisch.
Stel verder, dat men hierbij alleen de beschikking heeft over de waarden die de smartphone zelf heeft kunnen meten.
In dit geval is het niet voldoende om alleen de omgevingstemperatuur te meten. Ook gegevens als luchtvochtigheid en luchtdruk moeten beschouwd worden, om tot een betrouwbaarder waarde voor de context “weertype” te komen. Zo kan men besluiten om weertype de waarde “tropisch” toe te kennen, indien de temperatuur ≥ 30ºC en zowel de luchtvochtigheid als de luchtdruk hoge waarden hebben. Voor de waarde “arctisch” kan men wellicht volstaan met de constatering dat de temperatuur ≤ -20ºC, waarbij de waarden voor luchtvochtigheid en luchtdruk niet van belang zijn.

Als de genoemde omgevingswaarden niet via de sensoren van de telefoon kunnen worden gemeten, dan kunnen deze gegevens (m.b.v. locatieservices) alsnog via externe partijen worden verkregen.
Zo laten Android smartphones de gebruikers zien wat hun omgevingstemperatuur is en of het regent, bewolkt of zonnig is. Dit alles o.b.v. de vastgestelde locatie.

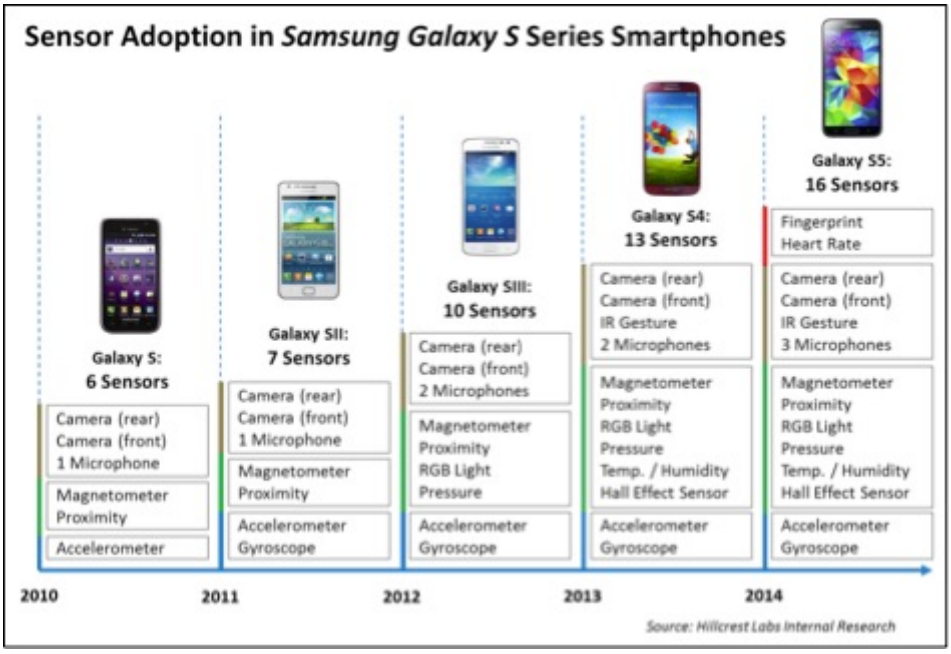
Het is gebruikelijk dat door sensors gemeten waarden niet zondermeer op contexten, zoals die binnen een systeem bruikbaar zijn, geprojecteerd kunnen worden [Hoyos et al., 2013].
Afbeelding 3 geeft een voorbeeld van gegevens en de uiteindelijke contexten waarop deze gegevens betrokken worden [Achilleos et al., 20odijk10]. Men kan afbeelding 3 als volgt lezen: De bovenste laag (Entity) staat voor de gehele context. Laag 2 geeft de subcontexten weer. In laag 3 staan de datastructuren, waarin de attributen van de onderste laag opgenomen zijn.




*Afbeelding 3: Voorbeeld verband tussen gegevens en uiteindelijke context, naar [Achilleos et al., 2010]*

### 4.3.2 Smartphones en sensoren

De laatste jaren neemt het aantal sensoren in smartphones toe. Als voorbeeld geeft onderstaande figuur deze groei weer voor Samsung Galaxy S modellen.




*Afbeelding 4: Voorbeeld van toename sensoren in smartphones*
*http://www.rcrwireless.com/20150302/opinion/reader-forum-sensor-adoption-in-smartphones-tag10*
*geraadpleegd 06-02-2016.*

Twee sensoren behoeven toelichting. De Hall Effect sensor is toegevoegd om waar te nemen of de cover open of gesloten is. De RGB Light sensor neemt de intensiteit van het licht waar en past de helderheid van het scherm hierop aan [webref: Samsung newsroom, 2014].
Naast de in afbeelding 4 genoemde sensoren, bevatten alle recente smartphones een GPS chip, een klok, Bluetooth en Wi-Fi functionaliteit. Vaak zijn ze ook voorzien van een FM radio-ontvanger, waardoor digitale informatie via RDS [Li et al., 2011] kan worden ontvangen. Smartphones kunnen dus diverse omgevings-waarden meten. Dat betekent echter niet dat apps direct toegang tot alle sensoren en hun meetwaarden hebben, zie onderstaand voorbeeld.

*Voorbeeld “Android Location”:*
Zo biedt Android het object Location [webref: Android Developers, 2016-1]. Location geeft de geografische locatie van de smartphone weer en biedt verder extra functionaliteit zoals richting, snelheid en afstandsberekening tot een andere locatie. De ontwikkelaar kan gebruik maken van dit object zonder zich druk te hoeven maken hoe deze geografische locatie precies wordt verkregen. Als bron voor dit Location object kunnen GPS, het GSM-netwerk (Cell-ID) en Wi-Fi locatiegegevens dienen. De gebruiker en de ontwikkelaar kunnen overigens wel invloed uitoefenen op hoe en hoe nauwkeurig deze locatiedata worden verkregen [webref: Android Developers, 2016-2].

Android biedt ook de algemene objecten Sensor [webref: Android Developers, 2016-3] en SensorEvent [webref: Android Developers, 2016-4]. Hiermee kan naar sensoren worden “geluisterd”. Het correct gebruiken van deze objecten is een low level activiteit [webref: Android Developers, 2016-4]. Dit werkt natuurlijk alleen voor die sensoren, die in de smartphone aanwezig zijn. De sensoren die via Sensor en SensorEvent door Android worden ondersteund, blijken niet voor locatiebepaling bruikbaar. Dit zijn sensoren als: accelerometer; ambient temperature sensor; rotation vector sensor; gravity sensor; gyroscope sensor; heart rate monitor; light sensor; magnetic field sensor; pressure sensor; proximity sensor; relative humidity sensor; step counter sensor en een step detector sensor [webref: Android Developers, 2016-3].
Deze ondersteuning is beschikbaar sinds Android 4.4W (juni 2014) [webref: Developer.xamarin.com, 2016].

### 4.3.3 Problemen bij vaststellen van een context

Uit het voorbeeld “Weertype” blijkt, dat meerdere soorten sensoren nodig kunnen zijn om tot een betere en betrouwbaardere contextwaarde te komen. Het voorbeeld “Android Location” beschrijft dat ook voor de locatiebepaling meerdere sensoren ingezet kunnen worden. In dat geval worden verschillende soorten sensoren benut om elkaar aan te vullen, of om als backup te dienen. Veel gebruikers zetten hun GPS uit, omdat GPS-gebruik veel van de accucapaciteit vergt. Vervolgens kan via Cell-ID toch nog een redelijk goede locatiebepaling plaatsvinden.

Het inzetten van meerdere sensoren kan echter ook tot problemen leiden. De gemeten waarden van verschillende sensoren kunnen inconsistent zijn. Zo kan een locatie die m.b.v. GPS is bepaald, beter zijn dan de locatie die 9 seconden later m.b.v. het Cell-ID wordt vastgesteld [webref: Android Developers, 2016-2]. Dit is het domein van “context inconsistency detection and resolution”. Zhang et al. geven in een overzichtspublicatie mogelijke oplossingsrichtingen en bijbehorende publicaties [Zhang et al., 2011].

### 4.3.4 Context reasoning

Naast het direct projecteren van meetwaarden van meerdere sensoren op de gewenste context, is het ook mogelijk om abstractere en impliciete contexten uit reeds bestaande contexten af te leiden. Dit is het gebied van context reasoning [Bettini et al., 2010]. Context reasoning kan al snel complexe vormen aannemen.

*Voorbeeld “Uitgaansgebied”:*
Stel dat we van een smartphonegebruiker willen vaststellen of hij/zij werkt, vrij is of uitgaat. Stel verder dat de smartphone alleen de contexten Tijd en Locatie vast kan stellen.
Als de gebruiker zich op zaterdagavond om 23:30 in een uitgaansgebied bevindt, hoeft dat nog niet te betekenen dat hij/zij aan het stappen is. De gebruiker zou ook een barkeeper, uitsmijter of dienstdoend agent kunnen zijn. In die gevallen is de gebruiker aan het werk.

Kortom, hier zijn de contexten Tijd en Locatie niet voldoende om de context Activiteit correct te kunnen bepalen. Er zijn extra contexten/gegevens zoals o.a. beroep, werkrooster, vakantieperiodes en een bijbehorend reasoningproces nodig om tot een goede uitspraak te komen. Men kan context reasoning op verschillende manieren benaderen. Hieronder worden drie voorbeelden uit recente publicaties beschreven.

**Data-driven context reasoning**
Data-driven technieken zijn gebaseerd op machine learning [Lu et al., 2012]. Guinness [2015] beschrijft een machine learning benadering voor context reasoning. Daarbij wordt gebruik gemaakt van GPS en accelerometers op smartphones, geospational informatie (points of interest, zoals bushaltes en treinstations). Vervolgens wordt machine learning toegepast om de mobiele activiteit van de gebruiker vast te stellen. Daarbij wordt onderscheid gemaakt tussen: lopen, rennen, auto rijden, gebruik maken van een bus of een trein. In de publicatie worden diverse technieken voor machine learning qua prestaties en accuraatheid met elkaar vergeleken.

**Knowledge-driven context reasoning**
Lu et al. [2012] stellen een benadering voor waarbij gebruik wordt gemaakt van predikaatlogica. Daarbij merken zij op, dat een activiteit niet alleen vastgesteld kan worden o.b.v. van (context)waarnemingen. Voor hun model is het noodzakelijk dat kennis van de relatie tussen de context en de activiteit expliciet binnen het model voor het reasoningproces wordt opgenomen. Zij kiezen daarmee voor een knowledge-driven benadering van context reasoning.

Predikaatlogica kent geen syntax waaruit de structuur van de omhullende context expliciet blijkt [Sowa, 2000]. Om dit te ondervangen, kan men zowel temporele als semantische relaties toevoegen aan een op predikaatlogica gebaseerd model [Lu et al., 2012]. Deze toegevoegde relaties zijn op hun beurt weer uitdrukkingen in predikaatlogica. De volgende beschrijving geeft de opbouw van zo’n model enigszins vereenvoudigd weer [Lu et al., 2012].

Stel dat een systeem een set contexten kent volgens $c_1, c_2,...c_n$ en het domein van $c_i$ is $D_i$.
Op een gegeven tijdstip $t$ wordt de waarde van context $c_i$ weergegeven als $c_i(t)$, waarbij $c_i(t) \in D_i$.
De waarden van context $c_1, c_2,...c_n$ op tijdstip $t$ wordt genoteerd als $C(t)$, waarbij $C(t) = (c_1(t), c_2(t),...c_n(t))$.
Stel vervolgens dat $d_j^{(i)} \subseteq D_i$, dan is $(c_i, d_j^{(i)})$ een context pattern. Dit context pattern $(c_i, d_j^{(i)})$ geldt op een tijdstip $t$, dan en slechts dan als $c_i(t) \in d_j^{(i)}$; weer te geven als $hold\_at((c_i, d_j^{(i)}), t)$.

Een context pattern kan gelden in één of meer niet overlappende tijdsintervallen. Hierbij wordt een interval voorgesteld door het paar $[t_s, t_e]$, hierin is $t_s$ de starttijd en $t_e$ de eindtijd van het betreffende interval. Nu kan men de intervalset $I(p)$ definiëren, waarin het context pattern $p$ geldig is volgens:

$$I(p) = \{ [t_s, t_e] \mid \forall t \in [t_s, t_e], hold\_at(p, t) \text{ and } \neg \exists [t_m, t_n], t_m \leq t_s, t_n \geq t_e, [t_m, t_n] \neq [t_s, t_e], \forall t \in [t_m, t_n], hold\_at(p, t) \}$$

Uitgaande van een context pattern $p$ en een activiteit $a$, kan men de in tabel 1 weergegeven temporele relaties definiëren. Hierin vormt $a.start$ het begin van de activiteit, terwijl $a.end$ het einde van de activiteit weergeeft.

| Relation | Definition |
|---|---|
| $Equal(p,a)$ | $\exists [t_{ps},t_{pe}] \in I(p), t_{ps} = a.start, t_{pe} = a.end$ |
| $During(p,a)$ | $\exists [t_{ps},t_{pe}] \in I(p), t_{ps} \geq a.start, t_{pe} \leq a.end$ |
| $During(a,p)$ | $\exists [t_{ps},t_{pe}] \in I(p), t_{ps} \leq a.start, t_{pe} \geq a.end$ |
| $Start(p,a)$ | $\exists [t_{ps},t_{pe}] \in I(p), t_{ps} = a.start, t_{pe} < a.end$ |
| $Start(a,p)$ | $\exists [t_{ps},t_{pe}] \in I(p), t_{ps} = a.start, t_{pe} > a.end$ |
| $Finish(p,a)$ | $\exists [t_{ps},t_{pe}] \in I(p), t_{ps} > a.start, t_{pe} = a.end$ |
| $Finish(a,p)$ | $\exists [t_{ps},t_{pe}] \in I(p), t_{ps} < a.start, t_{pe} = a.end$ |
| $Overlap(p,a)$ | $\exists [t_{ps},t_{pe}] \in I(p), t_{ps} < a.start, t_{pe} < a.end$ |
| $Overlap(a,p)$ | $\exists [t_{ps},t_{pe}] \in I(p), t_{ps} > a.start, t_{pe} > a.end$ |
| $Before\ (p,a)$ | $\exists [t_{ps},t_{pe}] \in I(p), t_{ps} < a.start$ |

*Tabel 1: Temporele relaties tussen context patterns en activiteiten [Lu et al., 2012]*

Ook kan men semantische relaties definiëren, zie tabel 2. In tabel 2 worden ook de overeenkomende temporele relaties benoemd.

| Relation | Meaning | Corresponding temporal realtion |
|---|---|---|
| COND($p$,$a$) | Context pattern $p$ is a condition of conducting activity $a$. If it is not satisfied as its corresponding temporal pattern, the activity is abnormally conducted. | $Before(p,a)$, $During(a,p)$, $Overlap(p,a)$ |
| PREM($p$,$a$) | Context pattern $p$ is a premise of conducting activity $a$. If it is not satisfied as its corresponding temporal pattern, the activity is not started or interrupted. | $Overlap(p,a)$, $During(a,p)$, $Finish(a,p)$ |
| ACCOMP($p$,$a$) | Context pattern $p$ will occur as its corresponding temporal pattern if the activity $a$ is conducted normally | $Start(p,a)$, $Equal(p,a)$, $During(p,a)$, $Finish(p,a)$, $Overlap(a,p)$ |
| CONSEQ($p$,$a$) | Context pattern $p$ is a consequence of activity $a$. | $Overlap(a,p)$ |

*Tabel 2: Semantische relaties tussen context patterns en activiteiten [Lu et al., 2012]*

O.b.v. het voorgaande kan men diverse patterns definiëren. Hier wordt volstaan met de definitie van het **start pattern** van een activiteit *a*. Dit pattern geldt als een activiteit wordt gestart.
Als $a \in A$, dan is *s-pattern(a)* een **start pattern** van activiteit *a*, dan en slechts dan als:

$$s\text{-}pattern(a) = \{ p \mid PREM(p,a) \vee (ACCOMP(p,a) \wedge (start(p,a) \vee equal(p,a) \vee start(a,p))) \}$$

Als men de activiteit modelleert met een state-transition diagram, dan kan men het zojuist geconstrueerde apparaat gebruiken om zowel de statussen als de transitieregels tussen deze statussen eenduidig uit te drukken.

**Knowledge-driven benadering of data-driven benadering?**
Knowledge-driven benaderingen en data-driven benaderingen sluiten elkaar niet uit.
Zo passen Stein en Gonzalez [2014] context reasoning juist toe om een slecht presterende machine learning gebaseerde oplossing te verbeteren. De case in deze publicatie wordt gevormd door een gesimuleerde havenkraan die boxen op moet kunnen pakken en weer neer moet kunnen zetten. Een pure machine learning benadering blijkt hier niet goed te werken. Nadat de uit te voeren taak in contexten wordt opgesplitst en context reasoning wordt toegepast, neemt de prestatie van de gehele simulatie significant toe.

Lu et al. [2012] merken op, dat data-driven benaderingen voornamelijk ingezet kunnen worden om uit ruwe sensordata eenvoudige activiteiten vast te stellen. De genoemde problemen in de publicatie van Stein en Gonzalez [2014] bevestigen deze waarneming.

Het voorgaande leidt ertoe dat, welke benadering men voor een te realiseren systeem ook kiest, men altijd kennis aan het te implementeren context reasoning proces toe zal moeten voegen. Alleen in relatief eenvoudige situaties kan men volstaan met een puur data-driven benadering.

**Afsluitend**
Het zal duidelijk zijn dat een te realiseren context reasoning proces toepassingsafhankelijk is.
Stel dat iemand door een groot vliegveld loopt en ineens behoefte aan ontspanning voelt. Voor de één is dat het spelen van een (gok)spel in een automatenhal, terwijl een ander misschien liever op een terras zit. Beide vormen van ontspanning kunnen ook op dezelfde persoon van toepassing zijn, afhankelijk van zijn of haar gemoedstoestand. Als men een mobiele app gaat bouwen, die gebruikers passende voorstellen kan doen, zal men eerst vast moeten stellen hoe men de gemoedstoestand van een persoon kan bepalen. In dit traject zal men ook een passend context reasoning proces moeten realiseren. Aangezien er vele mogelijkheden zijn, zal dit proces zeker niet triviaal zijn.
Voorgaand scenario is een totaal andere situatie dan het geheel geautomatiseerd laden, lossen en verplaatsen van zeecontainers op een overslagterminal op de Maasvlakte. Daar is immers sprake van een context die zich veel strakker laat definiëren (een werklocatie met een vaste fysieke indeling en een goede registratie van de locatie van iedere container die zich binnen deze werklocatie bevindt).

Het context reasoning proces is dus afhankelijk van het doel van het te realiseren systeem, de binnenkomende data en de kennis die nodig is om deze data om te zetten tot betekenisvolle gegevens en contexten binnen dit systeem.

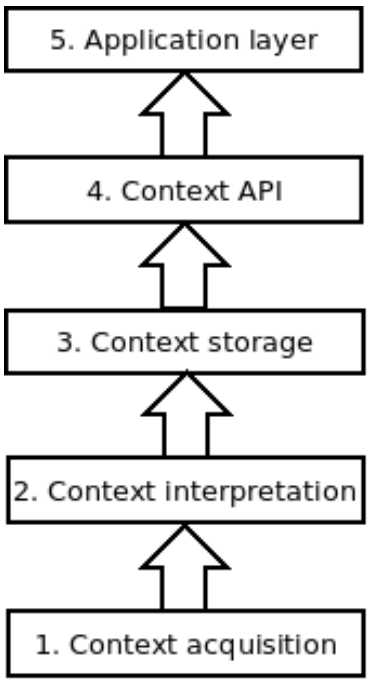
### 4.3.5 Contexten binnen het S3-project

In navolging van Göker en Myrhaug [2002] stellen Hoyos et al. [2013] dat locatie- en tijdsaspecten altijd deel uitmaken van een context. Daarbij beschikken alle moderne smartphones over de hardware om locatie en tijd vast te stellen (zie paragraaf 4.3.2).
Om het project beheersbaar te houden, wordt de focus daarom beperkt tot de subcontexten *Locatie* en *Tijd*. In opvolgende iteraties kunnen meer subcontexten worden toegevoegd.
Omdat door de beperkt beschikbare tijd slechts twee subcontexten in beschouwing worden genomen, wordt binnen het project ook geen bijzondere aandacht aan context reasoning besteed. Er wordt volstaan met een eenvoudige combinatie van de subcontexten *Locatie* en *Tijd*.

## 4.4 Bepalen en modelleren van context

### 4.4.1 Algemene architectuur voor context-aware systemen

Om op een gestructureerde manier over context-aware systemen te kunnen spreken, is het wenselijk om tot een algemene context-aware architectuur te komen. Sinds 1999 zijn diverse architecturen voorgesteld. Riahi en Moussa geven een uitgebreid overzicht van de belangrijkste voorstellen. Tevens stellen zij, dat deze architecturen terug zijn te voeren tot een vijf-lagen-model [Riahi en Moussa, 2015]. Dit vijf-lagen-model wordt -enigzins gewijzigd- weergegeven in afbeelding 5.




*Afbeelding 5: Algemene architectuur van context-aware systemen; vrij naar Riahi en Moussa.*

Riahi en Moussa beschrijven deze lagen als volgt:

1. Context acquisition
   In deze laag worden de meetwaarden van de sensoren verzameld.
2. Context interpretation
   Biedt de mogelijkheid voor analyse en verwerking van de ruwe meetwaarden. Tevens kunnen hier maatregelen worden getroffen m.b.t. elkaar tegensprekende meetwaarden. Context reasoning kan ook binnen deze laag geplaatst worden.
3. Context storage
   Hier worden de contextgegevens opgeslagen. Daarvoor is het noodzakelijk dat deze contextgegevens gemodelleerd zijn.
4. Context API
   Deze laag vormt de adapter tussen de context storage en de application layer. Het biedt services voor de applicatielaag. Ook kan het berichten naar de applicatielaag sturen, indien wijzigingen in de context daartoe aanleiding geven.
5. Application layer
   Hier bevindt zich de logica die op applicatieniveau nodig is om goed op (wijzigende) context(s) te reageren.

### 4.4.2 Implementaties voor contextmodellen

Contexten en context reasoning kan men weliswaar modelleren m.b.v. predikaatlogica [Sowa, 2000] [Lu et al., 2012], maar daarmee heeft men nog geen executeerbare implementatie in handen. Daarvoor zijn vertalingen naar programmeertalen nodig.

Als het over de gebruikerscontext gaat, stellen Göker en Myrhaug dat deze vijf subcontexten kan bevatten. Te weten: environment context; personal context; task context; social context en spatio-temporal context [Göker en Myrhaug, 2002]. Deze indeling kan gebruikt worden als een eerste stap om van meetwaarden naar bruikbare contexten te komen. Zo heeft de omgevingstemperatuur betrekking op de environment context en zegt de gemeten hartslag iets over de personal context.

Om meetwaarden naar bruikbare contexten te transformeren, moeten deze contexten gemodelleerd worden. Op het gebied van context modelling zijn sinds 2000 diverse publicaties verschenen. Zhang et al. geven hiervan een uitgebreid overzicht [Zhang et al., 2011]. Daarbij onderkennen zij acht verschillende technieken, te weten: key-value modelling; markup modelling; graphical modelling; object-oriented modelling; logic-based modelling; ontology-based modelling; multidisciplinary modelling en domain-focused modelling.
Zhang et al. stellen ook vast dat de vraag, hoe te evalueren dat een contextmodel goed is, nog steeds open ligt. Volgens Hoyos et al. is het noodzakelijk om verschillende typen contextinformatie te onderscheiden, zodat men een taxonomie toe kan passen om de concepten correct te modelleren [Hoyos et al., 2013].

In 2005 introduceerde Bardram het Java Context Awareness Framework (JCAF) [Bardram, 2005]. Dit framework werkt o.b.v. object-oriented modelling, maar kent enkele problemen [webref: Szántó, 2010]:

1. JCAF is gerealiseerd o.b.v. Remote Method Invocation (RMI). RMI is een Java-platformspecifieke technologie. Het gebruik van RMI is daardoor beperkt tot (W)LAN's. RMI is daarom niet bruikbaar binnen ubiquitous computing.
2. JCAF draait alleen onder een standaard Java installatie. Daardoor is JCAF alleen bruikbaar op computers en kan er alleen sprake zijn van communicatie tussen computers. Ook dit leidt tot een inzet op alleen (W)LAN's. Communicatie met hedendaagse mobile devices is daardoor onmogelijk.

In 2010 is dit framework voorzien van een RESTful API [webref: Szántó, 2010]. Dit houdt in dat JCAF nu via HTTP requests als GET, POST, PUT en DELETE aan kan worden gesproken om operaties op en met contexten uit te voeren.
De afgelopen jaren heeft JCAF een rol gespeeld binnen diverse contextgerelateerde publicaties, zoals de JCOOLS toolkit [Park et al., 2013] en de DSL MLContext [Hoyos et al., 2013].

**JCOOLS**
Een op JCOOLS gebaseerd systeem bestaat uit een context-aware server en één of meerdere end-user applicaties. De verwerking op de server wordt uitgevoerd m.b.v. JCAF en de Drools business rules engine. T.b.v. JCAF worden de contexten object-georiënteerd gemodelleerd. Drools wordt ingezet voor de business logica en reasoning.
De JCOOLS server werkt o.b.v. XML-definities voor objecten, relationships en actions. Hiertoe wordt met de contexteditor het context-informatieschema samengesteld. Uit dit schema worden vervolgens de benodigde XML-definities gegenereerd. Communicatie met de door de server geboden services verloopt via het XML-RPC protocol [Park et al., 2013].

**MLContext voor contextmodelling**
Met de ML Context DSL worden contexten op abstracte wijze logisch gemodelleerd. Het modelleringswerk vindt plaats m.b.v. de Eclipse ontwikkelomgeving. Vervolgens kunnen artifacten voor o.a. JCAF gegenereerd worden [Hoyos et al., 2013].

### 4.4.3 Contextmodelling binnen het S3-project

JCOOLS blijkt niet via internet beschikbaar. MLContext kan wel via internet worden verkregen [webref: MLContext, 2015]. JCAF is -met de RESTful uitbreiding- beschikbaar via internet [webref: Sourceforge.net, 2016]. Hoewel JCAF interessante mogelijkheden biedt, wordt dit framework niet binnen het project gebruikt. De redenen hiervoor zijn dat:

1. JCAF summier gedocumenteerd is, het leren van dit framework kost daardoor extra tijd;
2. het onderhoud aan JCAF sinds 2010 stil ligt;
3. het gebruik van JCOOLS of MLContext extra inleertijd kost. Dit gaat ten koste van de beschikbare tijd voor de implementatie;
4. JCAF –ondanks de RESTful uitbreiding– nog steeds niet op een Java webcontainer zoals Tomcat draait. De schaalbaarheid van JCAF is daardoor beperkt.

Gezien het beperkte aantal contexttypen binnen dit project (zie paragraaf 4.3.5) en de beperkte hoeveelheid beschikbare tijd voor de realisatie, is gekozen voor object-oriented contextmodelling. Dit sluit tevens aan bij het curriculum van de Open Universiteit, waar sinds 1997 de object-georiënteerde programmeertaal Java wordt gedoceerd.

## 4.5 Location augmentation services

In paragraaf 4.3.2 is vastgesteld dat alle recente smartphones voorzien zijn van een GPS chip en een klok. In paragraaf 4.3.5 is aangegeven dat de locatie altijd een rol speelt binnen context-aware systemen. Daarom worden binnen deze analyse alleen augmentation services voor locaties in beschouwing genomen. Location augmentation services bieden mogelijkheden tot verrijking van locatiegegevens. In het vervolg van dit hoofdstuk wordt aan de hand van enkele voorbeelden bekeken wat de mogelijkheden zijn om deze services vanuit een context-aware systeem te gebruiken.

Er zijn twee manieren om via het HTTP-protocol services aan te bieden, via webservices en via API's. Het verschil tussen beide mogelijkheden is subtiel. In het huidige spraakgebruik (per 2016) wordt de term webservices vaak gebruikt voor klassieke webservices die gebruik maken van het SOAP berichtformaat. Bij API's wordt doorgaans gedacht aan berichtuitwisseling in JSON formaat. Dit onderscheid is echter niet strikt. In beide gevallen is er sprake van loosely coupled functionaliteit, die dikwijls door externe partijen wordt aangeboden.

### 4.5.1 Overzicht services

Onderstaande tabel geeft een voorbeeldoverzicht van augmentation service providers en de mogelijke formaten waarin zij responses leveren.

| Augmentation service providers | JSON | SOAP | Opmerkingen |
|---|---|---|---|
| Google Maps Web Service API's http://developers.google.com/maps/web-services | X | - | XML responses ook mogelijk, maar dit wordt niet door Google aangeraden. |
| Kadaster Geocoderen http://www.kadaster.nl/web/artikel/producten/Geocoderen.htm | - | X | - |
| Publieke Dienstverlening op de kaart https://www.pdok.nl/nl/producten/pdok-services/overzicht-urls | X | - | Tevens diverse XML formaten die binnen de geo-wereld gangbaar zijn, afhankelijk van de aangeroepen service. |
| Postcode API http://www.postcodeapi.nu/ | X | - | - |
| Webservices.nl http://webservices.nl | - | X | Ook XML-RPC en HTML-RPC mogelijk. |

*Tabel 3: Voorbeelden augmentation services providers en response formats, per 21-02-2016.*

**Voorbeeld JSON gebaseerde API**
Een sprekend voorbeeld voor API's zijn de “Google Maps Web Service API's” [webref: Google Developers, 2016-1]. Dit is een verzameling API's die bestaat uit: Directions API; Distance Matrix API; Elevation API; Geocoding API; Geolocation API; Roads API; Time Zone API en de Places API Web Service. Om het werk voor ontwikkelaars ter vereenvoudigen, stelt Google voor de eerste zeven API's zowel een Java client library [webref: GitHub, 2016-1] als een Python client library [webref: GitHub, 2016-2] beschikbaar.

Kijkt men bijvoorbeeld naar de Distance Matrix API [webref: Google Developers, 2016-2], dan valt het volgende op:

1. De request bestaat uit een eenvoudig HTTP GET-request, voorzien van een query string.
2. De response bevat een bericht in JSON formaat.

Deze twee kenmerken maken dat het gebruik van een API m.b.v. een moderne programmeertaal relatief eenvoudig is. Voor Java- en Pythonontwikkelaars wordt het nog eenvoudiger, vanwege de beschikbare client libraries.

**Voorbeeld SOAP gebaseerde webservice**
Webservices.nl is een voorbeeld van een provider die een uitgebreid assortiment aan SOAP based webservices aanbiedt, waaronder geolocation services [webref: Webservices.nl, 2016-1].
Zo kan men met de webservice *geoLocationPostcodeCoordinatesLatLon*, m.b.v. op te geven lengte- en breedtecoordinaten, de nederlandse postcode van een gegeven locatie ophalen. Webservices.nl biedt, naast de standaard SOAP interface, ook interfaces voor de XML-RPC en HTTP-RPC prococollen [webref: Webservices.nl, 2016-2].

T.b.v. SOAP is de WS-Security specificatie beschikbaar. WS-Security beschijft diverse security gerelateerde uitbreidingen op de SOAP-standaard. Deze uitbreidingen hebben betrekking op het versleutelen van (delen van) de berichtinhoud. WS-Security wordt verder niet in beschouwing genomen, want:

- de genoemde augmentation services bieden geen ondersteuning voor WS-Security;
- toepassing van HTTPS is al voldoende om versleuteld HTTP verkeer te realiseren.

### 4.5.2 Augmentation services en het S3-project
Het voornaamste voordeel van deze services is, dat ze op eenvoudige wijze functionaliteit aan het context-aware systeem toe kunnen voegen. Daarmee wordt veel werk uit handen van de ontwikkelaars genomen. Het grootste nadeel is dat de latency toe zal nemen. De beschreven services worden altijd benaderd via het internet. Dat betekent dat er extra netwerkverkeer plaats moet vinden om van de geboden functionaliteit gebruik te maken. Extra netwerkverkeer leidt automatisch tot extra latency. Daardoor kunnen de responsetijden van het context-aware systeem toenemen.

Verder is het gebruik van services is niet (geheel) vrij. De door Google geleverde services zijn tot op zekere hoogte vrij in het gebruik. Er moet altijd een key worden opgevraagd. Als het gebruik van de services over een drempelwaarde komt, worden kosten berekend [webref: Google Developers, 2016-3]. Voor de webservices bij Webservice.nl moet altijd worden betaald.

Het project heeft geen budget. Daarnaast is de beschikbare tijd voor realisatie beperkt. De realisatie wordt daarom niet complexer uitgevoerd dan nodig is. Benader daarom:

1. vrij te gebruiken API's/webservices;
2. bij voorkeur de JSON varianten, deze zijn eenvoudiger in het gebruik. Gebruik alleen een SOAP gebaseerde service, daar waar geen JSON gebaseerde tegenhanger voorhanden is.

## 4.6 Conclusies

### 4.6.1 Beantwoording vraag 1

Vraag 1 uit paragraaf 4.1.2 luidt:
*“Wat kan men, m.b.v. algemeen beschikbare smartphones, aan omgevingswaarden bepalen en hoe kan men daaruit meer abstracte contexten vaststellen?”*

Volgens paragraaf 4.3 zijn alle smartphones voorzien van een GPS chip, Bluetooth en Wi-Fi. Naast het vaststellen van de locatie via het Location object, biedt Android vanaf versie 4.4W ondersteuning voor de volgende sensoren: accelerometer; ambient temperature sensor; rotation vector sensor; gravity sensor; gyroscope sensor; heart rate monitor; light sensor; magnetic field sensor; pressure sensor; proximity sensor; relative humidity sensor; step counter sensor en een step detector sensor.

De paragrafen 4.3.3 en 4.3.4 beschrijven dat het verwerken van meetgegevens tot abstractere contexten niet altijd even eenvoudig is. Het kan nodig zijn om inconsistenties binnen meetwaarden te onderkennen, om daar vervolgens acties op te ondernemen. M.b.v. context reasoning kan men uit meetwaarden en vastgestelde subcontexten abstractere contexten afleiden. Voor context reasoning wordt opgemerkt, dat het inbrengen van kennis eigenlijk altijd noodzakelijk is. Alleen in eenvoudige situaties kan men volstaan met een puur data-driven machine learning benadering.

Paragraaf 4.4 geeft een korte beschrijving van een algemene context-aware architectuur, terwijl paragraaf 4.4.1 ingaat op contextmodelling. Vervolgens worden vijf subcontexten van de gebruikerscontext genoemd, te weten: environment context; personal context; task context; social context en spatio-temporal context [Göker en Myrhaug, 2002]. Deze indeling kan hulp bieden bij de verwerking van gemeten contexten en contextwaarden, om vervolgens m.b.v. context reasoning tot die contexten te komen, waarmee de context-aware applictie kan werken. Als voorbeeld van contextmodelling tools komt kort het JCAF framework aan bod, samen met de tools JCOOLS en MLContext die met JCAF (kunnen) werken.

**Gevolgen voor het project**
M.b.t. het project worden in de paragrafen 4.3.5 en 4.4.3 de volgende beslissingen genomen:

1. De focus van het project ligt bij de subcontexten Locatie en Tijd.
2. De modellering van de subcontexten wordt op object-georiënteerde wijze uitgevoerd.
3. Er wordt geen bijzondere aandacht aan context reasoning besteed.

### 4.6.2 Beantwoording vraag 2

Vraag 2 uit paragraaf 4.1.2 luidt:
*“Hoe kan men contextgegevens verrijken m.b.v. informatie/functionaliteit van derde partijen?”*

Uit hoofdstuk 4.5 blijkt dat men, d.m.v. webservices en/of API's, eenvoudig de functionaliteit van context-aware systemen kan vergroten. Zo is het eenvoudig mogelijk een postcode op te halen bij een gegeven locatie.

**Gevolgen voor het project**
In paragraaf 4.5.2 worden de volgende keuzen gemaakt:

1. Benader vrij te gebruiken API's/webservices.
2. Gebruik bij voorkeur de JSON varianten; gebruik alleen een SOAP gebaseerde service, als er geen JSON gebaseerde tegenhanger voorhanden is.

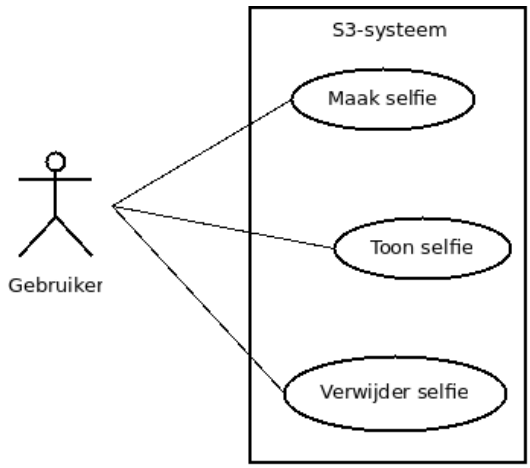
# 5 Beschrijving opgeleverde software

Dit hoofdstuk beschrijft het opgeleverde S3-systeem. De gerealiseerde functionaliteit en de globale architectuur vormen hierbij het startpunt. Vervolgens worden stapsgewijs alle samenstellende componenten besproken. Belangrijke gemaakte keuzes en beslisstingen worden beschreven en beargumenteerd.

Daarbij wordt binnen het project altijd hetzelfde uitgangspunt gehanteerd: houd alles zo eenvoudig mogelijk.

## 5.1 Geleverde functionaliteit

In deze paragraaf wordt het S3-systeem bekeken vanuit de eindgebruiker. Voor deze gebruiker zijn de belangrijkste drie Use Cases gerealiseerd. Deze Use Cases worden in onderstaande afbeelding weergegeven.




*Afbeelding 6: Use Cases voor eindgebruikers van het S3-systeem.*

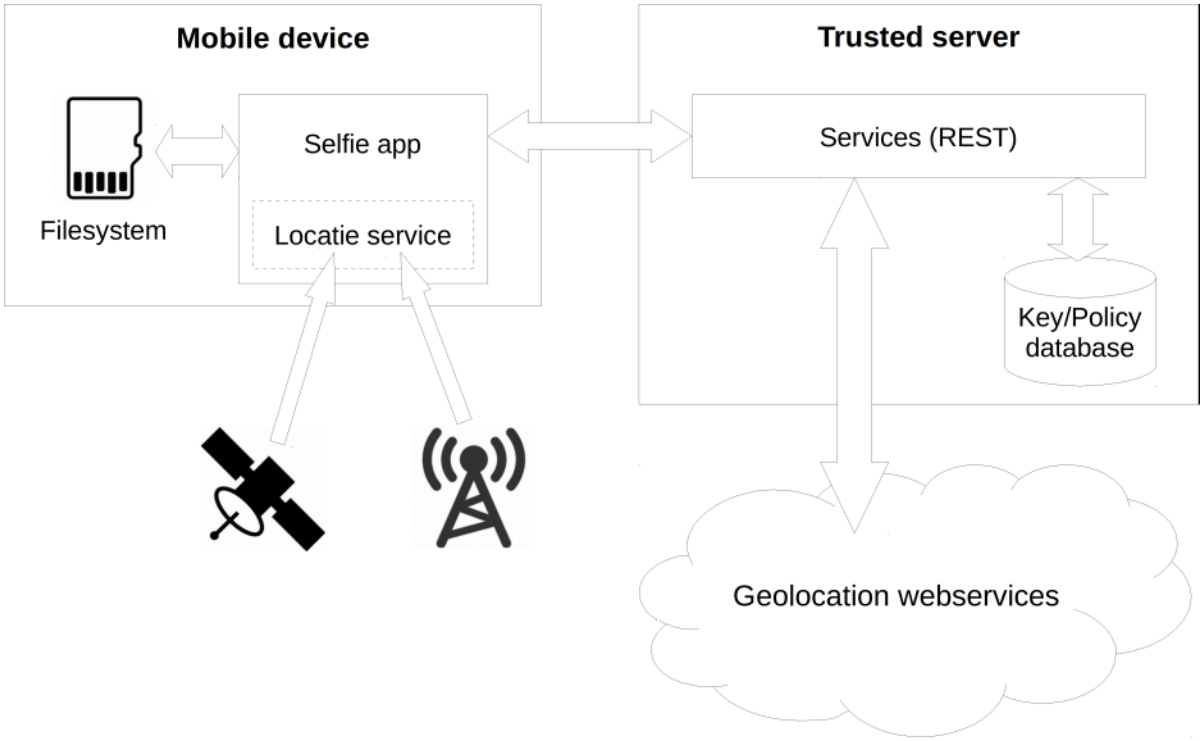
Omdat -volgens paragraaf 2.1- het realiseren van uitwisselfunctionaliteit voor selfies expliciet buiten de projectscope valt, wordt volstaan met deze drie Use Cases:

- Use case 1: Maak selfie.
- Use case 2: Toon selfie.
- Use case 3: Verwijder selfie.

De doelen van deze Use Cases liggen voor de hand en behoeven geen nadere toelichting. De GUI van de S3App biedt -daar waar en wanneer van toepassing- direct toegang tot deze functionaliteit. De drie Use Cases worden in paragraaf 5.3 verder uitgewerkt.

## 5.2 Client-server architectuur

Afbeelding 7 geeft de client-server architectuur van het gerealiseerde S3-systeem weer.




*Afbeelding 7: Globale architectuur S3-systeem.*

Bovenstaande architectuurschets bevat de volgende componenten.

Het Mobile device als client, met:

- de *Selfie app* (S3App), waarin de component *Locatie service* is opgenomen;
- het *Filesystem*, voor opslag van de encrypted selfies.

De Trusted server, met:

- de aangeboden *Services* (S3Server); deze zijn RESTful uitgevoerd, waarbij de communicatie verloopt via berichten in JSON-formaat;
- een *Key/Policy-database* (S3ServerDb); hierin worden de gebruikte keys opgeslagen.

Tenslotte maakt de Trusted server gebruik van Geolocation webservices. Deze services worden aangeboden door Google, zie paragraaf 4.5.2.

In de volgende subparagrafen worden enkele keuzes, die op dit abstractieniveau zijn gemaakt, nader toegelicht. Dit betreffen de taakverdeling tussen de app en de server, de gebruikte programmeertalen en frameworks, de keuze voor symmetrische encryptie en de keuze voor de S3ServerDb database.

### 5.2.1 Verdeling verantwoordelijkheden over S3App en S3Server

Bij de verdeling van de verantwoordelijkheden over S3App en S3Server wordt een duidelijke scheiding gemaakt tussen de opslag van de selfies en de opslag van de keys. S3App is verantwoordelijk voor het genereren van de keys en het beheer van de encrypted selfies. S3Server is verantwoordelijk voor het beheer van de keys die nodig zijn voor de encryptie en decryptie van de selfies.

De taken zijn als volgt over de twee hoofdcomponenten verdeeld:

**S3App**

1. Genereren van de key voor een nieuw gemaakte selfie.
2. Aanmelden gegenereerde key bij S3Server.
3. Volledig beheer van gemaakte selfies en de koppelingen naar hun keys.
4. De encryptie en decryptie van selfies.
5. Optionele invoer van expiration- en/of locatiepolicies, direct na aanmaak van de selfie.
6. Communiceren van deze eventueel toegevoegde policies naar S3Server.
7. Opvragen van de key bij S3Server indien gebruiker de selfie wil bekijken.

Ad 1 t/m 4

De keys worden wel door S3App gegenereerd en gebruikt, ze worden echter niet op het Android device zelf opgeslagen.

Algemeen

De selfie blijft op het Android device totdat deze of door de gebruiker wordt verwijderd of totdat S3App wordt gedeinstalleerd.

**S3Server**

1. Bieden van services aan S3App via een eenvoudige RESTful API.
2. Beheren geregistreerde keys en eventueel toegevoegde policies.
3. Verstrekken van de keys aan S3App, waarbij op -eventueel aanwezige- policies wordt gecontroleerd.
4. Communicatie met webservices van derden.

Ad 4

Diverse leveranciers van webservices vereisen registratie van de applicaties en/of de servers die van hun diensten gebruik maken. Uit praktische overwegingen worden externe webservices daarom uitsluitend vanuit S3Server benaderd en niet vanuit de client.

Algemeen

De functionaliteit van S3Server richt zich op het beheren van de keys met de eventueel daaraan gekoppelde policies. Verstrekken van een key vindt alleen plaats als aan eventueel gekoppelde policies is voldaan. S3Server richt zich op key-management, niet op de opslag van encrypted selfies.

### 5.2.2 Keuze voor symmetische encryptie

De encryptie en decryptie van selfies vindt op symmetrische wijze plaats. Naast een eenvoudiger implementatie, ligt de performance van symmetrische encryptie tot een factor 10.000 hoger dan bij vergelijkbare assymetrische encryptie [Pfleeger et al., 2015].

### 5.2.3 Toegepaste programmeertalen

S3App is met een combinatie van Java 7 en Groovy 2.4 gerealiseerd. De keuze voor Java 7 wordt hierbij afgedwongen door het Android 4.4 platform (actueel per november 2015). Voor S3Server wordt een combinatie van Java 8 en Groovy 2.4 gebruikt. Hierbij draait de S3Serverprogrammatuur onder de webserver Tomcat 8.
Deze keuzes zijn gemaakt daar ze aansluiten bij de ervaring van de opdrachtnemer en omdat Java sinds 1997 door de Open Universiteit tijdens de propedeuse van de informaticaopleiding wordt gedoceerd. De webserver Tomcat komt in de postpropedeusecursus “T21331 Webapplicaties: de serverkant” aan de orde.

Groovy is een dynamische uitbreiding bovenop Java. Hiermee worden op eenvoudige wijze elegante en efficiënte constructs -bekend uit talen als Python en JavaScript- aan de Java programmeertaal toegevoegd.

Voorbeeld 1
Vergelijk onderstaande Pythonstyle boolean expressie in Groovy (uit de PoliciesPolicy klasse):

```
result[0].formattedAddress[0..3] != key.locationPolicy.postalCode[0..3]
```

met het standaard Java equivalent:

```
!result[0].getFormattedAddress().substring(0, 4)
.equals(key.getLocationPolicy().getPostalCode().substring(0, 4))
```

Voorbeeld 2
Vergelijk het gebruik van JavaScriptstyle truthy en falsey in Groovy:

```
if (stringVariabele) { }
```

met het gebruikelijke Java idioom:

```
if (stringVariabele != null && !stringVariabele.equals(“”)) { }
```

Deze extra faciliteiten zijn de reden om Groovy binnen het S3-project toe te passen. Bijlage 1 geeft meer informatie over het gebruik van Groovy.

### 5.2.4 Keuze van toegepaste frameworks

Bij de realisatie van S3App en S3Server zijn diverse open source frameworks toegepast. De gemaakte keuzes worden nader beschreven in de paragrafen 5.4.2 (voor S3App) en 5.7.2 (voor S3Server).

Frameworks worden toegepast omdat:

1. het gebruik van frameworks leidt tot eenvoudiger te realiseren code, dus tot een effectiever en efficiënter ontwikkelproces;
2. de code en geleverde functionaliteit van frameworks doorgaans van betere kwaliteit is dan van eigen ad-hoc geschreven functionaliteit.

Om van deze voordelen te kunnen profiteren, moet wel aan enkele voorwaarden worden voldaan. Zo moeten actieve communities voor de betreffende frameworks bestaan en moeten betreffende frameworks een aanzienlijke hoeveelheid gebruikers hebben. Daarmee verkleind men het risico dat men ineens met een ‘dood’ framework zit opgescheept. Ook is het van belang dat binnen de open source projecten van deze frameworks een professioneel releasemanagement, inclusief testfasen, wordt gebruikt.

### 5.2.5 Keuze voor een NoSQL database voor S3ServerDb

Voor de dataopslag t.b.v. S3Server is niet gekozen voor een traditionele relationele database. In plaats daarvan wordt gebruik gemaakt van een NoSQL database van het type Document Database. Onderstaande argumenten om een NoSQL database binnen het S3-systeem toe te passen, zijn ontleend aan Sadalage en Fowler [2013]:

- NoSQL databases kennen geen expliciet databaseschema.
  De datastructuur binnen een NoSQL database wordt volledig door het domeinmodel van de applicatie bepaald. Er wordt geen gebruik gemaakt van starre databaseschema's, zoals deze binnen RDBMS'sen gebruikelijk zijn. Bij aanpassingen in het domeinmodel hoeft men dus niet ook nog een databaseschema aan te passen. Zeker als men werkt volgens het OTAP-principe, dan kan dit -zowel voor ontwikkelaars als DBA's- veel werk voorkomen. Wel kan het noodzakelijk zijn de reeds opgeslagen data aan te passen.

- Het afbeelden van classes op documents werkt eenvoudiger dan bij Object Rational Mapping (ORM). Binnen Document Databases ligt de nadruk op het werken met aggregaten [Evans, 2004]. Bij deze aggregaten gaat het om betekenisvolle samenstellingen van objecten, niet of deze data correct genormaliseerd wordt opgeslagen. Bij RDBMS'sen ligt de nadruk op het technisch effectief en efficient organiseren en opslaan van data, waarbij voorbij wordt gegaan aan de semantiek van de opgeslagen data.

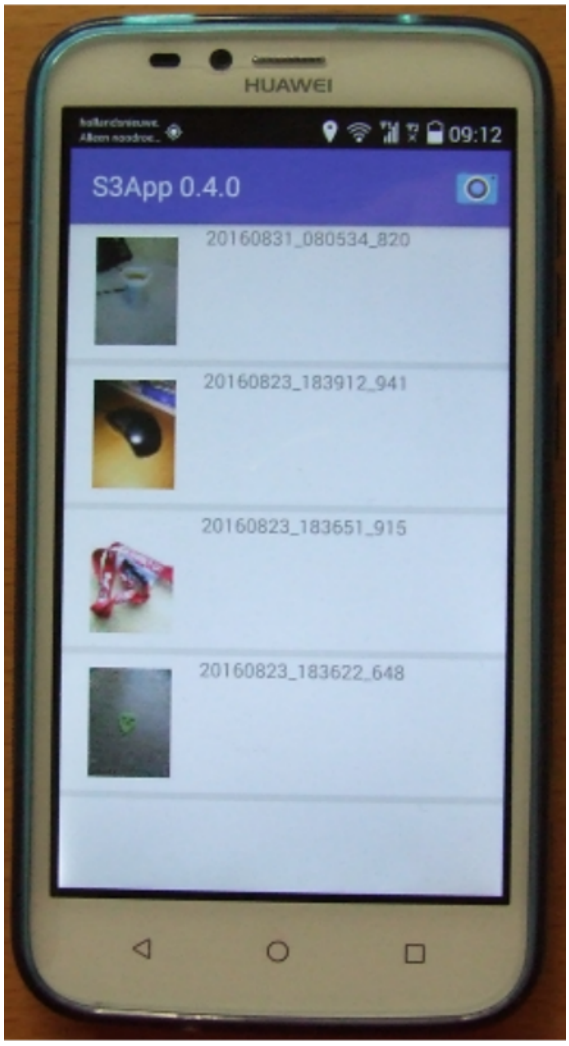
Er zijn meer argumenten om voor NoSQL databases te kiezen. Deze argumenten liggen op het gebied van clustering en Big Data [Sadalage en Fowler, 2013]. Dit valt echter buiten de scope van het S3-project.

## 5.3 Ontwerp van de client: S3App

Na een korte beschrijving van de GUI van S3App en enkele opmerkingen m.b.t. Android, wordt het ontwerp beschreven aan de hand van de drie use cases: Maak selfie, Toon selfie en Verwijder selfie.

### 5.3.1 De GUI van de app

De functionaliteit van de drie use cases is toegankelijk vanaf het eenvoudige hoofdscherm van de app, volgens onderstaande afbeelding.




*Afbeelding 8: Het hoofdscherm van S3App.*

Om een selfie te kunnen maken (use case 1), klikt men op het camera-icon rechts boven in het scherm. Als de selfie gemaakt en door de gebruiker geaccepteerd is, worden achtereenvolgens de schermen voor de tijdgebaseerde en de locatiegebaseerde policies getoond. Na het succesvol doorlopen van deze use case wordt een thumbnail van de selfie aan de gallery op het hoofdscherm toegevoegd.

Als er thumbnails in de gallery staan, kan men ook:

- een selfie tonen (use case 2) d.m.v. een korte klik op betreffende gallery item;
- een selfie verwijderen (use case 3) d.m.v. een lange klik op betreffende gallery item. Daarna verschijnt een dialoog, waar men de verwijderactie kan annuleren of bevestigen.

### 5.3.2 Enkele kenmerken van het Android platform

Het Android platform is gebaseerd op een event-driven architectuur. Daarbinnen zijn vier bouwstenen [Stuurman et al., 2014] van belang. Van deze vier bouwstenen wordt binnen S3App alleen gebruik gemaakt van Activities en Services.

**Activities**
Een Android Activity is een speciale Javaklasse, geassocieerd met één scherm. Activities kennen een specifieke lifecycle die bepalend is voor de opzet van de workflow van de app.
Voor S3App houdt dit in, dat alle geboden functionaliteit start bij de Activity MainActivity.

**Services**
Services draaien op de achtergrond en hebben geen schermen. Activities kunnen van services gebruik maken. Men kan Services toepassen om langlopende taken op de achtergrond uit te voeren, zonder dat de GUI van de app daar last van ondervindt.In S3App wordt één Service gebruikt voor de periodieke bepaling van de locatie: LocationService.

### 5.3.3 Use case 1: Maak selfie

'Maak selfie' is de meest complexe use case. Het schema in afbeelding 9 geeft de te doorlopen stappen weer. De vorm van dit schema houdt het midden tussen een klassieke flowchart en een UML activity diagram. Tevens zijn swimlanes toegevoegd zodat duidelijk wordt welke acties door S3App of door S3Server worden uitgevoerd. De startmarkering geeft aan dat de verwerking door een gebruikersactie wordt gestart; de eindemarkering geeft het einde van de verwerking t.b.v. de use case weer.
Om het diagram leesbaar te houden, wordt alleen het 'happy path' weergegeven. Ook bij de andere use cases worden alleen de happy paths getoond. Om de leesbaarheid verder te vergroten, bestaat het diagram uit drie hoofdfuncties: Nemen selfie, Genereren key en encryptie selfie en Toevoegen optionele policies.

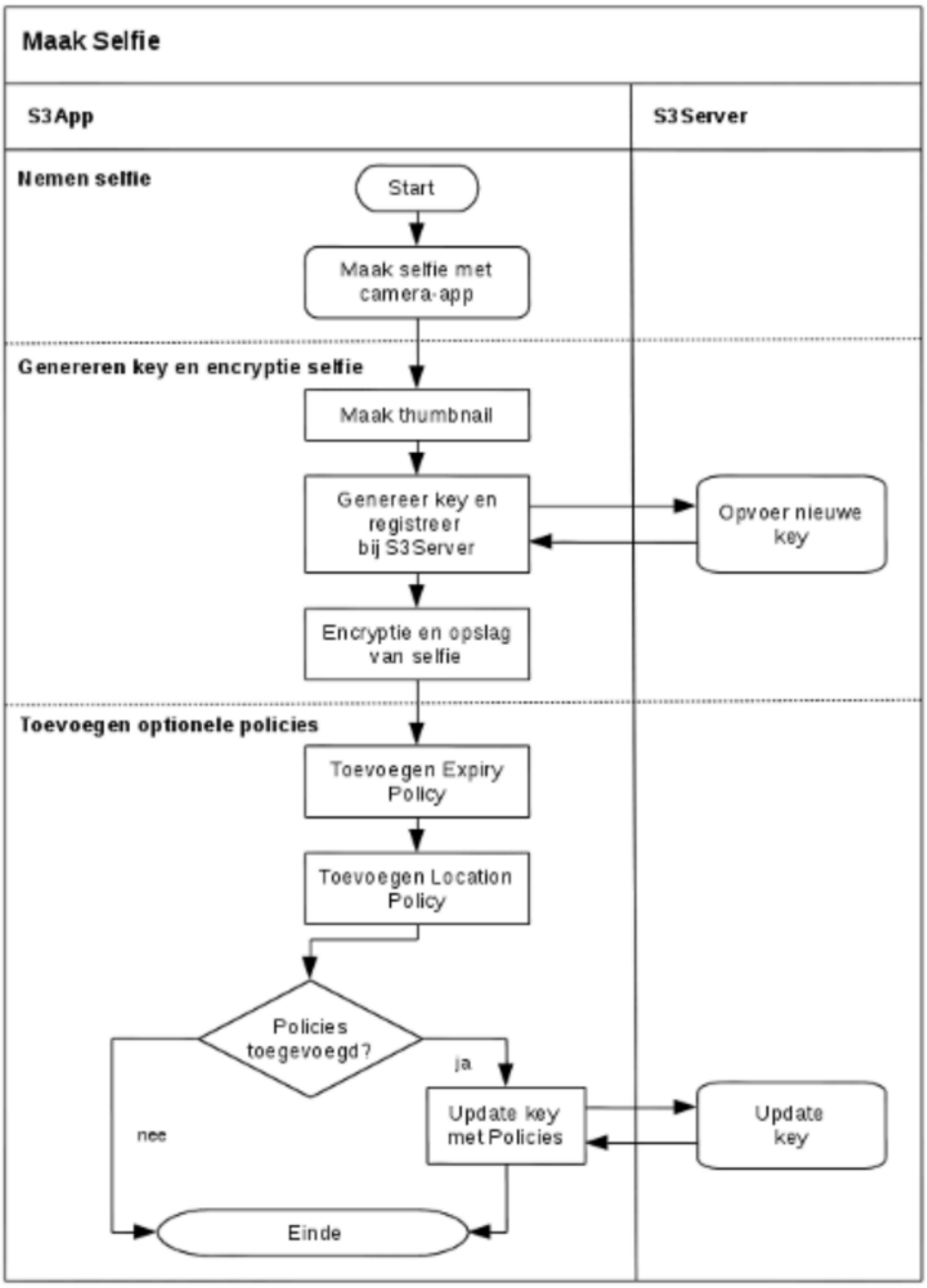
Voor de use case Maak selfie zijn de volgende keuzes gemaakt:

*S3App maakt gebruik van een standaard camera-app.*
Een selfie wordt m.b.v. een reeds geinstalleerde camera-app genomen, die de foto direct op het bestandssysteem opslaat. S3App zorgt vervolgens direct voor de encryptie en gooit de oorspronkelijke afbeelding weg. Dit levert een zeker risico op: de selfie staat enige tijd (enkele seconden) unencrypted op het device. Om de doorlooptijd van de realisatie beheersbaar te houden, is toch voor deze optie gekozen. De andere keuze leidt immers tot het moeten bouwen van eigen camera-app functionaliteit binnen S3App.

*Gebruik van Android Local store t.b.v. de selfies.*
Per selfie worden twee bestanden opgeslagen: een thumbnail (.jpg bestand) en een encrypted selfie (.S3-bestand). Android kent private directories voor dataopslag door apps [webref: Android Developers, 2016-5]. S3App maakt hier gebruik van. S3App kent twee subdirectories, één voor de thumbnails en één voor de .S3-bestanden. Een thumbnail en het overeenkomende .S3-bestand hebben, op hun extensie na, identieke bestandsnamen. Mocht de gebruiker S3App deinstalleren, dan worden deze private directories automatisch verwijderd.

*.S3-bestanden bevatten een keyId en een encrypted selfie.*
Na het bekendstellen van de gegenereerde key aan S3Server, geeft S3Server een uniek id voor de opgevoerde key terug. Na encryptie van de selfie wordt de gegenereerde key door de app 'vergeten'.
De verkregen id wordt, samen met de encrypted selfie, in een .S3-bestand in JSON-formaat opgeslagen.

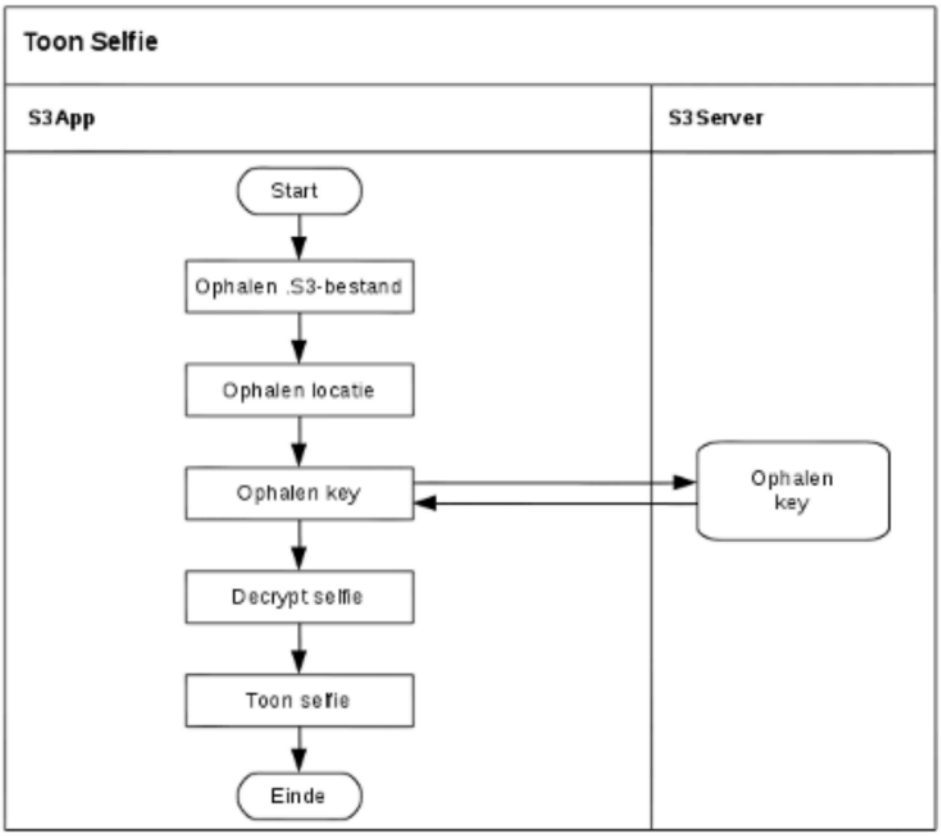
*Policies zijn optioneel.*
Als gebruiker geen policies aan een selfie koppelt, vindt alleen encryptie van betreffende selfie plaats.




*Afbeelding 9: Happy path flow voor use case 'Maak selfie'.*

### 5.3.4 Use case 2: Toon selfie

Onderstaande afbeelding toont het happy path, zoals gerealiseerd voor deze use case.



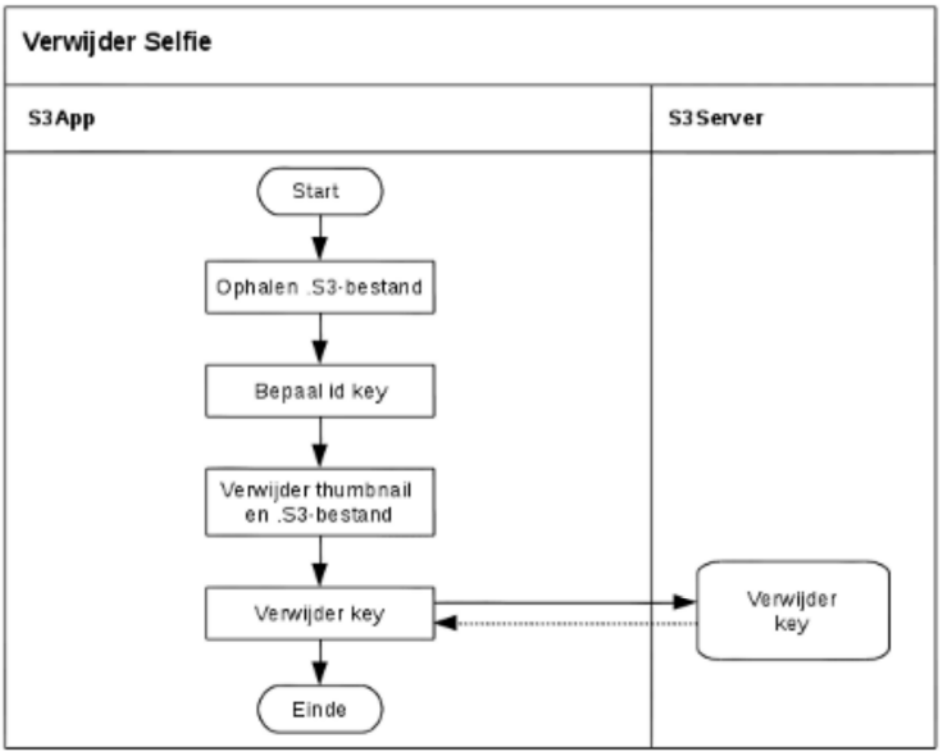

*Afbeelding 10: Happy path flow voor use case 'Toon selfie'.*

De naam van de aangeklikte thumbnail wordt gebruikt om de bestandsnaam van het .S3-bestand vast te stellen. Uit dit .S3-bestand kan de app zowel de encrypted selfie als de id van de benodigde key lezen.

Nadat de actuele locatiegegevens van het mobile device zijn opgehaald, wordt een poging gedaan om de correcte key bij S3Server op te halen. Als bij het gegeven id een key bestaat en de controles op optionele policies slagen, geeft S3Server de gewenste key terug. Na decryptie met deze key kan de selfie worden getoond en zal S3App de gebruikte key weer vergeten.
Als S3Server echter een foutmelding retourneert, zal S3App deze melding aan de gebruiker tonen.

### 5.3.5 Use case 3: Verwijder selfie

De laatste use case betreft het verwijderen van een selfie, zie onderstaande figuur.




*Afbeelding 11: Happy path flow voor use case 'Verwijder selfie'.*

De naam van de aangeklikte thumbnail wordt gebruikt om de bestandsnaam van het .S3-bestand vast te stellen. Uit dit .S3-bestand wordt de id van de gekoppelde key gelezen.

Vervolgens worden de thumbnail en het .S3-bestand uit het bestandssysteem verwijderd. Tenslotte wordt naar S3Server een delete-request, voorzien van het zojuist ingelezen id, voor de gekoppelde key gestuurd. Na de verwijderactie zal S3Server wordt weliswaar een response retourneren, maar deze wordt door S3App genegeerd.

## 5.4 Implementatiedetails S3App

Deze subparagraaf start met een overzicht van de genomen beslissingen t.b.v. S3App. Vervolgens worden de toegepaste frameworks globaal toegelicht. Tenslotte worden de implementatiedetails voor de drie use cases besproken.

### 5.4.1 Beslissingen t.b.v. S3App

Bij de implementatie van S3App zijn onderstaande beslissingen genomen.

*Symmetrtische encryptie geimplementeerd met AES.*
AES is een veel toegepaste en veilige encryptiemethode [Pfleeger et al., 2015]. Daarom wordt binnen S3 van AES gebruik gemaakt.

*AES-keys worden Base64-encoded met S3Server gecommuniceerd.*
Communicatie met S3Server verloopt via berichten in JSON-formaat. JSON is gebaseerd op ‘platte tekst’ en kent geen binaire typen. T.b.v. de communicatie wordt de binaire waarde van de key m.b.v. Base64-encoding omgezet naar een stringwaarde.

*Networkcalls worden uitgevoerd op de main thread.*
Bij lokaal draaiende GUI applicaties is het gebruikelijk dat langdurende operaties op een aparte thread worden uitgevoerd. Hieronder valt ook communicatie via een netwerk. Om de implementatie van S3App eenvoudig te houden, worden de netwerkoperaties binnen de main thread van S3App uitgevoerd.
In alle gevallen geldt dat verdere verwerking binnen S3App niet zinvol is, als betreffende netwerkoperatie faalt. De enige uitzondering is use case ‘Verwijder selfie’. Hier wordt de response vanuit S3Server altijd genegeerd.

*Android registratie emailadres vormt identificatie t.b.v. S3Server.*
Bij alle communicatie naar S3Server wordt ook het Android registratie-emailadres van de gebruiker meegestuurd. Dit emailadres vormt een eenvoudige identificatie van de gebruiker aan S3Server.

*De thumbnails hebben unieke (bestands)namen.*
Door de thumbnails unieke namen te geven (en deze namen ook voor de .S3-bestanden te gebruiken), kunnen de thumbnails eenvoudig aan de selfies gekoppeld worden. Dit leidt tot een eenvoudiger implementatie van S3App. Overigens werken de standaard camera-apps doorgaans ook op deze manier.

*Definitie berichtformaten m.b.v. Data Transfer Objects (DTO’s).*
Het berichtenverkeer tussen S3App en S3Server maakt gebruik van JSON. Om eenvoudig met JSON te kunnen werken worden DTO’s [Fowler et al., 2003] toegepast. DTO’s zijn objecten met placeholders voor de te transporteren gegevens. Met Jackson (zie volgende paragraaf) kunnen deze DTO’s op eenvoudige wijze van/naar JSON worden geserialiseerd.
De DTO’s worden dus feitelijk als formele definities van de JSON berichtstructuren toegepast.

*Policy m.b.t. de tijd geeft het tijdstip aan waarop geldigheid van selfie verloopt.*
Dit is een gemakkelijk te realiseren policy, waarmee op eenvoudige wijze kan worden getoond hoe men de context tijd kan toepassen m.b.t. de access-control op de encrypted selfies.

*Locatiepolicy werkt o.b.v. de postcodecijfers van het gebied waar selfie getoond mag worden.*
Tijdens de realisatie bleek dat de Google API (benaderd via S3Server) wel de goede postcodecijfers, maar niet altijd de goede letters bij een gegeven locatie retourneerde. Daarop is de keuze gemaakt om de locatiepolicy te beperken tot de postcodecijfers.

### 5.4.2 Toegepaste frameworks binnen S3App

Hieronder volgt een korte toelichting op de belangrijkste frameworks die binnen S3App zijn gebruikt. In de bijlagen worden meer details m.b.t. deze frameworks gegeven.

**Jackson**
Jackson is een Java framework dat binnen S3App gebruikt wordt om objecten eenvoudig van/naar JSON te serialiseren. Groovy biedt deze functionaliteit weliswaar out-of-the-box, maar dit blijkt op een smartphone dusdanig veel resources te vereisen, dat inzet van een andere oplossing noodzakelijk is.

**OkHttp**
OkHttp is een Java framework om op eenvoudiger wijze met HTTP-requests te kunnen werken.

**SwissKnife**
SwissKnife is een Groovy framework voor Android. Het heeft als doel het werken met de Android API te vereenvoudigen.

### 5.4.3 Implementatie use case ‘Maak selfie’

Het in afbeelding 8 getoonde hoofdscherm van S3App wordt aangestuurd door de klasse MainActivity. Vanuit de MainActivity kunnen de acties voor alle use cases worden gestart.

Bij de implementatie van de use case ‘Maak selfie’ zijn de volgende keuzes gemaakt:

*S3App benadert S3Server minimaal één keer tijdens het maken van een selfie.*
Vanwege het gebruik van een standaard camera-app is het belangrijk dat de genomen selfie zo snel mogelijk encrypted wordt opgeslagen. Daarom wordt meteen na acceptatie van de selfie in de camera-app gestart met:

- het genereren van de AES-key;
- opvoeren van deze key op S3Server;
- encrypten en opslaan van de selfie;
- weggooien van het oorspronkelijk fotobestand.

De gebruikers merken hier niets van, zij kijken op dat moment naar de schermen voor de toevoeging van één of meer policies. Eventueel door gebruikers opgegeven policies worden later m.b.v. een updateactie aan de eerder opgevoerde AES-key toegevoegd.

*Activities voor te koppelen Policies worden genest aangeroepen.*
Android Activities kunnen andere Activities opstarten, waarbij de alleen initiërende Activity de resultaten uit de opgestartte Activity kan verwerken. Android kent naast Activities ook Fragments, waarmee men gedeelten van schermen aan kan sturen. Om de implementatie van S3App zo eenvoudig mogelijk te houden, wordt geen gebruik van Fragments gemaakt.

Binnen de implementatie van deze use case kunnen maximaal twee policies worden opgegeven. Voor beide policies is één scherm inclusief Activity toegevoegd, te weten: LocationActivity en ExpirationActivity. De laatst beschreven beslissing houdt in, dat vanuit MainActivity de ExpirationActivity wordt aangeroepen en dat daarna LocationActivity vanuit ExpirationActivity wordt gestart.

### 5.4.4 Implementatie use case ‘Toon selfie’

Als de gebruiker op een thumbnail op het hoofdscherm klikt, wordt vanuit MainActivity direct doorgeschakeld naar de SelfieDialog klasse. Deze SelfieDialog klasse is verantwoordelijk voor alle uit te voeren acties van deze use case (zoals eerder weergegeven in afbeelding 10), waaronder het uiteindelijk tonen van de geselecteerde selfie.
Indien S3Server met een foutmelding reageert, dan wordt i.p.v. de gewenste selfie de ontvangen foutmelding getoond.

In afbeelding 10 staat één actie die nog moet worden toegelicht: ‘Ophalen locatie’.
De op te halen locatiegegevens zijn eerst door een andere programmacomponent vastgesteld. Daartoe beschikt S3App over de LocationService component. Deze Service is grotendeels overgenomen van [webref: Android Developers, 2016-2]. LocationService wordt opgestart vanuit de MainActivity en sluit automatisch af wanneer S3App wordt afgesloten. De LocationService staat dus volledig onder regie van S3App.

Om de vastgestelde locatie aan de rest van S3App beschikbaar te stellen, is gekozen voor de meest eenvoudige oplossing: de locatiegegevens worden in een onbeschermde public variabele (in klasse VolatileLocationData) geplaatst.

Binnen S3App zal dit geen problemen opleveren, want:

- LocationService schrijft maximaal 1x per 20 seconden een waarde in deze variabele;
- LocationService is het enige proces dat naar deze variabele schrijft;
- de overige delen van S3App voeren dus alleen leesacties op deze variabele uit.

### 5.4.5 Implementatie use case ‘Verwijder selfie’

Deze functionaliteit (volgens afbeelding 11) is geheel binnen de klasse MainActivity geimplementeerd. De verwerking wordt gestart als Android een longclick op een thumbnail waarneemt.
De code voor deze verwerking is voor het grootste gedeelte opgenomen binnen de Android AlertDialog die wordt getoond om de gebruiker te laten kiezen tussen doorgaan met of annuleren van de verwijderactie.

## 5.5 Mogelijkheden ter uitbreiding en verbetering van S3App

### 5.5.1 Mogelijke uitbreidingen

Hierbij enkele uitbreidingsmogelijkheden en aandachtspunten voor S3App:

*Toevoegen uitwisselfunctionaliteit voor selfies.*
In de oorspronkelijke opdracht was het de bedoeling, dat de implementatie van ‘Securely Sharing Selfies’ ook daadwerkelijk selfies zou kunnen delen. Zie ook paragraaf 8.2.1.

*Herzien van policies door de eigenaar van de selfie mogelijk maken.*
Men kan deze functionaliteit aan de eigenaar ter beschikking te stellen. Maar dan alleen zolang de selfie nog niet is gedeeld. Men kan echter ook besluiten dat de eigenaar altijd dit recht heeft. S3Server biedt updatefunctionaliteit die hiervoor kan worden gebruikt.

### 5.5.2 Mogelijke verbeteringen

S3App is zeker nog niet perfect. Onderstaande punten zijn voor verbetering vatbaar:

*Betere foutafhandeling.*
Een voorbeeld:
S3Server biedt heartbeat-functionaliteit, zodat men eenvoudig kan controleren of de S3Server bereikbaar is. Binnen S3App wordt daar nog geen gebruik van gemaakt. Daardoor is het gedrag van S3App bij een niet of moeilijk bereikbare S3Server nu nog onvoorspelbaar.

*Geen unit- en/of integratietests gerealiseerd.*
S3App kan alleen ‘functioneel’ worden getest. Dat betekent dat men ‘met de hand’ enkele scenario’s moet doorlopen om te kijken of de app naar behoren werkt. Vanwege de afhankelijkheid t.o.v. S3Server moet dan ook de achterliggende database op de server worden gecontroleerd. Uiteraard is dit niet de gewenste situatie.

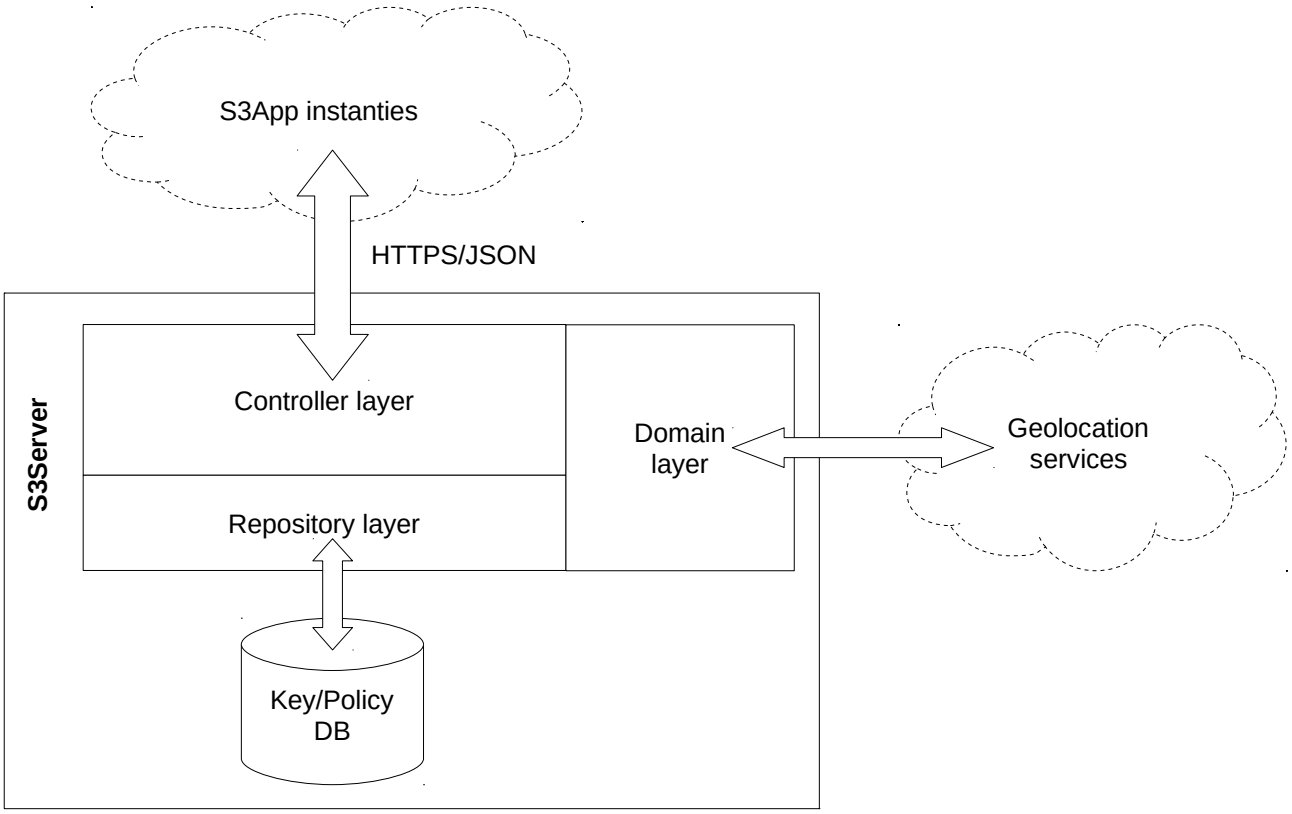
*Geen geautomatiseerde tests voor de GUI gerealiseerd.*
Zie voorgaand punt.

*Na deinstallatie van de app blijven de gegeneerde keys op S3Server achter.*
Bij het deinstalleren van een app wordt geen code van de betreffende app uitgevoerd. Bij deinstallatie van S3App blijven daardoor de door de app gegenereerde keys op S3Server achter. Alle encrypted selfies verdwijnen ook tijdens deinstallatie van S3App. De op S3Server achterblijvende sleutels veroorzaken daardoor datavervuiling binnen de S3ServerDb.
Waarschijnlijk is het in dit geval beter om binnen S3Server extra functionaliteit op te nemen, die op gezette tijden de database schoont.

## 5.6 Ontwerp van de server: S3Server

### 5.6.1 Globaal ontwerp S3Server

Afbeelding 12 toont de architectuur van S3Server.




*Afbeelding 12: Architectuur S3Server*

In de architectuur voor S3Server worden de volgende lagen onderkend:

1. Controller layer.
   De controller layer is verantwoordelijk voor de communicatie met instanties van S3App. Requests vanuit S3App instanties worden hier ontvangen en de benodigde acties worden gestart en gedelegeerd naar de andere layers. De resultaten worden, indien van toepassing, naar betreffende S3App instantie gecommuniceerd.
2. Domain layer.
   De domain layer bevat de feitelijk businesslogica van S3Server. Daarom communiceert deze layer met de Geolocation services. Ook de classes, die afgebeeld worden op documents binnen de Key/Policy-database, zijn in deze layer opgenomen.
3. Repository layer.
   De repository layer is verantwoordelijk voor de communicatie met de database. Deze laag bevat alleen de voor S3Server benodigde repositories.

In de volgende paragrafen (5.6.2 t/m 5.6.4) worden deze layers afzonderlijk toegelicht.

### 5.6.2 De Controller layer

De controllerlaag is een implementatie van het Service Layer pattern [Fowler et al., 2003], waarbij de beschikbare S3Server-operaties RESTful [Fielding, 2000] richting S3App worden aangeboden. Deze laag bevat één Java controllerklasse (SymmetricKeyController) die toegang biedt tot alle beschikbare operaties voor het key-management.

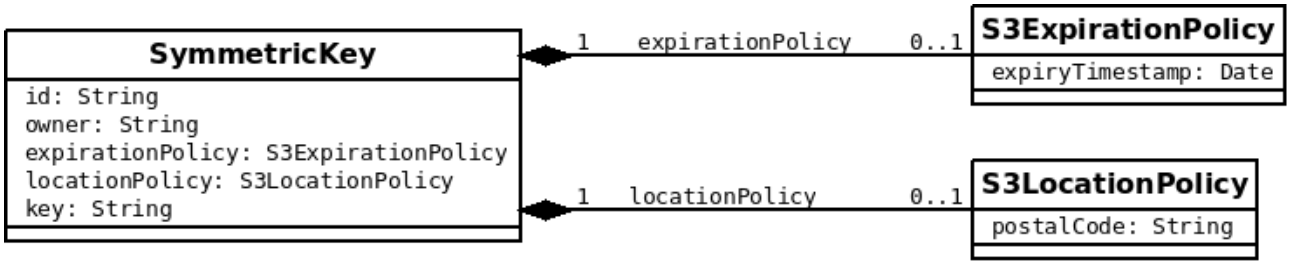
Communicatie met instanties van S3App vindt plaats via berichten in JSON-formaat.

### 5.6.3 Het domeinmodel in de Domain layer

S3Server kent een eenvoudig domeinmodel dat bestaat uit de SymmetricKey met optioneel gekoppelde policies. Binnen SymmetricKey zijn de velden id, owner en de key verplicht:

- *id* is de unieke identifier, die door de database wordt gegenereerd.
- *owner* is het tekstveld, bedoeld om het registratie-emailadres van de gebruiker (die bijbehorende selfie heeft gemaakt) op te slaan.
- *key* is het tekstveld waarin betreffende key Base64-encoded wordt opgeslagen.

Zowel de expirationPolicy als de locationPolicy zijn optioneel. Dit leidt tot onderstaand klassendiagram voor het domeinmodel.




*Afbeelding 13: Klassendiagram voor het domeinmodel.*

In dit diagram komt tot uiting dat alleen SymmetricKey de eigenschappen van een Entity heeft [Evans, 2004]. De opgeslagen key, gecombineerd met de optionele policies, vormt een identificeerbaar geheel. Dat geldt niet voor beide Policytypen. Deze objecten hebben bestaansrecht vanwege de data die ze bevatten, maar ze zijn alleen betekenisvol in combinatie met een SymmetricKey. Daarom zijn hun relaties met SymmetricKey weergeven als composities [Larman, 2012]. De Policy objecten hebben zelf geen logisch concept van identiteit. Deze Policy objecten zijn Value Objects [Evans, 2004].

### 5.6.4 De Repository layer

De Repository layer bevat de Repositories [Evans, 2004], die nodig zijn om op eenvoudige manier met de data binnen de database te kunnen werken. Vanwege het eenvoudige domeinmodel is alleen een Repository t.b.v. SymmetricKey nodig.

## 5.7 Implementatiedetails S3Server

### 5.7.1 Keuze voor MongoDB als NoSQL Document Database

Volgens db-engines.com is het open source MongoDB al geruime tijd de meest populaire Document Database [webref: db-engines.com, 2016]. Dat houdt o.a. in dat MongoDB door de meeste gangbare programmeertalen wordt ondersteund en dat veel online informatie direct beschikbaar is. Onderstaande afbeelding geeft de Top10 van databases per augustus 2016 weer.

**DB-Engines Ranking**

The DB-Engines Ranking ranks database management systems according to their popularity. The ranking is updated monthly.

Read more about the method of calculating the scores.

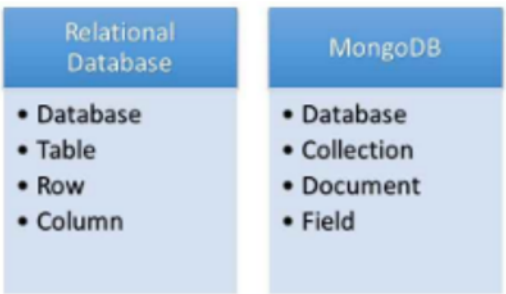
trend chart

309 systems in ranking, August 2016

| Rank Aug 2016 | Rank Jul 2016 | Rank Aug 2015 | DBMS | Database Model | Score Aug 2016 | Score Jul 2016 | Score Aug 2015 |
|---|---|---|---|---|---|---|---|
| 1. | 1. | 1. | Oracle | Relational DBMS | 1427.72 | -13.81 | -25.30 |
| 2. | 2. | 2. | MySQL | Relational DBMS | 1357.03 | -6.25 | +65.00 |
| 3. | 3. | 3. | Microsoft SQL Server | Relational DBMS | 1205.04 | +12.16 | +96.39 |
| 4. | 4. | 4. | MongoDB | Document store | 318.49 | +3.49 | +23.84 |
| 5. | 5. | 5. | PostgreSQL | Relational DBMS | 315.25 | +4.10 | +33.39 |
| 6. | 6. | 6. | DB2 | Relational DBMS | 185.89 | +0.81 | -15.35 |
| 7. | 7. | ↑ 8. | Cassandra | Wide column store | 130.24 | -0.47 | +16.24 |
| 8. | 8. | ↓ 7. | Microsoft Access | Relational DBMS | 124.05 | -0.85 | -20.15 |
| 9. | 9. | 9. | SQLite | Relational DBMS | 109.86 | +1.32 | +4.04 |
| 10. | 10. | 10. | Redis | Key-value store | 107.32 | -0.71 | +8.51 |

*Afbeelding 14: Populariteit databases volgens [webref: db-engines.com, 2016] per augustus 2016.*

NoSQL databases kennen andere terminologie dan de bekende relationele databases. Afbeelding 15 geeft de vertaling van enkele RDBMS-begrippen naar MongoDB weer.




*Afbeelding 15: RDBMS terminologie versus MongoDB terminologie. (http://csharpcorner.mindcrackerinc.netdna-cdn.com/UploadFile/1f5150/introduction-to-mongodb/Images/RDBMS%20vs%20MongoDB.jpg). Geraadpleegd 16 augustus 2016.*

### 5.7.2 Toegepaste frameworks S3Server

Hieronder volgt een korte toelichting op de belangrijkste frameworks die binnen S3Server zijn gebruikt. In de bijlagen worden meer details m.b.t. deze frameworks en hun toepassing binnen S3Server gegeven.

**Google Maps Services**
Dit is de in paragraaf 4.5.1 genoemde client-library om vanuit Java eenvoudig de Google Maps Services te kunnen benaderen. De developer hoeft daardoor niet zelf de benodigde HTTP-requests samen te stellen.

**Hibernate Validator**
Deze validator maakt het mogelijk om op declaratieve wijze restricties op objectattributen te definiëren. De toegepaste versie van Hibernate Validator is de reference implementation voor Bean Validation volgens Java standaard JSR349.

**Spock framework**
Framework voor (unit)testing. Spock is een uitbreiding op JUnit en werkt met Groovy. Omdat Spock een uitbreiding van JUnit is, kunnen Spock tests eenvoudig als JUnit-testgevallen worden uitgevoerd.

**Spring framework**
Een bekend framework om enterprise grade Java (web)applicaties te bouwen. Het biedt zowel een alternatief als uitbreidingen voor het Java EE platform. Binnen S3Server is de onderlinge samenhang van de layers geimplementeerd met het Spring framework.

**Spring Data (voor MongoDB)**
Spring Data is een uitbreiding op het Spring framework. Van Spring Data worden binnen S3Server drie aspecten gebruikt:

- Mapping van Entities [Evans, 2004] op MongoDB documents.
- Repositories [Evans, 2004], een Repository abstraheert een MongoDB collectie naar een Java Collection.
- Automatische validatie m.b.v. Bean Validation bij opslag van gegevens in MongoDB.

**Spring MVC**
Spring MVC is een uitbreiding op het Spring framework dat volgens het Front Controller pattern [Fowler et al., 2003] werkt. Binnen S3Server wordt alleen de REST functionaliteit van Spring MVC gebruikt.

### 5.7.3 Implementatie van de Controller layer

Om de symmetrische keys op S3Server te kunnen benaderen, zijn de standaard CRUD operaties gedefinieerd. Deze CRUD operaties worden geimplementeerd door de Java methods findKey, createKey, updateKey en deleteKey binnen de SymmetricKeyController klasse.
Deze methods coördineren de uit te voeren acties en delegeren direct naar de domain- en repository layers. De resultaten van de uitgevoerde bewerkingen worden -indien van toepassing- impliciet naar JSON geconverteerd en vervolgens naar betreffende S3App instantie geretourneerd. Deze automatische JSON-conversie is mogelijk door toepassing van Spring MVC.

Naast de SymmetricKeyController is ook een PingContoller toegevoegd. Deze controller kan vanuit S3App benaderd worden voor heartbeat-functionaliteit. S3App maakt hier echter nog geen gebruik van.

### 5.7.4 Implementatie Domain layer

In de domain layer zijn zowel de businesslogica als het domeinmodel ondergebracht.

Voor de drie objecten uit het klassendiagram (afbeelding 13) geldt dat ze als eenvoudige Java Beans zijn geimplementeerd. Qua businesslogica is één klasse toegevoegd: PoliciesPolicy. Deze klasse bevat de logica om de vanuit S3App verstrekte gegevens tegen de policies (indien deze bij een key van toepassing zijn) te matchen. Als daarbij fouten optreden, zorgt deze klasse tevens voor de juiste foutmelding.
In geval van een aanwezige expirationPolicy wordt gecheckt of het tijdstip waarop de check wordt uitgevoerd, vóór de expiryTimestamp ligt. Bij een aanwezige locationPolicy wordt de Google Geolocation webservice benaderd met de locatiecoördinaten. De geretourneerde postcode wordt vervolgens vergeleken met de postcode uit de policy.

### 5.7.5 Implementatie Repository layer

De repository layer bevat één Java interface: SymmetricKeyRepository. SymmetricKeyRepository heeft één bijzonder kenmerk, deze interface is leeg. Zie afbeelding 16.

```
public interface SymmetricKeyRepository extends MongoRepository<SymmetricKey, String> {

}
```

*Afbeelding 1 : ymmetricKeyRepository Interface zoals opgenomen in de Repository layer.*

Bovenstaande definitie declareert een nieuwe repository o.b.v. de MongoRepository Interface (uit het Spring Data framework) voor de Entity SymmetricKey die een id van type String heeft.

Dit is voldoende om, at runtime, de benodigde implementatieklasse automatisch te laten genereren. Deze gegenereerde klasse biedt vervolgens diverse methods (w.o. de standaard CRUD acties), waarmee men de SymmetricKey collectie binnen MongoDB kan benaderen. De SymmetricKeyController maakt daar dan ook direct gebruik van.

### 5.7.6 Gegevensvalidatie en opslagstructuur in S3ServerDB

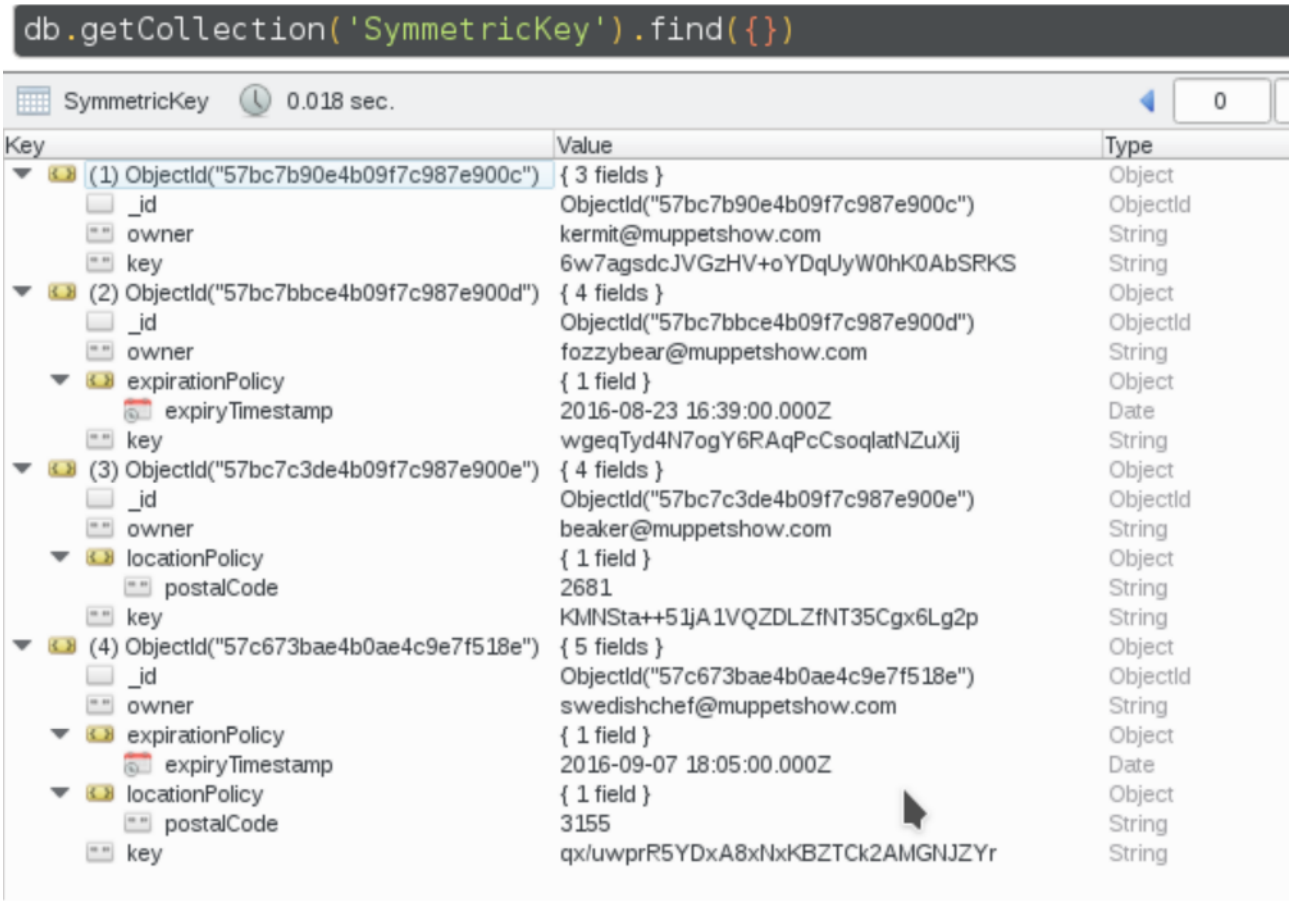
De SymmetricKey Entities worden, m.b.v. Spring Data, rechtstreeks gemapped op de documents binnen de MongoDB SymmetricKey collectie. De data uit het klassendiagram (van afbeelding 13) wordt als één aggregaat [Evans, 2004] op één MongoDB document geprojecteerd.

De velden binnen SymmetricKey zijn voorzien van Bean Validation annotaties. Dit zorgt ervoor dat validatie van de geannoteerde velden plaatsvindt, vóórdat de gegevens in MongoDB worden opgeslagen. Figuur 17 toont het veld owner als voorbeeld. Dit veld mag niet leeg zijn en de inhoud moet voldoen aan een geldig emailadresformaat. Ook de bijbehorende foutmeldingen zijn gespecificeerd.

```
@NotEmpty(message="Veld owner mag niet leeg zijn.")
@Email(message="Veld owner moet een geldig emailadres bevatten.")
String owner
```

*Afbeelding 17: Voorbeeld Bean Validation op veld owner in SymmetricKey*

Onderstaande figuur geeft een willekeurige momentopname van de SymmetricKey collectie. Hierin zijn diverse verschijningsvormen van SymmetricKey documenten, met de structuur volgens het klassendiagram uit afbeelding 13, duidelijk zichtbaar.




*Afbeelding 18: Momentweergave SymmetricKey collectie m.b.v. Robomongo (http://robomongo.org/).*

Afbeelding 18 toont tevens dat timestamps in ‘Zulu-tijd’ worden opgeslagen. Zulu-tijd is de standaard Greenwich Mean Time (GMT), echter zonder zomertijd. Dit betekent dat de timestamps automatisch volgens de referentie-tijdszone worden opgeslagen. Het S3 systeem is daardoor niet gevoelig voor verschillen tussen tijdzones waarbinnen de gebruikers zich kunnen bevinden.

## 5.8 Mogelijkheden voor uitbreiding en verbetering van S3Server

### 5.8.1 Mogelijke uitbreidingen

Hierbij enkele mogelijkheden en aandachtspunten voor uitbreidingen van S3Server:

*Structuur SymmetricKey aanpassen om sharing van selfies te faciliteren.*
In paragraaf 8.2.1 wordt hier nader op ingegaan.

*Faciliteren van een breder palet aan locatiegerelateerde policies.*
Zie paragraaf 8.2.2.

De structuren in bovengenoemde twee punten zijn redelijk eenvoudig te implementeren. De MongoDB zal deze wijzigingen direct ondersteunen. Alleen bij de tweede suggestie zal het nodig zijn een deel van de reeds opgeslagen SymmetricKey-data binnen MongoDB aan te passen aan de gewijzigde structuur.

### 5.8.2 Mogelijke verbeteringen

Ook S3Server is ook voor verbetering vatbaar:

*Communicatie tussen S3App en S3Server verloopt nog niet via HTTPS.*
Dit houdt in dat de communicatie onversleuteld via standaard HTTP plaatsvindt. Om de vertrouwelijkheid van deze communicatie te verhogen en Man-in-the-Middle attacks te bemoeilijken, dient deze communicatie via HTTPS te verlopen.

*Meegestuurd gebruikersregistratie-emailadres in App-Server communicatie is een URL-parameter.*
Daarmee is dit emailadres eenvoudig zichtbaar voor uitvoerders van Man-in-the middle (MITM) attacks. Dit emailadres moet in de versleutelde body van een HTTPS bericht (zie vorig punt) worden opgenomen.

*Er gaan teveel gegevens “over de lijn” tijdens de app-server communicatie.*
Dit is acceptabel voor een Proof-of-Concept, maar niet in een productief in te zetten systeem. Deze situatie is redelijk eenvoudig te wijzigen door meer (kleine) DTO’s toe te voegen, afhankelijk van de gegevens die wel over de lijn moeten gaan.

*Checks op policycompliance zijn te procedureel geimplementeerd (object PoliciesPolicy).*
Met het oog op de toekomst is implementatie van de checks op policycompliance binnen de Policies een betere keuze. Dit geldt met name voor de locatiegebaseerde policy.

## 5.9 Onderkende aanvallen op het S3-systeem

In deze paragraaf worden de onderkende aanvallen beschreven. Daar waar mogelijk, wordt aangegeven hoe deze aanvallen door S3App en/of S3Server tegen worden gegaan, of wat hiertoe nog binnen het S3-systeem geimplementeerd moet worden.

Man-in-the-Middle attack tussen S3App en S3Server.
Om Man-in-the-Middle attacks te bemoeilijken, is het van belang om ook de communicatie tussen S3App en S3Server te encrypten. Met communicatie via HTTPS kan dit relatief eenvoudig worden gerealiseerd.

Aanvaller heeft toegang tot de .S3-bestanden.
De app-private directories zijn toegankelijk voor de eindgebruiker of voor anderen die toegang tot de smartphone hebben verkregen. Bijvoorbeeld via een bestandsbeheer-app, via een uit een smartphone ontvreemde SD-kaart of via een USB-verbinding vanaf de eigen computer.

De aanvaller kan vervolgens:

1. Zelf een .S3-bestand inlezen en daar het keyId van de benodigde AES-key vinden.

2. Proberen m.b.v. de gevonden keyId betreffende AES-key bij S3Server op te vragen.
   Voor een succesvolle opvraagactie heeft de aanvaller onderstaande gegevens nodig:
   - Servernaam/IP-adres van S3Server.
   - Het poortnummer van de S3Serverprogrammatuur.
   - Het Android registratie-emailadres van de eigenaar van de encrypted selfie.
   - De juiste locatiegegevens (als bij S3Server bij betreffende key een locatiepolicy bekend is).

   Met een packetsniffer als bitShark (https://play.google.com/store/apps/details?id=blake.hamilton.bitshark), zijn de eerste twee gegevens vlot te achterhalen. Op dit moment is het emailadres van de gebruiker ook direct bekend, omdat dit als URL-parameter bij het POST-request is opgenomen (zie opmerking onder paragraaf 5.8.2).

3. Als de aanvaller de gegevens uit punt 2 op de juiste manier in een POST-request verwerkt, zal S3Server de gevraagde key retourneren.

Indien het S3-systeem in productie zou gaan, is het van belang dat de gebruikersidentificatie o.b.v. het emailadres wordt vervangen door een degelijker vorm van ‘shared secret’, zoals bijvoorbeeld een token of een unique user identifier.

Aanvaller probeert toegang te krijgen tot MongoDB.
Vanwege het Proof-of-Concept karakter van het S3-project staat MongoDB op dezelfde server als de S3Serverprogrammatuur.
In een productieve situatie zal men deze inrichting anders kiezen: aparte servers voor S3Server en MongoDB. Daarbij zorgt men ervoor dat de MongoDB-server niet vanaf het internet te benaderen is, zodat men alleen vanaf het eigen bedrijfsnetwerk beheerwerkzaamheden uit kan voeren.

# 8 Conclusies en aanbevelingen

## 8.1 Conclusies

Hoewel het huidige S3-systeem slechts twee subcontexten in beschouwing kan nemen, bevat het S3-systeem een succesvolle implementatie van Context-Driven Privacy. T.o.v. de eerder genoemde systemen BlogCrypt [Paulik et al., 2010] en P3 [Ra et al., 2013] neemt het S3-systeem de gebruikerscontext in acht, wanneer om toegang tot de content wordt verzocht.
Daar waar zowel BlogCrypt als P3 nauwelijks aandacht besteden aan het keymanagement, biedt het S3-systeem serverside keymanagement. Ook dit is een uitbreiding t.o.v. BlogCrypt en P3.

De opdrachtgever beschikt nu dus over een werkend prototype, met een Context-Driven Privacy implementatie voor toegang tot selfies via een app, voorzien van serverside keymanagement. Dit prototype kan door een volgende groep studenten verder verbeterd en uitgebouwd worden.

Wel is tijdens de realisatie gebleken dat het vaststellen van de contextgegevens (of de externe verwerking daarvan) niet altijd tot de gewenste resultaten leidt. Het gesignaleerde probleem met de Google API is hier een goed voorbeeld van. De betreffende API geeft niet altijd een correcte postcode terug. De cijfers van de geretourneerd postcode zijn niet altijd correct, zodat binnen S3 vooralsnog met de postcodecijfers moet worden volstaan.

De toegepaste combinatie van Java, Groovy en de gekozen frameworks heeft -zeker aan de serverzijde- geleid tot een eenvoudiger implementatie en een drastische reductie van de hoeveelheid benodigde code. Daarbij heeft de inzet van Spring Data ook tot gevolg, dat de koppeling van S3Server met MongoDB bijna triviaal is.

## 8.2 Aanbevelingen m.b.t. het S3-systeem

In de voorgaande hoofdstukken zijn al enkele mogelijke uitbreidingen en verbeteringen voor het S3-systeem aangedragen. Voor S3App zijn in paragraaf 5.5 de volgende punten benoemd:

- Toevoegen uitwisselfunctionaliteit voor selfies.
- Herzien van policies door de eigenaar van de selfie mogelijk maken.
- Betere foutafhandeling realiseren.
- Unit- en/of integratietests realiseren.
- Geautomatiseerde tests voor de GUI realiseren.
- Na deinstallatie van de app blijven de gegeneerde keys op S3Server achter.

Ter verbetering/uitbreiding van S3Server worden in paragraaf 5.8 achtereenvolgens genoemd:

- Structuur SymmetricKey aanpassen om sharing van selfies te faciliteren.
- Faciliteren van een breder palet aan locatiegerelateerde policies.
- De communicatie tussen S3App en S3Server via HTTPS laten verlopen.
- Meegestuurd gebruikersregistratie-emailadres in App-Server communicatie is een URL-parameter.
- Er gaan teveel gegevens “over de lijn” tijdens de app-server communicatie.
- Checks op policycompliance zijn te procedureel geimplementeerd.

De belangrijkste twee aanbevelingen worden nader toegelicht, te weten Faciliteren uitwisselfunctionaliteit voor selfies (zie paragraaf 8.2.1) en Toevoegen van een breder palet aan locatiegerelateerde policies (zie paragraaf 8.2.2).

### 8.2.1 Faciliteren uitwisselfunctionaliteit voor selfies

Na het inperken van de scope van dit project (paragraaf 2.1), blijft het faciliteren van uitwisselfunctionaliteit voor selfies de eerstvolgende stap. Hiervoor moet de structuur van de opslag van de keys onder S3Server worden aangepast.

Op dit moment wordt het Android registratie-emailadres van de S3App-gebruiker gebruikt om de gebruiker te identificeren. Dit emailadres is ook opgenomen in het veld ‘owner’ van geregistreerde SymmetricKey’s.
Als men aan SymmeticKey een extra meervoudig veld toevoegt, volgens:

```
List<String> accessableBy
```

dan heeft men een eerste stap gezet richting het “share-baar” maken van de selfies.
Het veld *accessableBy* bestaat dan uit een optionele lijst van registratie-emailadressen van andere S3App gebruikers, aan wie toegang tot de opgeslagen key verleend mag worden. Bij de checks die de server-programmatuur nu o.b.v. het veld ‘owner’ uitvoert, kan men vervolgens ook de waarden uit *accessableBy* betrekken.

### 8.2.2 Toevoegen van een breder palet aan locatiegerelateerde policies

Slechts één locatiegerelateerde policy is wat mager voor een systeem als S3. Men zou de hoeveelheid ondersteunde locatiepolicies als volgt uit kunnen breiden:

Wijzig in SymmetricKey het enkelvoudige attribuut locationPolicy door een lijst van policies, dus vervang

```
S3LocationPolicy locationPolicy
```

door

```
List<S3LocationPolicy> locationPolicies
```

S3LocationPolicy kan dan een abstracte klasse worden, waarvan men diverse gespecialiseerde locatiepolicyklassen af kan leiden.
Deze structuur zegt echter nog niets over de onderlinge samenhang van deze locationpolicies. De meest eenvoudige uitbreiding is om te eisen dat de ontvangen locatie aan alle bij betreffende key geregistreerde policies moet voldoen. In dat geval is er sprake van een logische AND-operatie op alle betrokken locationpolicies. Of zo’n uitbreiding voldoende werkbaar is, zal afhangen aan de eisen die men aan de locationpolicies zou willen stellen. Zo is het niet zinvol te eisen dat een gebruiker zich tegelijkertijd in twee verschillende postcodegebieden moet bevinden om een selfie te kunnen bekijken.

Men kan zich dan de volgende vragen stellen:

1. Wat zijn geschikte en voor gebruikers werkbare combinaties van locationpolicies?
2. Hier is waarschijnlijk een rol weggelegd voor een uitgebreidere vorm van context reasoning. Zoja, wat is dan een passende vorm voor het te realiseren context reasoning proces?

Bovenstaande vragen worden nog belangrijker als men ook zou besluiten tot uitbreiding van de tijdgebaseerde policies.

## 8.3 Praktische aanbevelingen

Bij de realisatie van het S3-systeem is gebruik gemaakt van de IDE's Android Studio (o.b.v. IntelliJ IDEA) en Spring Tool Suite (o.b.v. Eclipse for Java EE).
Android Studio wordt weliswaar gratis door Google aangeboden, maar het onderliggende IntelliJ IDEA is geen open source pakket.Praktisch betekent dit, dat een licentie zou moeten worden gekocht voor de volledige versie van IntelliJ IDEA die ook ondersteuning biedt voor de frameworks die binnen S3Server worden toegepast. Deze licentie is niet goedkoop en is om die reden dan ook niet aangeschaft. Dit leidt echter wel tot de volgende problemen.

*Twee verschillende ontwikkelomgevingen en twee verschillende buildtools.*
Bovengenoemde twee IDE's hebben ieder een eigen voorkeur voor een standaard buildtool. S3App is gebouwd in Android Studio en wordt gebuild met Gradle, terwijl S3Server in Spring Tool Suite is gebouwd en m.b.v. Maven wordt gebuild.
Dit is ongewenste situatie, zeker als een team studenten verder aan het S3-systeem gaat bouwen. Het verdient de aanbeveling om over te stappen naar één en dezelfde buildtool voor zowel S3App als S3Server. Zie ook Bijlage 6 voor meer informatie.

*S3App en S3Server kennen gemeenschappelijke code die niet is afgesplitst.*
Diverse definities worden zowel binnen S3App als binnen S3Server gebruikt. Vanwege de twee verschillende buildtools is daar geen actie op ondernomen.
Als men in staat is beide subprojecten met dezelfde buildtool op te bouwen, kan men de betreffende code in een apart subproject onderbrengen. Vervolgens kan men dit subproject vanuit S3App en S3Server op dezelfde manier benaderen en op identieke wijze tijdens het builden verwerken.

# 9 Begrippen en afkortingen

**A**

Aggregaat
: Een samenstelling van objecten dat als één geheel wordt behandeld.

API
: Application Programming Interface. Technische faciliteit waardoor programmeurs in staat worden gesteld functionaliteit van externe producten (geleverd door derden) te benutten. Doorgaans gebeurd dit in de vorm van een framework of d.m.v. webservices.

Assymetrische encryptie
: Zie ‘Public key encryption’.

**C**

Cell-ID
: Locatiebepaling via de identificatiegegevens van de GSM-mast(en), waarmee de smartphone in verbinding staat.

Collection (MongoDB)
: Opslagstructuur waarbinnen gelijksoortige/-vormige documenten worden opgeslagen. Vergelijkbaar met tabellen in RDBMS’sen, waarin records worden opgeslagen.

**D**

Dependency Injection (Inversion of Control)
: Een design pattern waarbij een object niet zelf afhankelijkheden instantiëert. Het object krijgt deze afhankelijkheden tijdens het instantiëren van buitenaf aangeleverd.

Document Database
: Type database waarin gegevens niet als records, maar als documents worden opgeslagen. Deze documents hebben doorgaans een JSON structuur.

Drools
: Een open source Business Rules Management System, gesponsord door Red Hat.

DSL
: Domain Specific Language. Een programmeertaal die speciaal ontworpen is om problemen binnen een bepaald domein eenvoudiger op te kunnen lossen.

**E**

Entity
: Een design pattern: Een object waarbij de eigen identiteit een belangrijke rol heeft, ongeacht de staat van de attributen binnen dit object.

**F**

Front Controller
: Een design pattern: Een controller die alle requests voor een website/webapplicatie verwerkt.

**G**

Groovy
: Een door Python geïnspireerde dynamische uitbreiding van de programmeertaal Java. Groovy biedt tevens optionele static typing.

**H**

HTTP-RPC
: HTTP Remote Procedure Call. Een protocol voor webservices.

**J**

JSON

: JavaScript Object Notation. Een eenvoudig formaat voor uitwisselen van berichten.

**M**

Machine learning

: De tak van informatica waar men computers het vermogen geeft om te acteren/leren, zonder dat ze daartoe expliciet geprogrammeerd zijn.

MongoDB

: Een Document Database, valt onder de categorie NoSQL databases.

**N**

NoSQL

: Categorie databases, ontstaan vanaf 2005, waarin het relationele model wordt verlaten.

**O**

OTAP

: Werkwijze tijdens systeemontwikkeling, waarbij de componenten stapsgewijs via het traject **O**ntwikkelomgeving → **T**estomgeving → (Gebruiker)**A**cceptatieomgeving → **P**roductieomgeving in gebruik worden genomen.

**P**

Public key encryption

: Encryptiemethode waarbij verschillende keys voor encryptie en decryptie nodig zijn

**R**

Repository

: Abstractie/pattern waarbij de set gegevens uit een databasetabel of uit een document collection (Document Database) als een native collectie wordt beschouwd.

REST

: Representational State Transfer. De software-architectuur achter het World Wide Web.

RESTful / RESTful API

: Een RESTful API kan alle of enkele CRUD operaties bieden. Deze operaties kunnen vanuit de client via HTTP-requests (GET, POST, PUT en DELETE) aangeroepen worden.

RMI

: Remote Method Invocation. Java platform specifieke Remote Procedure Call functionaliteit.

**S**

Service Layer

: Een design pattern: Definieert de grens van de applicatie via een laag die de beschikbare operaties bevat. Bepaalt ook de response van de applicatie voor iedere operatie.

SOAP

: Simple Object Access Protocol. Een protocol voor webservices.

Spring framework

: Een framework om enterprise grade Java (web)applicaties te bouwen. Het biedt een alternatief en uitbreidingen voor het Java EE platform.

Spring Data

: Uitbreiding op het Spring framework. Heeft als doel een uniforme abstractie te bieden voor benaderen van verschillende typen databases, waaronder RDBMS'sen, Document stores, Key-Value stores en Graph databases. Spring Data biedt developers abstracties o.b.v. het Repository pattern.

Spring MVC

: Uitbreiding op het Spring framework. Biedt zowel 'oldschool' MVC als uitgebreide ondersteuning voor realisatie van RESTful API's. Spring MVC werkt volgens het Front controller pattern.

**T**

Tomcat

: Een op Java gebaseerde webcontainer/webserver, waarin Java webapplicaties kunnen draaien. Tomcat is de reference implementation voor de Java EE specificaties t.b.v. Servlets en JSP's.

**U**

Ubiquitous computing

: Het concept dat het gebruik van computers altijd en overal mogelijk is (zoals bij smartphones).

**V**

Value Objects

: Een design pattern: Definieert een object dat wordt gerechtvaardigd door de ingekapselde data en logica, zonder dat het object een concept van identiteit heeft. Value Object gedragen zich immutable.

**X**

XML-RPC

: XML Remote Procedure Call. Een protocol voor webservices.

# 10 Literatuurlijst

## 10.1Wetenschappelijke publicaties


[Achilleos et al., 2010]
ACHILLEOS, A., YANG, K., AND GEORGALAS, N. 2010. Context modelling and a context-aware framework for pervasive service creation: A model-driven approach. *Pervasive and Mobile Computing 6(2)*: 281–296.

[Ahmed, 2010]
AHMED, A.A.A. 2010. Context-Aware Acces Control in Ubiquitous Computing (CRAAC). Doctoral dissertation. University of Manchester, Manchester UK.

[Bardram, 2005]
BARDRAM, J.E. 2005. The Java Context Awareness Framework (JCAF) – A Service Infrastructure and Programming Framework for Context-Aware Applications. In: H.-W. Gellersen, R. Want and A. Schmidt, eds., *Pervasive Computing*. Springer Berlin Heidelberg, 98–115.

[Bettini et al., 2010]
BETTINI, C., BRDICZKA, O., HENRICKSEN, K., ET AL. 2010. A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing 6(2)*: 161–180.

[Dey, 2001]
DEY, A.K. 2001. Understanding and Using Context. *Personal and Ubiquitous Computing 5(1)*: 4–7.

[Fielding, 2000]
Fielding, R.T. 2000. Architectural Styles and the Design of Network-based Software Architectures. Doctoral Dissertation. University of California, Irvine.

[Guiness, 2015]
GUINNESS, R.E. 2015. Beyond Where to How: A Machine Learning Approach for Sensing Mobility Contexts Using Smartphone Sensors. *Sensors 15*, 5, 9962–9985.

[Göker en Myrhaug, 2002]
GÖKER, A. AND MYRHAUG, H.I. User context and personalisation. In Proc. *1st Workshop on Case Based Reasoning and Personalisation*, pp. 1–7, 2002.

[Hoyos et al., 2013]
HOYOS, J.R., GARCÍA-MOLINA, J., AND BOTÍA, J.A. 2013. A domain-specific language for context modeling in context-aware systems. *Journal of Systems and Software 86(11)*: 2890–2905.

[Joinson et al, 2010]
JOINSON, A.N., REIPS, U.-D., BUCHANAN, T., AND SCHOFIELD, C.B.P. 2010. Privacy, Trust, and Self-Disclosure Online. *Human–Computer Interaction 25*, 1, 1–24.

[Li et al., 2011]
LI, L., XING, G., SUN, L., HUANGFU, W., ZHOU, R., AND ZHU, H. 2011. Exploiting FM Radio Data System for Adaptive Clock Calibration in Sensor Networks. *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, ACM, 169–182.

[Lu et al., 2012]
LU, T., LIU, X., LIU, X., AND ZHANG, S. 2012. An Interval-Based Context Reasoning Approach. *International Journal of Advanced Research in Artificial Intelligence 1*, 8.

[Neovius en Sere, 2009]
NEOVIUS, M. AND SERE, K. 2009. Formal Modular Modelling of Context-Awareness. In: F.S. de Boer, M.M. Bonsangue and E. Madelaine, eds., *Formal Methods for Components and Objects*. Springer Berlin Heidelberg, 102–118.

[Nissenbaum, 2010]
NISSENBAUM, H. 2010. *Privacy in Context – Technology, Policy, and the Integrity of Social Life.* Stanford University Press. Stanford, California.

[Park et al., 2013]
PARK, J., LEE, H.-C., AND LEE, M.-J. 2013. JCOOLS: A toolkit for generating context-aware applications with JCAF and DROOLS. *Journal of Systems Architecture 59(9)*: 759–766.

[Paulik et al., 2010]
Paulik, T., Foldes, A.M., and Gulyas, G.G. 2010. BlogCrypt: Private Content Publishing on the Web. *Proceedings of the 2010 Fourth International Conference on Emerging Security Information, Systems and Technologies*, IEEE, 123–128.

[Ra et al., 2013]
Ra, Govindan, and Ortega. 2013. P3: Toward Privacy-Preserving Photo Sharing. *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI '13)* , USENIX, 515-528.

[Reed et al., 2016]
Reed, P.J., Spiro, E.S., and Butts, C.T. 2016. Thumbs up for privacy?: Differences in online self-disclosure behavior across national cultures. *Social Science Research 59*, 155–170.

[Riahi en Moussa, 2015]
Riahi, I. and Moussa, F. 2015. A formal approach for modeling context-aware Human–Computer System. *Computers & Electrical Engineering 44*, 241–261.

[Sowa, 2000]
Sowa, J.F. 2000. *Knowledge Representation: Logical, Philosophical and Computational Foundations*, Brooks/Cole, Pacific Grove.

[Stein en Gonzalez, 2014]
Stein, G. and Gonzalez, A.J. 2014. Learning in context: enhancing machine learning with context-based reasoning. *Applied Intelligence 41*, 3, 709–724.

[Stuurman et al., 2014]
Stuurman, S., Gastel, B.E., and Passier, H.J.M. 2014. The Design of Mobile Apps: what and how to teach?. *Proceedings of the Computer Science Education Research Conference CSERC '14*, ACM, 93–100.

[Taddei en Contena, 2013]
Taddei, S. and Contena, B. 2013. Privacy, trust and control: Which relationships with online self-disclosure? *Computers in Human Behavior 29*, 3, 821–826.

[Zhang et al., 2011]
Zhang, D., Huang, H., Lai, C.-F., Liang, X., Zou, Q., and Guo, M. 2011. Survey on context-awareness in ubiquitous media. *Multimedia Tools and Applications 67(1)*: 179–211.

## 10.2Vakliteratuur

[Evans, 2004]
Evans, E. 2004. *Domain-Driven Design: Tackling Complexity in the Heart of Software*, Addison-Wesley, Upper Saddle River.

[Fowler et al., 2003]
Fowler, M., Rice, D., Foemmel, M., Hieatt, E., Mee, R. and Stafford, R. 2003. *Patterns of Entrerprise Application Architecture*, Addison-Wesley, Boston.

[Larman, 2012]
Larman, C. 2012. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, third edition (third ed.),* Pearson Education, Upper Saddle River.

[Pfleeger et al., 2015]
Pfleeger, C.P., Pfleeger, S.L.. AND Margulies, J. 2015. *Security in Computing – Fifth Edition*, Prentice Hall, Upper Saddle River.

[Sadalage en Fowler, 2013]
Sadalage, P.J. and Fowler, M. 2013. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*, Addison-Wesley, Upper Saddle River.

## 10.3Webreferenties


[webref: Android Developers, 2016-1]
Android Developers, 2016-1. *Location | Android Developers*.
http://developer.android.com/reference/android/location/Location.html. Geraadpleegd 24 januari 2016.

[webref: Android Developers, 2016-2]
Android Developers, 2016-2. *Location Strategies | Android Developers*.
http://developer.android.com/guide/topics/location/strategies.html. Geraadpleegd 24 januari 2016.

[webref: Android Developers, 2016-3]
Android Developers, 2016-3. *Sensor | Android Developers*.
http://developer.android.com/reference/android/hardware/Sensor.html. Geraadpleegd 2 februari 2016.

[webref: Android Developers, 2016-4]
Android Developers, 2016-4. *SensorEvent | Android Developers*.
http://developer.android.com/reference/android/hardware/SensorEvent.html. Geraadpleegd 2 februari 2016.

[webref: Android Developers, 2016-5]
Android Developers, 2016-5. *Storage Options | Android Developers.*
http://developer.android.com/guide/topics/data/data-storage.html#AccessingExtFiles. Geraadpleegd 11 augustus 2016.

[webref: db-engines.com, 2016]
DB-Engines, 2016. *DB-Engines Ranking - popularity ranking of database management systems.*
http://db-engines.com/en/ranking. Geraadpleegd 10 augustus 2016.

[webref: Developer.xamarin.com, 2016]
Developer.xamarin.com, 2016. *Understanding Android API Levels.*
https://developer.xamarin.com/guides/android/application_fundamentals/understanding_android_api_levels/
Geraadpleegd 21 februari 2016.

[webref: DeviceSpecifications, 2016]
DeviceSpecifications, 2016. *Huawei Y625 – Specifications*.
http://www.devicespecifications.com/en/model/8f153335. Geraadpleegd 2 februari 2016.

[webref: GitHub, 2016-1]
GitHub, 2016-1. *googlemaps/google-maps-services-java*.
http://github.com/googlemaps/google-maps-services-java. Geraadpleegd 1 februari 2016.

[webref: GitHub, 2016-2]
GitHub, 2016-2. *googlemaps/google-maps-services-python*.
http://github.com/googlemaps/google-maps-services-python. Geraadpleegd 1 februari 2016.

[webref: Google Developers, 2016-1]
Google Developers, 2016-1. *Google Maps Web Service APIs | Google Developers*.
http://developers.google.com/maps/web-services. Geraadpleegd 1 februari 2016.

[webref: Google Developers, 2016-2]
Google Developers, 2016-2. *Google Maps Distance Matrix API | Google Developers*.
https://developers.google.com/maps/documentation/distance-matrix. Geraadpleegd 1 februari 2016.

[webref: Google Developers, 2016-3]
Google Developers, 2016-3. *Pricing and Plans | Google Maps APIs | Google Developers*.
https://developers.google.com/maps/pricing-and-plans/. Geraadpleegd 2 februari 2016.

[webref: Javamex, 2012]
Javamex, 2012. *Comparison of encryption ciphers in Java.*
http://www.javamex.com/tutorials/cryptography/ciphers.shtml. Geraadpleegd 3 juli 2016.

[webref: MLContext, 2015]
MLContext, 2015. *MLContext Home Page*.
http://www.modelum.es/MLContext/. Geraadpleegd 14 februari 2016.

[webref: Oracle, 2016]
Oracle, 2016. *Standard Algorithm Name Documentation.*
https://docs.oracle.com/javase/7/docs/technotes/guides/security/StandardNames.html#Cipher.
Geraadpleegd 29 juni 2016.

[webref: Samsung newsroom, 2014]
Samsung newsroom, 2014. *10 Sensors of Galaxy S5: Heart Rate, Finger Scanner and more.*
https://news.samsung.com/global/10-sensors-of-galaxy-s5-heart-rate-finger-scanner-and-more
Geraadpleegd 6 februari 2016.

[webref: Sourceforge.net, 2016]
Sourceforge.net, 2016. *Java Context-Aware Framework*.
http://sourceforge.net/p/jcaf/code/HEAD/tree/. Geraadpleegd 7 februari 2016.

[webref: Szántó, 2010]
Szántó, 2010. *Extending the Java Context Awareness Framework for Android.*
http://sourceforge.net/projects/jcaf/files/documentation/RESTful_jcaf_for_Android.pdf/download.
Geraadpleegd 6 februari 2016.

[Webref: The Guardian, 2013-1]
The Guardian. *New NSA leaks show how US is bugging its European allies.*
https://www.theguardian.com/world/2013/jun/30/nsa-leaks-us-bugging-european-allies
Geraadpleegd 26 augustus 2016.

[Webref: The Guardian, 2013-2]
The Guardian. *GCHQ: EU surveillance hearing is told of huge cyber-attack on Belgian firm.*
https://www.theguardian.com/uk-news/2013/oct/03/gchq-eu-surveillance-cyber-attack-belgian
Geraadpleegd 26 augustus 2016.

[webref: Webservices.nl, 2016-1]
Webservices.nl, 2016-1. *GeoLocation - Webservices.nl.*
https://webview.webservices.nl/documentation/files/service_geolocation-php.html. Geraadpleegd 29 januari 2016.

[webref: Webservices.nl, 2016-2]
Webservices.nl, 2016-2. *Implementation - Webservices.nl.*
https://webview.webservices.nl/documentation/files/implementation-txt.html#Implementation.Integrating_web_services.
Geraadpleegd 1 februari 2016.

# Bijlage 1 – Introductie Groovy binnen het S3-project

Groovy is een dynamische uitbreiding op de Java programmeertaal. Met Groovy is het mogelijk om geavanceerde concepten uit talen als Python en JavaScript op het Java platform te gebruiken. Zo bood Groovy al jaren voor de release van Java 8 de mogelijkheid om met closures te werken. Sinds versie 2.4 is Groovy ook beschikbaar voor het Android platform. Groovy 2.4 wordt zowel binnen S3App als S3Server gebruikt.

Het inzetten van Groovy binnen een project kent voor- en nadelen. Deze worden hieronder besproken.

## Voordelen

*Java sourcecode is in zeer hoge mate compatible met Groovy.*
Dit geldt zeker t/m Java 7. Het wijzigen van de bestandsextensie *.java* naar *.groovy* is meestal voldoende om een valide Groovy sourcefile te verkrijgen. Inzet van Groovy leidt dus niet meteen tot sourcecode met een totaal andere syntax, zoals dat bijvoorbeeld met een taal als Scala het geval is.

*Groovy kent betere 'sensible defaults' dan Java.*
Bij het ontwerp van een programmeertaal worden soms keuzes gemaakt, die later ongelukkig uitvallen. Zo leidt bijvoorbeeld het gebruik van Java met standaard Java Beans i.c.m. object-relational mapping vaak tot enorme hoeveelheden getters en setters, die geen enkel praktisch nut hebben. De ontwerpers van Groovy hebben voor andere 'sensible defaults' gekozen, waardoor de hoeveelheid 'loodgieterscode' doorgaans enorm afneemt.

*Meer efficiënte en expressievere code.*
Java heeft niet altijd een goede naam m.b.t. efficiënte sourcecode, zie onderstaande voorbeelden.

Voorbeeld 1
Vergelijk onderstaande Pythonstyle boolean expressie in Groovy (uit de PoliciesPolicy klasse):

```
result[0].formattedAddress[0..3] != key.locationPolicy.postalCode[0..3]
```

met het standaard Java equivalent:

```
!result[0].getFormattedAddress().substring(0, 4)
.equals(key.getLocationPolicy().getPostalCode().substring(0, 4))
```

Voorbeeld 2
Vergelijk het gebruik van JavaScriptstyle truthy en falsey in Groovy:

```
if (stringVariabele) { }
```

met het gebruikelijke Java idioom:

```
if (stringVariabele != null && !stringVariabele.equals("")) { }
```

*De GroovyConsole kan Java en Groovy code uitvoeren.*
Java kent zelf geen console waarin men stukjes sourcecode uit kan proberen. Groovy biedt hiertoe de GroovyConsole waarmee men op eenvoudige wijze Java- en Groovycode uit kan voeren.

*Groovy wordt veel toegepast als scripting taal binnen op Java gebaseerde software.*
Voorbeelden zijn o.a. SoapUI (testtool voor webservices en API's) en Jenkins (tool voor Continuous Integration en Continuous Delivery).

## Nadelen

*Groovy valt niet onder een officiële Java standaard.*
Praktisch gezien betekent dit, dat het Groovy-team niet direct de optimale integratie van Groovy met een nieuwe Javaversie op orde zal hebben. Groovy is wel een populair open source project, dat onder de paraplu van de Apache Software Foundation valt.

*Dynamisch karakter van Groovy leidt tot lagere performance.*
Dit is een algemeen probleem met dynamische programmeertalen zoals Python en Ruby. Sinds versie 2.0 biedt Groovy de annotatie *@CompileStatic*. Hiermee wordt een stukje van de geboden dynamiek ingeleverd, maar het performance-gat met standaard Javacode wordt hiermee nagenoeg volledig gedicht.

## Het gebruik van Groovy binnen S3

De voordelen blijken dusdanig groot, dat binnen S3App volledig met Groovy wordt gewerkt, terwijl binnen S3Server een groot deel van de code met Groovy is gerealiseerd. Binnen het S3-project wordt Groovy op de volgende manieren toegepast:

*Groovy zodanig toegepassen dat de code voor personen met Javakennis eenvoudig te volgen is.*
Men kan het gebruik van Groovy zover doorvoeren, dat de code moeilijker te volgen wordt voor Java developers. Vanwege de te verwachten Javakennis van een volgende groep studenten, volgt de Groovycode doorgaans de bekende Javasyntax.

*Groovyklassen worden altijd voorzien van @CompileStatic.*
Dit heeft te maken met bovengenoemde mogelijke performanceproblemen. Bij gebruik onder Android is deze annotatie zelfs verplicht. Binnen S3Server zijn slechts twee uitzonderingen:

1. de AppConfig-klasse; dit is configuratiecode, geen applicatiecode;
2. de PoliciesPolicy-klasse; het toepassen van @CompileStatic levert hier problemen op met het benaderen van de Google Geolocation API.

*Statements binnen Groovyklassen eindigen niet op ";".*
Groovy vereist geen punt-komma's. De punt-komma's worden weggelaten, zodat het meteen duidelijk is dat betreffende klasse een Groovyklasse is.

*Weglaten van de Java 'public' access modifier.*
Dit is een 'sensible default' binnen Groovy. Binnen Groovy wordt een variabele/attribuut, method of klasse die geen access modifier heeft, als 'public' beschouwd.
Binnen in Groovy geschreven Java Beans gaat Groovy nog een stapje verder. Hier worden bij deze attributen (at compile time) 'private' attributen met 'public' getters en setters gegenereerd.
Binnen het S3-project wordt hier volop gebruik van gemaakt.

*Instantiëren van Java Beans m.b.v. constructor met benoemde optionele parameters.*
Binnen Groovycode kan men Java Beans (zowel in Java als in Groovy geschreven!) instantiëren m.b.v. een constructor met benoemde optionele parameters. Zo zou men een Java Bean van een type Persoon met de attributen naam en beroep kunnen instantiëren volgens:

```
Persoon persoon = new Persoon(naam: "Fozzy Bear", beroep: "Komiek")
```

*Inzet 'slimme' methods voor omgang met Java Collections.*
Onderstaand voorbeeld is gewijzigd overgenomen uit de MainActivity van S3App. Het laat ook zien hoe men een closure in Groovy toe kan passen.

```
private File thumbnailsPath
private List<Uri> thumbnails = []

thumbnailsPath.listFiles().sort().reverse(true).each {
    File f -> thumbnails.add(Uri.fromFile(f))
}
```

*Groovy Strings voor dynamisch samen te stellen stringwaarden.*
Ook hier een voorbeeld:

```
String woord1 = "hallo"
String halloWereld = "${woord1} wereld!"
```

Dankzij de placeholder ${woord1} evalueert de stringvariabele *halloWereld* naar *"hallo wereld!"*.

*Bij een niet-void method is het resultaat van de laatste expressie bruikbaar als returnwaarde.*
Gebruik van het `return` statement is dus niet altijd verplicht.

## Documentatie

De officiële Groovy documentatie:
http://docs.groovy-lang.org/docs/groovy-2.4.3/html/documentation/

Een recente en duidelijk tutorial op vogella.com:
http://www.vogella.com/tutorials/Groovy/article.html.

# Bijlage 2 – Spring, Spring MVC en Spring Data binnen S3Server

Deze bijlage gaat in op het gebruik van het Spring framework plus de uitbreidingen Spring MVC en Spring Data binnen S3Server. De benodigde Spring configuratie komt aan het einde van deze bijlage aan de orde.

## Het Spring framework

In 2003 is het Spring framework ontstaan als een eenvoudiger open source alternatief t.o.v. de Java Enterprise Edition (Java EE) en de daarop gebaseerde applicationservers. Per 2016 is de Spring community nog steeds een bron van innovatie binnen de Javawereld, waarbij men vaak jaren voorloopt op zowel Java EE als de standaardisatie via JSR's.

Binnen S3Server wordt Spring toegepast om de applicatielagen onderling te verbinden. Daartoe wordt gebruik gemaakt van de Spring applicatiecontext. Deze context vormt een applicatiebrede 'super'scope. De componten binnen deze context kunnen op eenvoudige wijze gebruik van elkaar maken. Hiertoe wordt gebruik gemaakt van Dependency Injection o.b.v. de @Autowired annotatie.

Tijdens deployment van S3Server op Tomcat loopt Spring alle @Autowired annotaties binnen de code na en kijkt of de gewenste componenten binnen de context beschikbaar zijn. Mocht dit falen, dan zal S3Server niet starten.

## Spring MVC

Spring MVC is ooit ontstaan als alternatief voor o.a. het Struts framework. T.o.v. Struts kent Spring MVC een veel modernere insteek en wordt het nog steeds onderhouden en verbeterd. Net als Struts is Spring MVC een implementatie van het Front Controller pattern. Dat heeft tot gevolg dat in de *web.xml* van S3Server maar één servlet wordt gedefiniëerd: de Spring DispatcherServlet (als Front Controller).

De Spring MVC webcontext leunt op de Servletcontext. Deze valt echter niet onder de bovengenoemde applicatiecontext en heeft daarom een apart stukje configuratie nodig (zie aan het einde van deze bijlage).

S3Server maakt geen gebruik van MVC in de klassieke betekenis. Er wordt wel gebruik gemaakt van de RESTful functionaliteit die Spring MVC biedt. Onderstaande afbeelding toont hoe eenvoudig de heartbeatfunctionaliteit van S3Server RESTful geïmplementeerd is.

```
@RestController
@RequestMapping("/ping")
public class PingController {

    /**
     * Levert ping- cq. heartbeatfunctionaliteit.
     */
    @RequestMapping(method = GET)
    public String ping() {
        return "I'm still alive!";
    }

}
```

*Afbeelding 19: Body van de S3Server PingController klasse.*

Een HTTP GET-request naar S3Server (via *http://<servernaam:poortnr>/S3Server/ping*) levert de tekst "I'm still alive!" op, mits de server bereikbaar is. Zo kan men met een webbrowser eenvoudig nagaan of de S3-Server volledig in de lucht is.

## Spring Data (voor MongoDB)

Spring Data biedt een uniforme abstractie voor het benaderen van verschillende soorten databases, waaronder RDBMS'sen, Document databases, Key-Value stores en Graph databases. Spring Data biedt deze abstractie o.b.v. het Repository pattern.

Omdat binnen S3Server met MongoDB wordt gewerkt, wordt van de Spring Data variant voor MongoDB gebruik gemaakt. De configuratie van Spring Data valt onder de Spring applicatiecontext, zodat hiervoor geen aparte configuratieklasse of -bestand nodig is.

Van Spring Data worden binnen S3Server drie aspecten gebruikt:

1. Mapping van Entities, uit de Domain layer, op MongoDB documents.
2. Repositories; een Repository abstraheert een MongoDB collectie naar een Java Collection.
3. Automatische validatie m.b.v. Bean Validation (zie Bijlage 3) bij opslag van gegevens in de database.

ad 1

Onderstaande afbeelding geeft weer hoe de Spring Data annotaties binnen de SymmetricKey (Groovy) entity zijn toegepast. Het toont alleen het begin van de implementatie van deze klasse.

```
@Document(collection="SymmetricKey")
@CompileStatic
class SymmetricKey {

    @Id
    String id
```

*Afbeelding 20: Aanhef van de  ymmetricKey entity.*

De annotatie @Document geeft aan, dat deze bean gemapped moet worden op de collectie SymmetricKey binnen MongoDB. @Id geeft aan, dat de string id de unieke identificatie voor instanties van SymmetricKey vormt.

ad 2

Zoals reeds in paragraaf 5.7.5. is aangegeven, is onderstaande definitie van de SymmetricKeyRepository in de Repository laag voldoende voor de huidige functionaliteit van S3Server.

```
public interface SymmetricKeyRepository extends MongoRepository<SymmetricKey, String> {

}
```

*Afbeelding 21:  ymmetricKeyRepository Interface in de Repository layer.*

Spring Data biedt echter veel meer mogelijkheden, waaronder het declareren van 'finders'. Stel dat men later op alle keys van een bepaalde owner wil kunnen zoeken. Men kan dan volstaan met uitbreiding van de interface met de volgende methoddeclaratie:

```
List<SymmetricKey> findByOwner(String owner);
```

Ook hier wordt weer, at-runtime, automatisch de juiste code gegenereerd. Met deze techniek wordt geabstraheerd van het onderliggende type database. Dit is een basisconcept binnen Spring Data.

## Spring configuratie voor S3Server

Spring was vroeger berucht om zijn uitgebreide XML-configuratiebestanden. Sinds Spring 3 (2009) biedt het framework de mogelijkheid om deze configuratie m.b.v. speciale JavaConfig klassen te definiëren.
S3Server maakt van deze mogelijkheid gebruik d.m.v. de AppConfig en WebConfig klassen die in de config-package zijn ondergebracht.
De connectie tussen de applicatie- en de webcontext wordt in het *web.xml*-bestand vastgelegd.

Binnen de AppConfig en WebConfig klassen wordt zoveel mogelijk gebruik gemaakt van autoscanning van Javapackages en van defaultwaarden. Daardoor is WebConfig klasse nagenoeg leeg, zie onderstaande afbeelding.

```
@Configuration
@ComponentScan(basePackages="nl.ou.s3server.controller")
public class WebConfig extends WebMvcConfigurationSupport {

}
```

*Afbeelding 22: Body van de S3Server WebConfig klasse.*

Door de annotaties @Configuration en @ComponentScan weet Spring MVC dat dit een configuratieklasse is en dat de componenten in de controllerpackage staan. Samen met het feit dat deze klasse erft van WebMvc-ConfigurationSupport, is dit voldoende om de webapplicatiecontext (t.b.v. de REST API) van S3Server te configureren.

## Officiële documentatie

Spring framework en Spring MVC 4.3.2:
http://docs.spring.io/spring/docs/4.3.2.RELEASE/spring-framework-reference/htmlsingle/

Spring Data (voor MongoDB) 1.9.2:
http://docs.spring.io/spring-data/data-mongo/docs/1.9.2.RELEASE/reference/html/

# Bijlage 3 – Overige Java frameworks

## Hibernate Validator

Hibernate Validator is de officiële referentie implementatie van de Java specificatie voor Bean Validation (JSR303 en JSR349). Met Bean Validation is het mogelijk om binnen een Java Bean declaratieve validatie-criterie op attributen en bijbehorende foutmeldingen te definiëren. Zie onderstaande Groovycode van de klasse SymmetricKey als voorbeeld:

```
@Document(collection="SymmetricKey")
@CompileStatic
class SymmetricKey {

    @Id
    String id

    @NotEmpty(message="Veld owner mag niet leeg zijn.")
    @Email(message="Veld owner moet een geldig emailadres bevatten.")
    String owner

    /** Optionele Policy t.b.v. tijdstip einde geldigheid. */
    @Valid
    S3ExpirationPolicy expirationPolicy

    /** Optionele Policy m.b.t. locatie van gebruiker/device. */
    @Valid
    S3LocationPolicy locationPolicy

    /** De daadwerkelijke symmetrische key. */
    @NotEmpty(message="Veld key mag niet leeg zijn.")
    String key

}
```

*Afbeelding 23: Implementatie SymmetricKey met Bean Validation annotaties.*

In voorgaand codefragment worden de volgende Bean Validation annotaties gebruikt: @NotEmpty, @Email en @Valid.
Het doel van de annotaties @NotEmpty en @Email met bijbehorende messages is direct duidelijk.
De @Valid annotatie op beide policies zorgt ervoor, dat tijdens de validatie van een SymmetricKey instantie ook (eventuele) validatiecriteria binnen de gekoppelde policyobjecten mee worden geëvalueerd.

Voor documentatie zie:
http://docs.jboss.org/hibernate/validator/5.2/reference/en-US/html_single/

Directe link naar documentatie van de mogelijke validatie-annotaties:
http://docs.jboss.org/hibernate/validator/5.2/reference/en-US/html_single/#section-builtin-constraints

## Jackson

Jackson wordt ingezet t.b.v. het snel serialiseren van Java objecten van en naar JSON formaat. Zo wordt in de SelfieDialog klasse (S3App) onderstaand statement gebruikt om de JSON-data binnen de ontvangen serverresponse snel om te zetten naar een nieuwe instantie van SymmetricKeyDto:

```
def symKeyDto = new ObjectMapper().readValue(getResponse.body().bytes(), SymmetricKeyDto)
```

Voor meer informatie, zie:
https://github.com/FasterXML/jackson-docs

En een misschien wel betere online tutorial:
http://tutorials.jenkov.com/java-json/jackson-objectmapper.html

## OkHttp

OkHttp framework wordt gebruikt om eenvoudiger met HTTP-requests en responses te kunnen werken. Onderstaand codefragment (gewijzigd overgenomen uit S3App's MainAcivity) illustreert dit.

```
Request postRequest = new Request.Builder()
        .url(Constants.SERVER_KEY_URL)
        .post(RequestBody.create(Constants.JSON, jsonMessage))
        .build()
Response postResponse = client.newCall(postRequest).execute()
```

Met de standaard HTTP-voorzieningen binnen Java kost het duidelijk meer moeite om dit soort functionaliteit te bouwen.

De website voor OkHttp:
http://square.github.io/okhttp/

Een duidelijke tutorial op vogella.com:
http://www.vogella.com/tutorials/JavaLibrary-OkHttp/article.html

## Google Maps Services

De Google Maps Services library wordt gebruikt om vanuit Java eenvoudig de Google Maps Services te kunnen benaderen. De developer hoeft daardoor niet zelf de benodigde HTTP-requests samen te stellen. Onderstaand codefragment uit de PoliciesPolicy klasse geeft de aanroep naar de Google API weer.

```
GeoApiContext geoApiContext = new GeoApiContext().setApiKey(googleApiKey)
GeocodingResult[] result = null
LatLng latLng = new LatLng(location.latitude, location.longitude)

// Probeer de postcode van meegestuurde locatie te bepalen.
try {
    result = GeocodingApi.reverseGeocode(geoApiContext, latLng).resultType(AddressType.POSTAL_CODE)
            .language("nl").await()
} catch (Exception e) {
```

*Afbeelding 24: Groovy code t.b.v. benaderen Google geolocation API.*

Om deze Google API te kunnen benaderen, moet men een API-key aanvragen (hierboven als *googleApiKey* weergegeven). Aanvragen van deze key kan via:
https://developers.google.com/maps/documentation/geocoding/start#get-a-key

Officiële documentatie:
https://developers.google.com/maps/documentation/geocoding/start en
https://github.com/googlemaps/google-maps-services-java

# Bijlage 4 – Toegepaste Groovy frameworks

## SwissKnife

SwissKnife is een Groovy framework voor gebruik onder Android. Het heeft als doel het ontwikkelen onder Android te vereenvoudigen. SwissKnife biedt daarvoor veel annotaties die de hoeveelheid 'loodgieterscode', die onder Android vereist is, flink af doet nemen.
De geboden annotaties hebben vooral betrekking op het koppelen van (in XML gedefinieerde) views aan Java/Groovy variabelen. Ook biedt SwissKnife veel annotaties voor de Android lifecycle- en eventhandling-methods.

Documentatie:
http://arasthel.github.io/SwissKnife/

## Spock framework

Spock is een framework voor (unit)testing. Het is een uitbreiding op JUnit en werkt met Groovy. Omdat Spock een uitbreiding van JUnit is, kunnen Spock tests eenvoudig als JUnit-testgevallen worden uitgevoerd. Het Spock framework heeft een 'batteries included' benadering, het biedt standaard voorzieningen voor mocking en stubbing. In de praktijk heeft dat als voordeel dat men niet een extra mocking framework, zoals EasyMock of Mockito, hoeft te leren.

Mocht een volgende groep studenten niet met Spock willen of kunnen werken, dan kan men verder gaan met het toevoegen van unittestklassen o.b.v. JUnit. De bestaande Spock testklassen zullen dan gewoon blijven werken.

Documentatie:
http://spockframework.org/spock/docs/1.0/index.html

Enkele tutorials:
http://farenda.com/spock-framework-tutorial/
http://zeroturnaround.com/rebellabs/using-spock-to-test-groovy-and-java-applications

# Bijlage 5 – MongoDB t.b.v. S3ServerDb

## Reflectie op gebruik van MongoDB binnen het S3-project

Tijdens de realisatie van S3Server bleek de inzet van MongoDB inderdaad eenvoudiger dan de inzet van een RDBMS. Dat is niet alleen te danken aan de eerder beschreven punten van Sadalage en Fowler [2013]. Het gebruik van een effectief en eenvoudig toe te passen Java framework (Spring Data MongoDB), draagt hier nadrukkelijk aan bij.

De inzet van MongoDB kent echter één praktisch nadeel. Als men een database nodig heeft, grijpt men al snel naar de bekende relationele databases. Voornamelijk omdat men dat nu eenmaal zo gewend is. Daardoor zijn NoSQL databases nog niet zover doorgedrongen als dat men zou verwachten.

## Security

In april 2016 kwam het volgende bericht binnen:
*“Het CERT team van de OU heeft … via SURFcert van het Shadowserver project (Zie: https://mongodbscan.shadowserver.org/ ) diverse meldingen binnengekregen over MongoDB databases die onbeveiligd openstonden naar het Internet.”*

Bij de opgeleverde versie van S3Server draait MongoDB in ‘auth’-mode en is MongoDB toegankelijk via een afwijkend poortnummer. Hiermee zijn voldoende maatregelen getroffen m.b.t. bovenstaand bericht.
Voor meer details, zie de opgeleverde installatiehandleiding.

## Een configuratietip

In de AppConfig-klasse van S3Server is de koppeling met MongoDB zodanig geconfigureerd, dat bij opslag van data in MongoDB geen metadata wordt opgeslagen. Dit komt door de huidige implementatie van de method mongoTemplate():

```
@Override
@Bean(name="template")
MongoTemplate mongoTemplate() throws Exception {

    // We willen geen "_class" veld in de MongoDB collections zien!
    def converter = new MappingMongoConverter(mongoDbFactory(),
            new MongoMappingContext())
    converter.typeMapper = new DefaultMongoTypeMapper(null)
    new MongoTemplate(mongoDbFactory(), converter)
}
```

Als men deze implementatie wijzigt naar:

```
@Override
@Bean(name="template")
MongoTemplate mongoTemplate() throws Exception {
    new MongoTemplate(mongoDbFactory())
}
```

Dan ziet men dat bij nieuw opgeslagen documents een extra veld wordt toegevoegd, dat naar de overeenkomende Java/Groovy-klasse verwijst.
Aan het begin van de realisatie van S3Server werd dit gezien als ‘gegevensvervuiling’. Maar als men de suggestie uit paragraaf 8.2.2 zou volgen, dan blijkt dit gedrag van grote waarde. Het extra opgeslagen veld geeft dan immers aan, op welke subklassen van S3LocationPolicy de betreffende opgeslagen locatiepolicies dan betrekking hebben.

## Documentatie

Officiële MongoDB 3.2.4 documentatie:
https://docs.mongodb.com/v3.2/

# Bijlage 6 – Gebruikte buildtools: Maven en Gradle

Per 2016 is Maven de meest gebruikte buildtool in de Javawereld. Iedere Java-ontwikkelaar die zich bezighoudt met enterprise grade systeemontwikkeling, heeft wel eens met Maven te maken gehad.

De gebruikte ontwikkelomgeving voor S3Server (Spring Tool Suite) biedt out-of-the-box ondersteuning voor Maven. Voor Android ligt dit iets anders. Google biedt Android Studio aan. Dit pakket is gebaseerd op de IntelliJ ontwikkelomgeving. Binnen Android Studio wordt Gradle als standaard buildtool gebruikt.

Maven kent een stringente build lifecycle en staat bekend om zijn uitgebreide (lees: bulky) XML-configuratie mogelijkheden. Een sterk punt van Maven is het ingebouwde dependencymanagement, dat tevens transitieve dependencies correct voor de ontwikkelaar afhandeld.

Gradle is gebaseerd op Groovy en is compatible met het dependencymanagement van Maven. Daarnaast neemt het gebruik van Gradle langzaam maar zeker toe. Wellicht dat het vervangen van Maven door Gradle daardoor de beste optie is om het S3-systeem eenvoudiger onderhoudbaar te maken.

Documentatie Maven:
http://maven.apache.org/guides/

Documentatie Gradle:
https://gradle.org/documentation/

Twee recente artikelen met vergelijkingen tussen Maven en Gradle:
https://dzone.com/articles/maven-vs-gradle-one-year-later
https://programmingmitra.blogspot.nl/2016/05/java-build-tools-comparisons-ant-vs.html